ATAC-Seq pipeline v1 specifications

Sep 18th, 2019

0a. FASTQ read adaptor detecting

ATAC-seq parameters: function detect_adapter()

Program(s)	• detect_adapter.py in GGR_code (https://github.com/nboley/GGR_code) (https://github.com/kundajelab/TF_chipseq_pipeline/blob/master/utils/detect_adapter.py) • modules/align_trim_adapters.bds
Input(s)	• Input: $fastq
Output(s)	• log with detected adaptors: $log
Commands	log = "$prefix.adapter.txt" python3 $script_dir/GGR_code/scripts/detect_adapter.py $fastq > $log # parse log to take 3rd column in the first line of the adapter table in $log
QC to report	Output from last command
Status	Frozen

0b. FASTQ read adaptor trimming

ATAC-seq parameters: function trim_adapters()

■

Program(s)	• cutadapt 1.9.1 • **modules/align_trim_adapters.bds**
Input(s)	• Input: $fastq • Adapter sequence: $adapter • Adapter error rate: $adapter_err_rate (0.2 by default)
Output(s)	• Trimmed fastq $trimmed_fastq
Commands	trimmed_fastq = "$prefix.trim.fastq.gz" cutadapt -m 5 -e $adapter_err_rate -a $adapter $fastq | gzip -nc > $trimmed_fastq
QC to report	
Status	Frozen

1a. Read alignment (Bowtie2 aligner)

Single-End ATAC-seq parameters: function _bowtie2()

■

Program(s)	• Bowtie2 version 2.2.6 • SAMtools 1.7 • SAMstats (version 0.2.1)
Input(s)	• Input: $fastq • Bowtie2 index: $bwt2_idx • # alignments reported for multimapping (4 for ENCODE3 by default): $multimapping
Output(s)	• BAM file $bam • mapping stats from flagstat (SAMstats) $flagstat_qc • mito-free BAM $non_mito_bam
Commands	bam = "$prefix.bam" log = "$prefix.align.log" flagstat_qc = “$prefix.flagstat.qc” # If $multimapping=0 i.e. unique mapping reads only bowtie2 --mm -x $bwt2_idx --threads $nth_bwt2 -U <(zcat -f $fastq) 2> $log | \ samtools view -Su /dev/stdin | samtools sort - $prefix # else i.e allow reads that have at max $mutimapping hits bowtie2 -k $((multimapping+1)) --mm -x $bwt2_idx --threads $nth_bwt2 -U <(zcat -f $fastq) 2> $log | \ samtools view -Su /dev/stdin | samtools sort - $prefix samtools sort -n --threads 10 ${bam} -O SAM | SAMstats --sorted_sam_file - --outf ${flagstat_qc} non_mito_bam = "$prefix.non_mito.bam" # make a mito-free BAM to calculate fraction of mito reads in BAM later non_mito_bam="$prefix.non_mito.bam" nth=2 samtools idxstats ${bam} \| cut -f 1 \| grep -v -P "^chrM$" | xargs samtools view ${bam} -@ ${nth} -b> ${non_mito_bam}
QC to report	Output from last command i.e. samtools flagstat
Status	Frozen

Paired-End ATAC-seq parameters : function _bowtie2_PE()

■

Program(s)	• Bowtie2 version 2.2.6 • SAMtools 1.7 • SAMstats (version 0.2.1)
Input(s)	• Input: $fastq1, $fastq2 • Bowtie2 index: $bwt2_idx

	• # alignments reported for multimapping (4 by default): $multimapping
Output(s)	• BAM file $bam • mapping stats from flagstat (SAMstats) $flagstat_qc • mito-free BAM $non_mito_bam
Commands	bam = "$prefix.bam" log = "$prefix.align.log" flagstat_qc = "$prefix.flagstat.qc" # If $multimapping=0 i.e. unique mapping reads only bowtie2 -X2000 --mm --threads $nth_bwt2 -x $bwt2_idx \ -1 $fastq1 -2 $fastq2 \ 2>$log | \ samtools view -Su /dev/stdin | samtools sort - $prefix # else i.e allow reads that have at max $mutimapping hits bowtie2 -k $((multimapping+1) -X2000 --mm --threads $nth_bwt2 -x $bwt2_idx \ -1 $fastq1 -2 $fastq2 \ 2>$log | \ samtools view -Su /dev/stdin | samtools sort - $prefix samtools sort -n --threads 10 ${bam} -O SAM | SAMstats --sorted_sam_file - --outf ${flagstat_qc} # make a mito-free BAM to calculate fraction of mito reads in BAM later non_mito_bam="$prefix.non_mito.bam" nth=2 samtools idxstats ${bam} \| cut -f 1 \| grep -v -P "^chrM$" | xargs samtools view ${bam} -@ ${nth} -b> ${non_mito_bam}
QC to report	Output from last command i.e. samtools flagstat
Comment	
Status	Frozen

1c. Fraction of mitochondrial reads

function _frac_mito()

Program(s)	• SAMstats (version 0.2.1)
Input(s)	• Input: $non_mito_bam, $mito_bam
Output(s)	• frac_mito QC (Rn, Rm, fraction)
Commands	samtools sort -n --threads 10 ${non_mito_bam} -O SAM | SAMstatspython SAMstats.sort.stat.filter.py --sorted_sam_file - --outf ${non_mito_samstat_qc} Rn = number of MAPPED READS in ${non_mito_samstat_qc} samtools sort -n --threads 10 ${mito_bam} -O SAM | SAMstatspython SAMstats.sort.stat.filter.py

	--sorted_sam_file - --outf ${mito_samstat_qc} Rm = number of MAPPED READS in ${mito_samstat_qc} Fraction of mito reads = Rm / (Rm + Rn)
QC to report	Rn Rm Fraction = Rm / (Rn+Rm)
Comment	
Status	Frozen

1b. Post-alignment filtering

Single-End ATAC-seq parameters: function _dedup_bam()

- Remove reads unmapped, not primary alignment, reads failing platform, duplicates (-F 1804)
- If max. number of multimappers is activated then use explicit multimapping filtering code
- If unique mapping is activated then remove multi-mapped reads (i.e. those with MAPQ < 30, using -q in SAMtools)
  - http://samtools.sourceforge.net/
  - Remove PCR duplicates (using Picard's MarkDuplicates)
  - PICARD: http://picard.sourceforge.net/command-line-overview.shtml#MarkDuplicates

Program(s)	• SAMtools (1.7) • MarkDuplicates (Picard - latest version 1.126) • bedtools 2.26
Input(s)	• Raw BAM file ${RAW_BAM_FILE}
Output(s)	• Filtered deduped position sorted BAM and index file ${FINAL_BAM_FILE} ${FINAL_BAM_INDEX_FILE} • Flagstat Metric for filtered BAM file ${FINAL_BAM_FILE_MAPSTATS} • Duplication metrics from MarkDuplicates ${DUP_FILE_QC} • Library complexity measures ${PBC_FILE_QC}
Commands	# ============================ # Remove unmapped, mate unmapped # not primary alignment, reads failing platform # ================== # for multimapping > 0 (i.e allow reads that have at max $mutimapping hits) QNAME_SORT_BAM_FILE="${OFPREFIX}.qnmsrt.bam" FILT_BAM_PREFIX="${OFPREFIX}.filt" FILT_BAM_FILE="${FILT_BAM_PREFIX}.bam" samtools sort -n ${RAW_BAM_FILE} -o ${QNAME_SORT_BAM_FILE} samtools view -h ${QNAME_SORT_BAM_FILE} | $(which assign_multimappers.py) -k $multimapping | samtools view -F 1804 -Su /dev/stdin | samtools sort /dev/stdin -o ${FILT_BAM_FILE} rm -f ${QNAME_SORT_BAM_FILE} # for multimapping == 0: (# for unique mapping reads only) MAPQ_THRESH=30 samtools view -F 1804 -q ${MAPQ_THRESH} -u ${RAW_BAM_FILE} | samtools sort /dev/stdin -o ${FILT_BAM_FILE} -T ${FILT_BAM_FILE_PREFIX} # ======================== # Mark duplicates # ====================== module add picard-tools/1.126 TMP_FILT_BAM_FILE="${FILT_BAM_PREFIX}.dupmark.bam" MARKDUP="/srv/gs1/software/picard-tools/1.126/MarkDuplicates.jar"

```
DUP_FILE_QC="${FILT_BAM_PREFIX}.dup.qc" # QC file

java -Xmx4G -jar ${MARKDUP} INPUT=${FILT_BAM_FILE} OUTPUT=${TMP_FILT_BAM_FILE}
METRICS_FILE=${DUP_FILE_QC} VALIDATION_STRINGENCY=LENIENT ASSUME_SORTED=true
REMOVE_DUPLICATES=false

mv ${TMP_FILT_BAM_FILE} ${FILT_BAM_FILE}

# ============================
# Remove duplicates
# Index final position sorted BAM
# ============================
FINAL_BAM_PREFIX="${OFPREFIX}..nodup"
FINAL_BAM_FILE="${FINAL_BAM_PREFIX}.bam" # To be stored
FINAL_BAM_INDEX_FILE="${FINAL_BAM_FILE}.bai" # To be stored
FINAL_BAM_FILE_MAPSTATS="${FINAL_BAM_PREFIX}.flagstat.qc" # QC file

samtools view -F 1804 -b ${FILT_BAM_FILE} > ${FINAL_BAM_FILE}

# Index Final BAM file
samtoolsindex ${FINAL_BAM_FILE}

samtools sort -n --threads 10 ${FINAL_BAM_FILE} -O SAM  | SAMstats --sorted_sam_file -  --outf
${FINAL_BAM_FILE_MAPSTATS}

# ============================
# Compute library complexity
# ============================
# sort by position and strand
# Obtain unique count statistics

module add bedtools/2.26

PBC_FILE_QC="${FINAL_BAM_PREFIX}.pbc.qc"

# PBC File output
# TotalReadPairs [tab] DistinctReadPairs [tab] OneReadPair [tab] TwoReadPairs [tab] NRF=Distinct/Total [tab]
PBC1=OnePair/Distinct [tab] PBC2=OnePair/TwoPair

bedtools bamtobed -i ${FILT_BAM_FILE} | awk 'BEGIN{OFS="\t"}{print $1,$2,$3,$6}' | grep -v 'chrM' | sort | uniq
-c | awk 'BEGIN{mt=0;m0=0;m1=0;m2=0} ($1==1){m1=m1+1} ($1==2){m2=m2+1} {m0=m0+1} {mt=mt+$1}
END{printf "%d\t%d\t%d\t%d\t%f\t%f\t%f\n",mt,m0,m1,m2,m0/mt,m1/m0,m1/m2}' > ${PBC_FILE_QC}

rm ${FILT_BAM_FILE}



###### assign_multimappers.py

import sys
import random
import argparse

def parse_args():
        '''
        Gives options
        '''
        parser = argparse.ArgumentParser(description='Saves reads below a alignment threshold and
discards all others')
        parser.add_argument('-k', help='Alignment number cutoff')
```

	parser.add_argument('--paired-end', dest='paired_ended', action='store_true', help='Data is paired-end') args = parser.parse_args() alignment_cutoff = int(args.k) paired_ended = args.paired_ended return alignment_cutoff, paired_ended if __name__ == "__main__": ''' Runs the filtering step of choosing multimapped reads ''' [alignment_cutoff, paired_ended] = parse_args() if paired_ended: alignment_cutoff = int(alignment_cutoff) * 2 # Store each line in sam file as a list of reads, # where each read is a list of elements to easily # modify or grab things current_reads = [] current_qname = '' for line in sys.stdin: read_elems = line.strip().split('\t') if read_elems[0].startswith('@'): sys.stdout.write(line) continue # Keep taking lines that have the same qname if read_elems[0] == current_qname: # Add line to current reads current_reads.append(line) pass else: # Discard if there are more than the alignment cutoff if len(current_reads) >= alignment_cutoff: current_reads = [line] current_qname = read_elems[0] elif len(current_reads) > 0: # Just output all reads, which are then filtered with # samtools for read in current_reads: sys.stdout.write(str(read)) # And then discard current_reads = [line] current_qname = read_elems[0] else: # First read in file current_reads.append(line) current_qname = read_elems[0]
QC to report	(1) Flagstat output from Final filtered deduped BAM file ${FINAL_BAM_FILE_MAPSTATS} (2) PICARD MarkDup output ${DUP_FILE_QC} http://sourceforge.net/apps/mediawiki/picard/index.php?title=Main_Page#Q:_What_is_meaning_of_the_histogr

	am_produced_by_MarkDuplicates.3F (3) Library complexity measures ${PBC_FILE_QC} ● Format of file TotalReads [tab] DistinctReads [tab] OneRead [tab] TwoRead [tab] NRF=Distinct/Total [tab] PBC1=OnePair/Distinct [tab] PBC2=OnePair/TwoPair ● NRF (non redundant fraction) ● PBC1 (PCR Bottleneck coefficient 1) ● PBC2 (PCR Bottleneck coefficient 2) ● PBC1 is the primary measure. Provisionally, ○ 0-0.5 is severe bottlenecking ○ 0.5-0.8 is moderate bottlenecking ○ 0.8-0.9 is mild bottlenecking ○ 0.9-1.0 is no bottlenecking.
Status	Frozen

Paired-End ATAC-seq parameters: function _dedup_bam_PE()

- Remove reads unmapped, mate unmapped, not primary alignment, reads failing platform, duplicates (-F 524), reads mapping to chrM.
- Retain properly paired reads -f 2
- If max. number of multimappers is activated then use explicit multimapping filtering code
- If unique mapping is activated then remove multi-mapped reads (i.e. those with MAPQ < 30, using -q in SAMtools)
    - http://samtools.sourceforge.net/
- Remove PCR duplicates (using Picard's MarkDuplicates or FixSeq)
    - PICARD: http://picard.sourceforge.net/command-line-overview.shtml#MarkDuplicates

Program(s)	• SAMtools (1.7) • MarkDuplicates (Picard - latest version 1.126) • bedtools 2.26
Input(s)	• Raw BAM file ${RAW_BAM_FILE}
Output(s)	• Filtered deduped position sorted BAM and index file ${FINAL_BAM_FILE} ${FINAL_BAM_INDEX_FILE} • Filtered deduped name sorted BAM file ${FINAL_NMSRT_BAM_FILE} • Flagstat Metric for filtered BAM file ${FINAL_BAM_FILE_MAPSTATS} • Duplication metrics from MarkDuplicates ${DUP_FILE_QC} • Library complexity measures ${PBC_FILE_QC}
Commands	# ============================ # Remove unmapped, mate unmapped # not primary alignment, reads failing platform # Only keep properly paired reads # Obtain name sorted BAM file # ================== FILT_BAM_PREFIX="${OFPREFIX}.filt" FILT_BAM_FILE="${FILT_BAM_PREFIX}.bam" TMP_FILT_BAM_PREFIX="tmp.${FILT_BAM_PREFIX}.nmsrt" TMP_FILT_BAM_FILE="${TMP_FILT_BAM_PREFIX}.bam" ## for multimapping > 0: samtools view -F 524 -f 2 -u ${RAW_BAM_FILE} | samtools -n /dev/stdin -o ${TMP_FILT_BAM_FILE} TMP_FILT_FIXMATE_BAM_FILE="${TMP_FILT_BAM_PREFIX}.fixmate.bam" samtools view -h ${TMP_FILT_BAM_FILE} | assign_multimappers.py -k $multimapping --paired-end | samtools fixmate -r /dev/stdin ${TMP_FILT_FIXMATE_BAM_FILE} ## for multimapping==0: MAPQ_THRESH=30 samtools view -F 1804 -f 2 -q ${MAPQ_THRESH} -u ${RAW_BAM_FILE} | samtools sort -n /dev/stdin -o ${TMP_FILT_BAM_FILE} samtools fixmate -r ${TMP_FILT_BAM_FILE} ${TMP_FILT_FIXMATE_BAM_FILE} # Remove orphan reads (pair was removed) # and read pairs mapping to different chromosomes # Obtain position sorted BAM

```
samtools view -F 1804 -f 2 -u ${TMP_FILT_FIXMATE_BAM_FILE} | samtools sort /dev/stdin -o
${FILT_BAM_FILE}

rm  ${TMP_FILT_FIXMATE_BAM_FILE}

rm ${TMP_FILT_BAM_FILE}

# =============
# Mark duplicates
# =============
module add picard-tools/1.126

TMP_FILT_BAM_FILE="${FILT_BAM_PREFIX}.dupmark.bam"
MARKDUP="/srv/gs1/software/picard-tools/1.126/MarkDuplicates.jar"
DUP_FILE_QC="${FILT_BAM_PREFIX}.dup.qc"

java -Xmx4G -jar ${MARKDUP} INPUT=${FILT_BAM_FILE} OUTPUT=${TMP_FILT_BAM_FILE}
METRICS_FILE=${DUP_FILE_QC} VALIDATION_STRINGENCY=LENIENT
ASSUME_SORTED=true REMOVE_DUPLICATES=false

mv ${TMP_FILT_BAM_FILE} ${FILT_BAM_FILE}

# ============================
# Remove duplicates
# Index final position sorted BAM
# Create final name sorted BAM
# ============================
FINAL_BAM_PREFIX="${OFPREFIX}.nodup"
FINAL_BAM_FILE="${FINAL_BAM_PREFIX}.bam" # To be stored
FINAL_BAM_INDEX_FILE="${FINAL_BAM_FILE}.bai"
FINAL_BAM_FILE_MAPSTATS="${FINAL_BAM_PREFIX}.flagstat.qc" # QC file

samtools view -F 1804 -f 2 -b ${FILT_BAM_FILE} > ${FINAL_BAM_FILE}


# Index Final BAM file
samtools index ${FINAL_BAM_FILE}

samtools sort -n --threads 10 ${FINAL_BAM_FILE} -O SAM  | SAMstats --sorted_sam_file -
--outf ${FINAL_BAM_FILE_MAPSTATS}

# ============================
# Compute library complexity
# ============================
# Sort by name
# convert to bedPE and obtain fragment coordinates
# sort by position and strand
# Obtain unique count statistics

module add bedtools/2.26

PBC_FILE_QC="${FINAL_BAM_PREFIX}.pbc.qc"

# TotalReadPairs [tab] DistinctReadPairs [tab] OneReadPair [tab] TwoReadPairs [tab]
NRF=Distinct/Total [tab] PBC1=OnePair/Distinct [tab] PBC2=OnePair/TwoPair


samtools sort -n ${FILT_BAM_FILE} -o ${OFPREFIX}.srt.tmp.bam
```

```
bedtools bamtobed -bedpe -i ${OFPREFIX}.srt.tmp.bam | awk 'BEGIN{OFS="\t"}{print
$1,$2,$4,$6,$9,$10}' | grep -v 'chrM' | sort | uniq -c | awk 'BEGIN{mt=0;m0=0;m1=0;m2=0}
($1==1){m1=m1+1} ($1==2){m2=m2+1} {m0=m0+1} {mt=mt+$1} END{printf
"%d\t%d\t%d\t%d\t%f\t%f\t%f\n",mt,m0,m1,m2,m0/mt,m1/m0,m1/m2}' > ${PBC_FILE_QC}
rm ${OFPREFIX}.srt.tmp.bam


rm ${FILT_BAM_FILE}

###### assign_multimappers.py

import sys
import random
import argparse

def parse_args():
        '''
        Gives options
        '''
        parser = argparse.ArgumentParser(description='Saves reads below a alignment
threshold and discards all others')
        parser.add_argument('-k', help='Alignment number cutoff')
        parser.add_argument('--paired-end', dest='paired_ended', action='store_true',
help='Data is paired-end')
        args = parser.parse_args()
        alignment_cutoff = int(args.k)
        paired_ended = args.paired_ended

        return alignment_cutoff, paired_ended


if __name__ == "__main__":
        '''
        Runs the filtering step of choosing multimapped reads
        '''

        [alignment_cutoff, paired_ended] = parse_args()

        if paired_ended:
        alignment_cutoff = int(alignment_cutoff) * 2

        # Store each line in sam file as a list of reads,
        # where each read is a list of elements to easily
        # modify or grab things
        current_reads = []
        current_qname = ''

        for line in sys.stdin:

        read_elems = line.strip().split('\t')

        if read_elems[0].startswith('@'):
        sys.stdout.write(line)
        continue

        # Keep taking lines that have the same qname
        if read_elems[0] == current_qname:
        # Add line to current reads
        current_reads.append(line)
        pass
```

	else: # Discard if there are more than the alignment cutoff if len(current_reads) >= alignment_cutoff: current_reads = [line] current_qname = read_elems[0] elif len(current_reads) > 0: # Just output all reads, which are then filtered with # samtools for read in current_reads: sys.stdout.write(str(read)) # And then discard current_reads = [line] current_qname = read_elems[0] else: # First read in file current_reads.append(line) current_qname = read_elems[0]
QC to report	(1) Flagstat output from Final filtered deduped BAM file ${FINAL_BAM_FILE_MAPSTATS} (2) PICARD MarkDup output ${DUP_FILE_QC} http://sourceforge.net/apps/mediawiki/picard/index.php?title=Main_Page#Q:_What_is_meaning_of_the_histogram_produced_by_MarkDuplicates.3F (3) Library complexity measures ${PBC_FILE_QC} ● Format of file TotalReadPairs [tab] DistinctReadPairs [tab] OneReadPair [tab] TwoReadPairs [tab] NRF=Distinct/Total [tab] PBC1=OnePair/Distinct [tab] PBC2=OnePair/TwoPair ● NRF (non redundant fraction) ● PBC1 (PCR Bottleneck coefficient 1) ● PBC2 (PCR Bottleneck coefficient 2) ● PBC1 is the primary measure. Provisionally, ○ 0-0.5 is severe bottlenecking ○ 0.5-0.8 is moderate bottlenecking ○ 0.8-0.9 is mild bottlenecking ○ 0.9-1.0 is no bottlenecking.
Status	Frozen

2a. Convert SE BAM to tagAlign (BED 3+3 format) : function _bam_to_tag()

Program(s)	• bedtools 2.26 • gawk • shuf
Input(s)	• Filtered BAM file ${FINAL_BAM_FILE}
Output(s)	• tagAlign file ${FINAL_TA_FILE} • Subsampled tagAlign file for CC analysis ${SUBSAMPLED_TA_FILE}
Commands	# ==================== # Create tagAlign file # ==================== module add bedtools/2.26 # Create SE tagAlign file FINAL_TA_FILE="${FINAL_BAM_PREFIX}.SE.tagAlign.gz" bedtools bamtobed -i ${FINAL_BAM_FILE} | awk 'BEGIN{OFS="\t"}{$4="N";$5="1000";print $0}' | gzip -nc > ${FINAL_TA_FILE} # ================================ # Subsample tagAlign file # ================================ NREADS=25000000 SUBSAMPLED_TA_FILE="${OFPREFIX}.filt.nodup.sample.$((NREADS / 1000000))M.SE.tagAlign.gz" zcat ${FINAL_TA_FILE} | grep -v “chrM” | shuf -n ${NREADS} --random-source=<(openssl enc -aes-256-ctr -pass pass:$(zcat -f ${FINAL_TA_FILE} | wc -c) -nosalt </dev/zero 2>/dev/null) | gzip -nc > ${SUBSAMPLED_TA_FILE}
QC to report	None
Status	Frozen

2a. Convert PE BAM to tagAlign (BED 3+3 format): function _bam_to_bedpe() and function _bam_to_tag()

Program(s)	• bedtools 2.26 • gawk • shuf
Input(s)	• Filtered BAM file ${FINAL_BAM_FILE}
Output(s)	• tagAlign file ${FINAL_TA_FILE} • BEDPE file (with read pairs on each line) ${FINAL_BEDPE_FILE} • Subsampled tagAlign file for CC analysis ${SUBSAMPLED_TA_FILE}
Commands	# ==================== # Create tagAlign file # ==================== module add bedtools/2.26 # ================ # Create BEDPE file # ================

	FINAL_BEDPE_FILE="${FINAL_NMSRT_BAM_PREFIX}.bedpe.gz" bedtools bamtobed -bedpe -mate1 -i ${FINAL_NMSRT_BAM_FILE} | gzip -nc > ${FINAL_BEDPE_FILE} # Create final TA file zcat ${FINAL_BEDPE_FILE} | awk 'BEGIN{OFS="\t"}{printf "%s\t%s\t%s\tN\t1000\t%s\n%s\t%s\t%s\tN\t1000\t%s\n",$1,$2,$3,$9,$4,$5,$6,$10}' | \ gzip -nc > ${FINAL_TA_FILE} # ================================ # Subsample tagAlign file # Restrict to one read end per pair for CC analysis # ================================ NREADS=25000000 SUBSAMPLED_TA_FILE="${OFPREFIX}.filt.nodup.sample.$((NREADS / 1000000)).MATE1.tagAlign.gz" zcat ${FINAL_BEDPE_FILE} | grep -v “chrM” | shuf -n ${NREADS} --random-source=<(openssl enc -aes-256-ctr -pass pass:$(zcat -f ${FINAL_TA_FILE} | wc -c) -nosalt </dev/zero 2>/dev/null) | awk 'BEGIN{OFS="\t"}{print $1,$2,$3,"N","1000",$9}' | gzip -nc > ${SUBSAMPLED_TA_FILE}
QC to report	None
Status	Frozen

2b. Calculate Cross-correlation QC scores: function _xcor()

- Code package: https://code.google.com/p/phantompeakqualtools/ (Updated version is imminent)
- Dependencies: unix, bash, R-3.2 and above, gawk, samtools, boost C++ libraries, R packages: SPP (1.14), caTools, snow

Program(s)	• phantompeakqualtools (v1.2.1)
Input(s)	• Subsampled TagAlign file ${SUBSAMPLED_TA_FILE}
Output(s)	• outFile containing NSC/RSC results in tab-delimited file of 11 columns (same file can be appended to from multiple runs) ${CC_SCORES_FILE} • cross-correlation plot ${CC_PLOT_FILE
Commands	CC_SCORES_FILE="${SUBSAMPLED_TA_FILE}.cc.qc" CC_PLOT_FILE="${SUBSAMPLED_TA_FILE}.cc.plot.pdf" # CC_SCORE FILE format # Filename <tab> numReads <tab> estFragLen <tab> corr_estFragLen <tab> PhantomPeak <tab> corr_phantomPeak <tab> argmin_corr <tab> min_corr <tab> phantomPeakCoef <tab> relPhantomPeakCoef <tab> QualityTag Rscript $(which run_spp.R) -c=${SUBSAMPLED_TA_FILE} -p=${NTHREADS} -filtchr=chrM -savp=${CC_PLOT_FILE} -out=${CC_SCORES_FILE} sed -r 's/,[^\t]+//g' ${CC_SCORES_FILE} > temp mv temp ${CC_SCORES_FILE}
QC to report	format:Filename<tab>numReads<tab>estFragLen<tab>corr_estFragLen<tab>PhantomPeak<tab>corr_phantomPeak<tab>argmin_corr<tab>min_corr<tab>phantomPeakCoef<tab>relPhantomPeakCoef<tab>QualityTag • Normalized strand cross-correlation coefficient (NSC) = col9 in outFile • Relative strand cross-correlation coefficient (RSC) = col10 in outFile • Estimated fragment length = col3 in outFile, take the top value • Important columns highlighted, but all/whole file can be stored for display
Status	Frozen

2c. Generate self-pseudoreplicates for each replicate (SE datasets): function _spr()

Program(s)	• UNIX shuf • UNIX split • gawk
Input(s)	• TagAlign file ${FINAL_TA_FILE}
Output(s)	• 2 pseudoreplicate virtual SE tagAlign files ${PR1_TA_FILE} ${PR2_TA_FILE}
Commands	# ======================== # Create pseudoReplicates # ======================== PR_PREFIX="${OFPREFIX}.filt.nodup" PR1_TA_FILE="${PR_PREFIX}.SE.pr1.tagAlign.gz" PR2_TA_FILE="${PR_PREFIX}.SE.pr2.tagAlign.gz" # Get total number of read pairs nlines=$(zcat ${FINAL_TA_FILE} \| wc -l) nlines=$(((nlines + 1) / 2)) # Shuffle and split BED file into 2 equal parts zcat ${FINAL_TA_FILE} \| shuf --random-source=<(openssl enc -aes-256-ctr -pass pass:$(zcat -f ${FINAL_TA_FILE} | wc -c) -nosalt </dev/zero 2>/dev/null) | split -d -l ${nlines} - ${PR_PREFIX} # Will produce ${PR_PREFIX}00 and ${PR_PREFIX}01 # Convert reads into standard tagAlign file gzip -nc “${PR_PREFIX}00" > ${PR1_TA_FILE} rm "${PR_PREFIX}00" gzip -nc “${PR_PREFIX}01" > ${PR2_TA_FILE} rm "${PR_PREFIX}01"
QC to report	None
Status	Frozen

2c. Generate self-pseudoreplicates for each replicate (PE datasets): function _spr_PE()

Program(s)	• UNIX shuf • UNIX split • gawk
Input(s)	• TAGALIGN file ${FINAL_TA_FILE}
Output(s)	• 2 pseudoreplicate virtual SE tagAlign files ${PR1_TA_FILE} ${PR2_TA_FILE}
Commands	# ======================== # Create pseudoReplicates # ======================== PR_PREFIX="${OFPREFIX}.filt.nodup" PR1_TA_FILE="${PR_PREFIX}.PE2SE.pr1.tagAlign.gz" PR2_TA_FILE="${PR_PREFIX}.PE2SE.pr2.tagAlign.gz" joined=”temp.bedpe” # Make temporary fake BEDPE file from FINAL_TA_FILE zcat ${FINAL_TA_FILE} | sed 'N;s/\n/\t/' | gzip -nc > $joined

	# Get total number of read pairs nlines=$(zcat ${joined} \| wc -l) nlines=$(((nlines + 1) / 2)) # Shuffle and split BEDPE file into 2 equal parts zcat -f ${joined} \| shuf --random-source=<(openssl enc -aes-256-ctr -pass pass:$(zcat -f ${FINAL_TA_FILE} | wc -c) -nosalt </dev/zero 2>/dev/null) | split -d -l ${nlines} - ${PR_PREFIX} # Will produce ${PR_PREFIX}00 and ${PR_PREFIX}01 # Convert fake BEDPE to reads into standard tagAlign file awk 'BEGIN{OFS="\t"}{printf "%s\t%s\t%s\t%s\t%s\t%s\n%s\t%s\t%s\t%s\t%s\t%s\n",$1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12}' "${PR_PREFIX}00" | gzip -nc > ${PR1_TA_FILE} rm "${PR_PREFIX}00" awk 'BEGIN{OFS="\t"}{printf "%s\t%s\t%s\t%s\t%s\t%s\n%s\t%s\t%s\t%s\t%s\t%s\n",$1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12}' "${PR_PREFIX}01" | gzip -nc > ${PR2_TA_FILE} rm "${PR_PREFIX}01" rm $joined
QC to report	None
Status	Frozen

2d. Generate pooled dataset and pooled-pseudoreplicates : function _ppr()

Program(s)	• gzip
Input(s)	• Final tagalign files for all replicates ${REP1_TA_FILE} ${REP2_TA_FILE} obtained from ${FINAL_TA_FILE} of the step 2a. • Self-consistency pseudoreplicates for all replicates REP*_PR1_TA_FILE and REP*_PR2_TA_FILE obtained from ${PPR1_TA_FILE} ${PPR2_TA_FILE} of step 2c.
Output(s)	• Pooled tagAlign file ${POOLED_TA_FILE} • 2 pooled-pseudoreplicate tagAlign files
Commands	# ======================== # Create pooled datasets # ======================== REP1_TA_FILE="${DATASET_PREFIX}.Rep1.tagAlign.gz" REP2_TA_FILE="${DATASET_PREFIX}.Rep2.tagAlign.gz" POOLED_TA_FILE="${DATASET_PREFIX}.Rep0.tagAlign.gz" zcat ${REP1_TA_FILE} ${REP2_TA_FILE} | gzip -nc > ${POOLED_TA_FILE} # ======================== # Create pooled pseudoreplicates # ======================== REP1_PR1_TA_FILE="${DATASET_PREFIX}.Rep1.pr1.tagAlign.gz" REP1_PR2_TA_FILE="${DATASET_PREFIX}.Rep1.pr2.tagAlign.gz" REP2_PR1_TA_FILE="${DATASET_PREFIX}.Rep2.pr1.tagAlign.gz" REP2_PR2_TA_FILE="${DATASET_PREFIX}.Rep2.pr2.tagAlign.gz" PPR1_TA_FILE="${DATASET_PREFIX}.Rep0.pr1.tagAlign.gz" PPR2_TA_FILE="${DATASET_PREFIX}.Rep0.pr1.tagAlign.gz"

	zcat ${REP1_PR1_TA_FILE} ${REP2_PR1_TA_FILE} | gzip -nc > ${PPR1_TA_FILE} zcat ${REP1_PR2_TA_FILE} ${REP2_PR2_TA_FILE} | gzip -nc > ${PPR2_TA_FILE}
QC to report	None
Status	Frozen

2e. TN5 shifting of tagaligns for ATAC Seq : _tn5_shift()

Program(s)	• gawk
Input(s)	• Tagalign file $tag
Output(s)	• Shifted tagalign $shifted_tag
Commands	shifted_tag = "$prefix.tn5.tagAlign.gz" zcat $tag | awk -F $'\t' 'BEGIN {OFS = FS}{ if ($6 == "+") {$2 = $2 + 4} else if ($6 == "-") {$3 = $3 - 5} print $0}' | gzip -nc > $shifted_tag
QC to report	None
Status	Frozen

2f. Calculate Jensen-Shannon distance (JSD): _jsd()

Program(s)	• deeptools (v3.3.0) plotFingerprint
Input(s)	• Filtered/deduped BAM ${NODUP_BAM_REP1}, ${NODUP_BAM_REP2} • Blacklist BED file ${BLACKLIST} • mapping quality threshold (30 for bwa, 30 for bowtie2) ${MAPQ_THRESH}
Output(s)	• JSD Plot ${JSD_PLOT} • JSD log ${JSD_LOG}: column description: https://github.com/deeptools/deepTools/blob/master/deeptools/plotFingerprint.py#L454 ○ 7 col including synthetic JS distance ■ auc, syn_auc, x_intercept, syn_x_intercept, elbow_pt, syn_elbow_pt, syn_jsd
Commands	NTH=4 # number of threads # BAMs are blacklist-filtered first for each replicate NODUP_BFILT_BAM_REP1=${NODUP_BAM_REP1}.bfilt.bam NODUP_BFILT_BAM_REP2=${NODUP_BAM_REP2}.bfilt.bam bedtools intersect -nonamecheck -v -abam ${NODUP_BAM_REP1} -b ${BLACKLIST} > ${NODUP_BFILT_BAM_REP1}' bedtools intersect -nonamecheck -v -abam ${NODUP_BAM_REP2} -b ${BLACKLIST} > ${NODUP_BFILT_BAM_REP2}' LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8 plotFingerprint -b ${NODUP_BFILT_BAM_REP1} ${NODUP_BFILT_BAM_REP2} --labels rep1 rep2 --outQualityMetrics ${JSD_LOG} --minMappingQuality ${MAPQ_THRESH} -T "Fingerprints of different samples" --numberOfProcessors ${NTH} --plotFile ${JSD_PLOT}

QC to report	None
Status	Frozen

2g. Calculate GC bias: _gc_bias()

Program(s)	• CollectGcBiasMetrics (Picard - version 1.126) • python packages: pandas, matplotlib
Input(s)	• Filtered/deduped BAM ${NODUP_BAM} • Reference genome fasta ${REF_FA}
Output(s)	• GC bias Plot ${GC_BIAS_PLOT} • GC bias log ${GC_BIAS_LOG}
Commands	# we don't use plot directly generated from picard # we process picard's text output and make a plot java -Xmx6G -XX:ParallelGCThreads=1 -jar \Tss picard.jar \ CollectGcBiasMetrics R=${REF_FA} I=${NODUP_BAM} O=${GC_BIAS_LOG} \ USE_JDK_DEFLATER=TRUE USE_JDK_INFLATER=TRUE \ VERBOSITY=ERROR QUIET=TRUE \ ASSUME_SORTED=FALSE \ CHART=${GC_BIAS_PLOT} S=summary.txt # use ${GC_BIAS_LOG} into the following pyhton script # data_file: ${GC_BIAS_LOG} # prefix: any good prefix for output file name def plot_gc(data_file, prefix): ''' Replot the Picard output as png file to put into the html ''' # Load data data = pd.read_table(data_file, comment="#") # Plot the data fig = plt.figure() ax = fig.add_subplot(111) plt.xlim((0, 100)) lin1 = ax.plot(data['GC'], data['NORMALIZED_COVERAGE'], label='Normalized coverage', color='r') ax.set_ylabel('Normalized coverage') ax2 = ax.twinx() lin2 = ax2.plot(data['GC'], data['MEAN_BASE_QUALITY'], label='Mean base quality at GC%', color='b') ax2.set_ylabel('Mean base quality at GC%') ax3 = ax.twinx() lin3 = ax3.plot(data['GC'], data['WINDOWS']/np.sum(data['WINDOWS']), label='Windows at GC%', color='g') ax3.get_yaxis().set_visible(False)

	lns = lin1 + lin2 + lin3 labs = [l.get_label() for l in lns] ax.legend(lns, labs, loc='best') # plot_img = BytesIO() # fig.savefig(plot_img, format='png') prefix = data_file.rstrip('.gc.txt') plot_png = prefix + '.gc_plot.png' fig.savefig(plot_png, format='png')
QC to report	None
Status	Frozen

2h. Fragment length statistics (PE only): _frag_len_stat()

Program(s)	• CollectInsertSizeMetrics (Picard - version 1.126) • python packages: numpy, scipy.signal, matplotlib
Input(s)	• Filtered/deduped BAM ${NODUP_BAM}
Output(s)	• Nucleosomeal QC ${NUCLEOSOMAL_QC} • Fragment length distribution plot ${FRAGLEN_DIST_PLOT}
Commands	# intermediate files # ${INSERT_DATA} # ${INSERT_PLOT} java -Xmx6G -XX:ParallelGCThreads=1 -jar \ picard.jar \ CollectInsertSizeMetrics \ INPUT=${NODUP_BAM} OUTPUT=${INSERT_DATA} H=${INSERT_PLOT} \ VERBOSITY=ERROR QUIET=TRUE \ USE_JDK_DEFLATER=TRUE USE_JDK_INFLATER=TRUE \ W=1000 STOP_AFTER=5000000 # using an intermediate file ${INSERT_DATA} # generate Nucleosomeal QC ${NUCLEOSOMAL_QC} and # Fragment length distribution plot ${FRAGLEN_DIST_PLOT} # OUTPUT_PREFIX: output file prefix nucleosomal_qc = fragment_length_qc(read_picard_histogram(insert_data), OUTPUT_PREFIX) fraglen_dist_plot = fragment_length_plot(insert_data, OUTPUT_PREFIX) QCResult = namedtuple('QCResult', ['metric', 'qc_pass', 'message']) INF = float("inf") class QCCheck(object): def __init__(self, metric):

```
        self.metric = metric

    def check(self, value):
        return True

    def message(self, value, qc_pass):
        return ('{}\tOK'.format(value) if qc_pass
                else '{}\tFailed'.format(value))

    def __call__(self, value):
        qc_pass = self.check(value)
        return QCResult(self.metric, qc_pass, self.message(value, qc_pass))


class QCIntervalCheck(QCCheck):
    def __init__(self, metric, lower, upper):
        super(QCIntervalCheck, self).__init__(metric)
        self.lower = lower
        self.upper = upper

    def check(self, value):
        return self.lower <= value <= self.upper

    def message(self, value, qc_pass):
        return ('{}\tOK'.format(value) if qc_pass else
                '{}\tout of range [{}, {}]'.format(value, self.lower,
                                                   self.upper))


class QCLessThanEqualCheck(QCIntervalCheck):
    def __init__(self, metric, upper):
        super(QCLessThanEqualCheck, self).__init__(metric, -INF, upper)


class QCGreaterThanEqualCheck(QCIntervalCheck):
    def __init__(self, metric, lower):
        super(QCGreaterThanEqualCheck, self).__init__(metric, lower, INF)


class QCHasElementInRange(QCCheck):
    def __init__(self, metric, lower, upper):
        super(QCHasElementInRange, self).__init__(metric)
        self.lower = lower
        self.upper = upper

    def check(self, elems):
        return (len([elem for elem in elems
                     if self.lower <= elem <= self.upper]) > 0)

    def message(self, elems, qc_pass):
        return ('OK' if qc_pass else
                'Cannot find element in range [{}, {}]'.format(
                    self.lower, self.upper))


def read_picard_histogram(data_file):
    with open(data_file) as fp:
        for line in fp:
            if line.startswith('## HISTOGRAM'):
                break
```

```
        data = np.loadtxt(fp, skiprows=1)

    return data

def get_insert_distribution(final_bam, prefix):
    '''
    Calls Picard CollectInsertSizeMetrics
    '''
    logging.info('insert size distribution...')
    insert_data = '{0}.inserts.hist_data.log'.format(prefix)
    insert_plot = '{0}.inserts.hist_graph.pdf'.format(prefix)
    graph_insert_dist = ('java -Xmx6G -XX:ParallelGCThreads=1 -jar '
                         '{3} '
                         'CollectInsertSizeMetrics '
                         'INPUT={0} OUTPUT={1} H={2} '
                         'VERBOSITY=ERROR QUIET=TRUE '
                         'USE_JDK_DEFLATER=TRUE USE_JDK_INFLATER=TRUE '
                         'W=1000 STOP_AFTER=5000000').format(final_bam,
                                                            insert_data,
                                                            insert_plot,
                                                            locate_picard())
    logging.info(graph_insert_dist)
    os.system(graph_insert_dist)
    return insert_data, insert_plot

def fragment_length_qc(data, prefix):
    results = []

    NFR_UPPER_LIMIT = 150
    MONO_NUC_LOWER_LIMIT = 150
    MONO_NUC_UPPER_LIMIT = 300

    # % of NFR vs res
    nfr_reads = data[data[:,0] < NFR_UPPER_LIMIT][:,1]
    percent_nfr = nfr_reads.sum() / data[:,1].sum()
    results.append(
        QCGreaterThanEqualCheck('Fraction of reads in NFR', 0.4)(percent_nfr))

    # % of NFR vs mononucleosome
    mono_nuc_reads = data[
        (data[:,0] > MONO_NUC_LOWER_LIMIT) &
        (data[:,0] <= MONO_NUC_UPPER_LIMIT)][:,1]

    percent_nfr_vs_mono_nuc = (
        nfr_reads.sum() /
        mono_nuc_reads.sum())
    results.append(
        QCGreaterThanEqualCheck('NFR / mono-nuc reads', 2.5)(
            percent_nfr_vs_mono_nuc))

    # peak locations
    pos_start_val = data[0,0] # this may be greater than 0
    peaks = find_peaks_cwt(data[:, 1], np.array([25]))
    nuc_range_metrics = [('Presence of NFR peak', 20 - pos_start_val, 90 - pos_start_val),
                         ('Presence of Mono-Nuc peak', 120 - pos_start_val, 250 - pos_start_val),
                         ('Presence of Di-Nuc peak', 300 - pos_start_val, 500 - pos_start_val)]
    for range_metric in nuc_range_metrics:
        results.append(QCHasElementInRange(*range_metric)(peaks))

    out = prefix + '.nucleosomal.qc'
```

	with open(out, 'w') as fp: for elem in results: fp.write('\t'.join([elem.metric, str(elem.qc_pass), elem.message]) + '\n') return out def fragment_length_plot(data_file, prefix, peaks=None): try: data = read_picard_histogram(data_file) except IOError: return '' except TypeError: return '' fig = plt.figure() plt.bar(data[:, 0], data[:, 1]) plt.xlim((0, 1000)) if peaks: peak_vals = [data[peak_x, 1] for peak_x in peaks] plt.plot(peaks, peak_vals, 'ro') # plot_img = BytesIO() # fig.savefig(plot_img, format='png') plot_png = prefix + '.fraglen_dist.png' fig.savefig(plot_png, format='png') return plot_png
QC to report	None
Status	Frozen

3. Call peaks on replicates, self-pseudoreplicates, pooled data and pooled-pseudoreplicates

Call peaks on all replicates, pooled data, self-pseudoreplicates of each replicate and the pooled-pseudoreplicates using MACS2

3a. Peak calling - MACS2 : function macs2()

- P-value thresholds (-p [P-VAL-THLD] in MACS2 parameter) 0.01 is used.

Program(s)	MACSv2 (2.1.0) https://github.com/taoliu/MACS/ Installation Instructions (https://github.com/taoliu/MACS/blob/master/INSTALL.rst). NOTE: Works only with Python 2.7

	(>=2.7.5). Does not work with Python 3. Also requires slopBed, bedClip and bedGraphToBigWig from KentTools (ucsc_tools)
Input(s)	RepN ChIP ${REP1_TA_FILE} ${REP2_TA_FILE} Pooled replicate ${POOLED_TA_FILE} RepN pseudoreplicate1 ${REP*_PR1_TA_FILE} RepN pseudoreplicate2 ${REP*_PR2_TA_FILE} Pooled-pseudoreplicate 1 ${PPR1_TA_FILE} Pooled-pseudoreplicate 2 ${PPR2_TA_FILE}
Output(s)	Narrowpeak file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.narrowPeak.gz ~~Gappedpeak file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.gappedPeak.gz~~ Fold-enrichment bigWig file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.fc.signal.bw -log10(pvalue) bigWig file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.pval.signal.bw
Commands	prefix_sig = “$prefix” peakfile = "$prefix.narrowPeak.gz" ~~bpeakfile = "$prefix.broadPeak.gz"~~ ~~gpeakfile = "$prefix.gappedPeak.gz"~~ pval_thresh = 0.01 NPEAKS=300000 # capping number of peaks called from MACS2 fc_bedgraph = "$prefix.fc.signal.bedgraph" fc_bedgraph_srt = "$prefix.fc.signal.srt.bedgraph" fc_bigwig = "$prefix_sig.fc.signal.bigwig" pval_bedgraph = "$prefix.pval.signal.bedgraph" pval_bedgraph_srt = "$prefix.pval.signal.srt.bedgraph" pval_bigwig = "$prefix_sig.pval.signal.bigwig" smooth_window=150 # default shiftsize=$((-$smooth_window/2)) ~~macs2 callpeak \~ ~~-t $tag -f BED -n "$prefix" -g "$gensz" -p $pval_thresh \~ ~~--nomodel --shift -$shiftsize --extsize $smooth_window --broad --keep-dup all~~ ~~//# Sort by Col8 in descending order and replace long peak names in Column 4 with Peak_<peakRank>~~ ~~sort -k 8gr,8gr "$prefix"_peaks.broadPeak | awk 'BEGIN{OFS="\t"}{$4="Peak_"NR ; print $0}' | gzip -nc > $bpeakfile~~ ~~sort -k 14gr,14gr "$prefix"_peaks.gappedPeak | awk 'BEGIN{OFS="\t"}{$4="Peak_"NR ; print $0}' | gzip -nc > $gpeakfile~~ ~~rm -f "$prefix"_peaks.broadPeak~~ ~~rm -f "$prefix"_peaks.gappedPeak~~ macs2 callpeak \ -t $tag -f BED -n "$prefix" -g "$gensz" -p $pval_thresh \ --shift $shiftsize --extsize $smooth_window --nomodel -B --SPMR --keep-dup all --call-summits # Sort by Col8 in descending order and replace long peak names in Column 4 with Peak_<peakRank> sort -k 8gr,8gr "$prefix"_peaks.narrowPeak | awk 'BEGIN{OFS="\t"}{$4="Peak_"NR ; print $0}' | head -n ${NPEAKS} | gzip -nc > $peakfile

	rm -f "$prefix"_peaks.narrowPeak rm -f "$prefix"_peaks.xls rm -f "$prefix"_summits.bed macs2 bdgcmp -t "$prefix"_treat_pileup.bdg -c "$prefix"_control_lambda.bdg --o-prefix "$prefix" -m FE slopBed -i "$prefix"_FE.bdg -g "$chrsz" -b 0 | bedClip stdin "$chrsz" $fc_bedgraph rm -f "$prefix"_FE.bdg sort -k1,1 -k2,2n $fc_bedgraph > $fc_bedgraph_srt bedGraphToBigWig $fc_bedgraph_srt "$chrsz" "$fc_bigwig" rm -f $fc_bedgraph $fc_bedgraph_srt # sval counts the number of tags per million in the (compressed) BED file sval=$(wc -l <(zcat -f "$tag") | awk '{printf "%f", $1/1000000}') macs2 bdgcmp -t "$prefix"_treat_pileup.bdg -c "$prefix"_control_lambda.bdg --o-prefix "$prefix" -m ppois -S "${sval}" slopBed -i "$prefix"_ppois.bdg -g "$chrsz" -b 0 \| bedClip stdin "$chrsz" $pval_bedgraph rm -f "$prefix"_ppois.bdg sort -k1,1 -k2,2n $pval_bedgraph > $pval_bedgraph_srt bedGraphToBigWig $pval_bedgraph_srt "$chrsz" "$pval_bigwig" rm -f $pval_bedgraph $pval_bedgraph_srt rm -f "$prefix"_treat_pileup.bdg "$prefix"_control_lambda.bdg
QC to report	
Status	Beta

3b. Blacklist filtering for peaks : function blacklist_filter_peak()

Program(s)	bedtools 2.26 (intersectBed) blacklist filtering is also integrated in IDR (4.) and naive overlap thresholding (3c.).
Input(s)	Original peak: ${PEAK}=${PREFIX}.narrowPeak.gz Blacklist: ${BLACKLIST} hg19: http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeMapability/wgEncodeDacMapabilityConsensusExcludable.bed.gz mm9: http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/mm9-mouse/mm9-blacklist.bed.gz GRCh38: http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/hg38-human/hg38.blacklist.bed.gz mm10: http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/mm10-mouse/mm10.blacklist.bed.gz
Output(s)	Blacklist filtered peaks ${FILTERED_PEAK}=${PREFIX}.filt.narrowPeak.gz

Commands	PEAK = "${PREFIX}.narrowPeak.gz" FILTERED_PEAK = "${PREFIX}.narrowPeak.filt.gz" bedtools intersect -v -a ${PEAK} -b ${BLACKLIST} \ | awk 'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0}' \ | grep -P 'chr[\dXY]+[\t]' | gzip -nc > ${FILTERED_PEAK}
QC to report	
Status	Beta

3c. bed to bigbed conversion for narrowpeaks : function _narrowpeak_to_bigbed()

Program(s)	requires bedClip and bedToBigbed from KentUtils (ucsc_tools)
Input(s)	Original peak: ${PEAK}=${PREFIX}.narrowPeak.gz ~~(gappedPeak and broadPeak are also supported)~~ 2-column chromosome sizes file: ${CHRSZ} narrowpeak definition file from KentUtils: narrowPeak.as (https://github.com/ENCODE-DCC/kentUtils/blob/master/src/hg/lib/encode/narrowPeak.as)
Output(s)	bigbed: ${BIGBED}=${PREFIX}.narrowPeak.bb
Commands	cat ${CHRSZ} | grep -P 'chr[\dXY]+[\t]' > ${BIGBED}.chrsz.tmp zcat ${PEAK} | sort -k1,1 -k2,2n > ${BIGBED}.tmp bedClip ${BIGBED}.tmp ${BIGBED}.chrsz.tmp ${BIGBED}.tmp2 bedToBigBed -type=bed6+4 -as=narrowPeak.as ${BIGBED}.tmp2 ${BIGBED}.chrsz.tmp ${BIGBED} rm -f ${BIGBED}.tmp ${BIGBED}.tmp2 ${BIGBED}.chrsz.tmp
QC to report	
Status	Beta

3d <u>Naive overlap thresholding for MACS2 peak calls :</u> function naive_overlap_peak()
These are peaks in the pooled data (reads pooled across reps) that overlap peaks in BOTH true replicates OR overlap peaks in BOTH pooled pseudoreplicates. Repeat this procedure for 4 sets of peaks described in Section 4.

Program(s)	bedtools 2.26 (intersectBed)
Input(s)	Peaks (narrowPeak ~~or gappedPeak~~) for Pooled, Rep1, Rep2.
Output(s)	For narrowPeak finalPeakList.narrowPeak.gz ~~For gappedPeak finalPeakList.gappedPeak.gz~~
Commands	# ====================== # For narrowPeak files # ======================

```
# Find pooled peaks that overlap Rep1 and Rep2 where overlap is defined as the fractional overlap wrt any one of the overlapping peak pairs  >= 0.5

intersectBed -wo -a Pooled.narrowPeak.gz -b Rep1.narrowPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$13-$12; if (($21/s1 >= 0.5) || ($21/s2 >= 0.5))
{print $0}}' | cut -f 1-10 | sort | uniq |
intersectBed -wo -a stdin -b Rep2.narrowPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$13-$12; if (($21/s1 >= 0.5) || ($21/s2 >= 0.5))
{print $0}}' | cut -f 1-10 | sort | uniq > PooledInRep1AndRep2.narrowPeak.gz

# ============================
# Filter using black list
# ============================
bedtools intersect -v -a PooledInRep1AndRep2.narrowPeak.gz -b ${BLACKLIST} | awk
'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0}' | grep -P 'chr[\dXY]+[ \t]'  | gzip -nc >
PooledInRep1AndRep2.filt.narrowPeak.gz
```

~~# ======================~~
~~For BroadPeak files (there is just a difference is the awk commands wrt the column numbers)~~
~~# ======================~~

```
# Find pooled peaks that overlap Rep1 and Rep2 where overlap is defined as the fractional overlap wrt any one of the overlapping peak pairs  >= 0.5

intersectBed -wo -a Pooled.broadPeak.gz -b Rep1.broadPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$12-$11; if (($19/s1 >= 0.5) || ($19/s2 >= 0.5))
{print $0}}' | cut -f 1-9 | sort | uniq |
intersectBed -wo -a stdin -b Rep2.broadPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$12-$11; if (($19/s1 >= 0.5) || ($19/s2 >= 0.5))
{print $0}}' | cut -f 1-9 | sort | uniq > PooledInRep1AndRep2.broadPeak.gz

# Find pooled peaks that overlap PooledPseudoRep1 and PooledPseudoRep2 where overlap is defined as the fractional overlap wrt any one of the overlapping peak pairs  >= 0.5

intersectBed -wo -a Pooled.broadPeak.gz -b PsRep1.broadPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$12-$11; if (($19/s1 >= 0.5) || ($19/s2 >= 0.5))
{print $0}}' | cut -f 1-9 | sort | uniq |
intersectBed -wo -a stdin -b PsRep2.broadPeak.gz |
awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$12-$11; if (($19/s1 >= 0.5) || ($19/s2 >= 0.5))
{print $0}}' | cut -f 1-9 | sort | uniq > PooledInPsRep1AndPsRep2.broadPeak.gz

# Combine peak lists

zcat PooledInRep1AndRep2.broadPeak.gz PooledInPsRep1AndPsRep2.broadPeak.gz | sort |
uniq | awk 'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0}' \
```

	~~| grep -P 'chr[0-9XY]+(?!_)' > finalPeakList.broadPeak.gz~~ ~~# ======================~~ ~~For gappedPeak files (there is just a difference is the awk commands wrt the column numbers)~~ ~~# ======================~~ ~~# Find pooled peaks that overlap Rep1 and Rep2 where overlap is defined as the fractional overlap wrt any one of the overlapping peak pairs -> = 0.5~~ ~~intersectBed -wo -a Pooled.gappedPeak.gz -b Rep1.gappedPeak.gz |~~ ~~awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$18-$17; if (($31/s1 >= 0.5) || ($31/s2 >= 0.5)) {print $0}}' | cut -f 1-15 | sort | uniq |~~ ~~intersectBed -wo -a stdin -b Rep2.gappedPeak.gz |~~ ~~awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$18-$17; if (($31/s1 >= 0.5) || ($31/s2 >= 0.5)) {print $0}}' | cut -f 1-15 | sort | uniq > PooledInRep1AndRep2.gappedPeak.gz~~ ~~# Find pooled peaks that overlap PooledPseudoRep1 and PooledPseudoRep2 where overlap is defined as the fractional overlap wrt any one of the overlapping peak pairs -> = 0.5~~ ~~intersectBed -wo -a Pooled.gappedPeak.gz -b PsRep1.gappedPeak.gz |~~ ~~awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$18-$17; if (($31/s1 >= 0.5) || ($31/s2 >= 0.5)) {print $0}}' | cut -f 1-15 | sort | uniq |~~ ~~intersectBed -wo -a stdin -b PsRep2.gappedPeak.gz |~~ ~~awk 'BEGIN{FS="\t";OFS="\t"}{s1=$3-$2; s2=$18-$17; if (($31/s1 >= 0.5) || ($31/s2 >= 0.5)) {print $0}}' | cut -f 1-15 | sort | uniq > PooledInPsRep1AndPsRep2.gappedPeak.gz~~ ~~# Combine peak lists~~ ~~zcat PooledInRep1AndRep2.gappedPeak.gz PooledInPsRep1AndPsRep2.gappedPeak.gz |~~ ~~sort | uniq | awk 'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0}' \~~ ~~| grep -P 'chr[0-9XY]+(?!_)' > finalPeakList.gappedPeak.gz~~
QC to report	
Status	Beta

4. Run IDR on all pairs of replicates, self-pseudoreplicates and pooled pseudoreplicates

IDR is optional. The IDR peaks are a subset of the naive overlap peaks that pass a specific IDR threshold of 5%.

Use IDR to compare all pairs of matched replicates : function _idr()
(1) True replicates narrowPeak files: ${REP1_PEAK_FILE} vs. ${REP2_PEAK_FILE} IDR results transferred to Pooled-replicates narrowPeak file ${POOLED_PEAK_FILE}
(2) Pooled-pseudoreplicates: ${PPR1_PEAK_FILE} vs. ${PPR2_PEAK_FILE} IDR results transferred to

Pooled-replicates narrowPeak file ${POOLED_PEAK_FILE}
(3) Rep1 self-pseudoreplicates: ${REP1_PR1_PEAK_FILE} vs. ${REP1_PR2_PEAK_FILE} IDR results transferred to Rep1 narrowPeak file ${REP1_PEAK_FILE}
(4) Rep2 self-pseudoreplicates: ${REP2_PR1_PEAK_FILE} vs. ${REP2_PR2_PEAK_FILE} IDR results transferred to Rep2 narrowPeak file ${REP2_PEAK_FILE}

IDR Threshold: Use IDR threshold of 5% for all pairwise analyses

4a. For True Replicates
Below we show the use for true replicates. The same steps can be applied for all other pairs.

- We cap number of peaks called from MACS2 by 300K

Program(s)	IDR 2.0.4 (https://github.com/kundajelab/idr) / Installation instructions (https://github.com/kundajelab/idr#installation). NOTE: Works only with Python3
Input(s)	• a pair of narrowPeak files for replicates ${REP1_PEAK_FILE} ${REP2_PEAK_FILE} • a pooled-replicate narrowPeak file ${POOLED_PEAK_FILE} • a blacklist ${BLACKLIST} ○ hg19: http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeMapability/wgEncodeDacMapabilityConsensusExcludable.bed.gz ○ mm9: http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/mm9-mouse/mm9-blacklist.bed.gz ○ GRCh38: http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/hg38-human/hg38.blacklist.bed.gz ○ mm10: http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/mm10-mouse/mm10.blacklist.bed.gz • For SPP use signal.value for ranking with a parameter '--rank signal.value' • For GEM use signal.value for ranking with a parameter '--rank signal.value' • For PeakSeq use q.value for ranking with a parameter '--rank q.value'
Output(s)	• The output from EM fitting: suffixed by overlapped-peaks.txt.png • The full set of peaks that overlap between the replicates with local and global IDR: suffixed by overlapped-peaks.txt ${IDR_OUTPUT} • IDR output file ${IDR_OUTPUT} ○ # Columns 1-10 are same as pooled common peaks narrowPeak columns ○ # Col 11: -log10(local IDR value) ○ # Col 12: -log10(global IDR value) ○ # Col 15: ranking measure from Rep1 ○ # Col 19: ranking measure from Rep2 • Final IDR thresholded file ${REP1_VS_REP2}.IDR0.05.narrowPeak.gz • Final IDR thresholded file filtered using a blacklist ${REP1_VS_REP2}.IDR0.05.filt.narrowPeak.gz
Commands	IDR_THRESH=0.05

	# ========== # Perform IDR analysis. # Generate a plot and IDR output with additional columns including IDR scores. # ========== idr --samples ${REP1_PEAK_FILE} ${REP2_PEAK_FILE} --peak-list ${POOLED_PEAK_FILE} --input-file-type narrowPeak --output-file ${IDR_OUTPUT} --rank p.value --soft-idr-threshold ${IDR_THRESH} --plot --use-best-multisummit-IDR # ========== # Get peaks passing IDR threshold of 5% # ========== IDR_THRESH_TRANSFORMED=$(awk -v p=${IDR_THRESH} 'BEGIN{print -log(p)/log(10)}') awk 'BEGIN{OFS="\t"} $12>="'${IDR_THRESH_TRANSFORMED}'" {print $1,$2,$3,$4,$5,$6,$7,$8,$9,$10}' ${IDR_OUTPUT} | sort | uniq | sort -k7n,7n | gzip -nc > ${REP1_VS_REP2}.IDR0.05.narrowPeak.gz NPEAKS_IDR=$(zcat ${REP1_VS_REP2}.IDR0.05.narrowPeak.gz | wc -l) # ========== # Filter using black list # ========== bedtools intersect -v -a ${REP1_VS_REP2}.IDR0.05.narrowPeak.gz -b ${BLACKLIST} | awk 'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0}' | grep -P 'chr[\dXY]+[\t]' | gzip -nc > ${REP1_VS_REP2}.IDR0.05.filt.narrowPeak.gz
Parameters	• --samples : [REP1_PEAK_FILE] and [REP2_PEAK_FILE] are the peak calls for the pair of replicates in narrowPeak format. They must be compressed files. e.g. /peaks/reps/chipSampleRep1_VS_controlSampleRep0.narrowPeak.gz AND /peaks/reps/chipSampleRep2_VS_controlSampleRep0.narrowPeak.gz • --input-file-type : the peak file format (narrowPeak or broadPeak). Set to narrowPeak if it is narrowPeak/regionPeak or broadPeak if it is broadPeak. BroadPeak files do not contain Column 10. • --rank : the ranking measure to use. It can take only one of the following values signal.value , p.value or q.value • --soft-idr-threshold : IDR threshold, Set to ${IDR_THRESH}
QC to report	• Number of peaks passing IDR thresholds of 5% ${NPEAKS_IDR} • For each pairwise analysis, we have a *overlapped-peaks.txt file. The 12th column of the overlapped-peaks.txt file has the global IDR score for each pair of overlapping peaks. • Also store ${POOLED_COMMON_PEAKS_IDR} • To get the number of peaks that pass an IDR threshold of T (e.g. 0.01) you simply find the number of lines in ${POOLED_COMMON_PEAKS_IDR} that have Column 14 <= T
Status	Frozen

- If you have more than 2 true replicates select the longest peak list from all pairs that passes the IDR threshold.
- **Nt = Best no. of peaks passing IDR threshold by comparing true replicates**

4b. IDR analysis - self-pseudoreplicates

- Perform as with real replicates, but comparing pseudoreplicate 1 vs pseudoreplicate 2 made from each of the real biological replicate peaks
- **Rep1 self-pseudoreplicates:** ${REP1_PR1_PEAK_FILE} vs. ${REP1_PR2_PEAK_FILE} and use ${REP1_PEAK_FILE} as pooled file
- **Rep2 self-pseudoreplicates:** ${REP2_PR1_PEAK_FILE} vs. ${REP2_PR2_PEAK_FILE} and use ${REP2_PEAK_FILE} as pooled file
- This gives the self-consistent IDR peaks
- **N1 and N2 = No. of peaks passing IDR threshold by comparing self-pseudoReplicates for Rep1 and Rep2 respectively**

4c. IDR analysis - pooled pseudoreplicates

- Perform as with real replicates, but comparing pooled-pseudoreplicate 1 vs pooled-pseudoreplicate 2 made from the pooled biological replicate peaks
- ${PPR1_PEAK_FILE} vs. ${PPR2_PEAK_FILE} and use ${POOLED_PEAK_FILE} as pooled file
- **Np = No. of peaks passing IDR threshold by comparing pooled pseudo-replicates**

4d. Select final peak calls - conservative set

- If you have more than 2 true replicates select the longest peak list from all pairs that passes the 5% IDR threshold. This is the conservative peak set.
- **Nt = Best no. of peaks passing IDR threshold by comparing true replicates**
- Filter using black list: http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeMapability/wgEncodeDacMapabilityConsensusExcludable.bed.gz

4e. Select final peak calls - optimal set

- **Longest of the Nt and Np peak lists**
- Filter using black list: http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeMapability/wgEncodeDacMapabilityConsensusExcludable.bed.gz

4f. Compute IDR QC scores

- **Rescue Ratio = max(Np,Nt) / min(Np,Nt)**
  Nt and Np should be within a factor of 2 of each other
- **Self-consistency Ratio = max(N1,N2) / min(N1,N2)**
  N1 and N2 should be within a factor of 2 of each other
- If Rescue Ratio AND self-consistency Ratio are both > 2, Flag the file for reproducibility FAIL (-1)
- If Rescue Ratio OR self-consistency Ratio are > 2, Flag the file for reproducibility Borderline (0)

Rescue Ratio v2 <TODO>
Self-Consistency Ratio v2 <TODO>

4g. Compute Fraction of Reads in Peaks (FRiP) : function frip(ta_file, idr_peak_file)
You compute the fraction of reads from each replicate tagAlign and pooled tagAlign that fall within the Np and Nt peak sets.

Program(s)	bedtools 2.26 (https://github.com/kundajelab/idr) / Installation instructions
Input(s)	• Final tagAlign file for Rep1 ${REP1_TA_FILE} vs. IDR Peak file for self pseudo replicates of Rep1 ${REP1_PR_IDR_PEAK_FILE}. • Final tagAlign file for Rep2 ${REP2_TA_FILE} vs. IDR Peak file for self pseudo

	replicates of Rep2 ${REP2_PR_IDR_PEAK_FILE}. • Pooled tagAlign file zcat ${REP1_TA_FILE} ${REP2_TA_FILE} | gzip -nc > ${POOLED_TA_FILE} vs. Conservative IDR Peak file (Nt) ${CONS_IDR_PEAK_FILE}. • Pooled tagAlign file ${POOLED_TA_FILE} vs.Optimal IDR Peak file (Np) ${OPTIMAL_IDR_PEAK_FILE}.
Output(s)	• Text file(${FRIP} with FRiP value
Commands	val1=$(bedtools intersect -a <(zcat -f ${TA_FILE}) -b <(zcat -f ${IDR_PEAK_FILE}) -wa -u \| wc -l) val2=$(zcat ${TA_FILE} \| wc -l) awk 'BEGIN {print '${val1}'/'${val2}'}' > $FRIP
Parameters	•
QC to report	• Fraction of reads in peak
Status	Frozen

5. Count signal track generation

function: _count_signal_track()

Program(s)	• bedtools 2.26.0 • bedGraphToBigWig
Input(s)	RepX ATAC ${REPX_TA_FILE}, or Pooled replicate ${POOLED_TA_FILE} Chromosome sizes file ${CHRSZ}
Output(s)	Positive bigWig file prefix.positive.bigwig Negative bigWig file prefix.negative.bigwig
Commands	zcat -f ${TA_FILE} | sort -k1,1 -k2,2n | bedtools genomecov -5 -bg -strand + -g ${CHRSZ} -i stdin > TMP.POS.BED bedGraphToBigWig TMP.POS.BED ${CHRSZ} ${TA_FILE_PREFIX}.positive.bigwig zcat -f ${TA_FILE} | sort -k1,1 -k2,2n | bedtools genomecov -5 -bg -strand - -g ${CHRSZ} -i stdin > TMP.NEG.BED bedGraphToBigWig TMP.NEG.BED ${CHRSZ} ${TA_FILE_PREFIX}.negative.bigwig
QC to report	
Status	Beta

7. Annotation based analyses

7.1 TSS Enrichment

function: _tss_enrich()

Program(s)	• python script
Input(s)	- Filtered/deduped BAM file ${PREFIX}.nodup.bam - Chromosome sizes file ${CHRSZ} - Read length estimated from FASTQ - TSS BED file: tss.bed.gz TSS Bed file Example for GRCh38 chr1 69089 69090 ENSG00000186092.4 0 + chr1 451678 451679 ENSG00000278566.1 0 - chr1 686654 686655 ENSG00000273547.1 0 - chr1 924878 924879 ENSG00000187634.10 0 + ...
Output(s)	TSS plot file: ${PREFIX}.tss_enrich.png TSS log file: ${PREFIX}.tss_enrich.log.
Python script	def make_tss_plot(bam_file, tss, prefix, chromsizes, read_len, bins=400, bp_edge=2000, processes=8, greenleaf_norm=True): ''' Take bootstraps, generate tss plots, and get a mean and standard deviation on the plot. Produces 2 plots. One is the aggregation plot alone, while the other also shows the signal at each TSS ordered by strength. ''' logging.info('Generating tss plot...') tss_plot_file = '{0}.tss_enrich.png'.format(prefix) tss_plot_large_file = '{0}.large_tss_enrich.png'.format(prefix) tss_log_file = '{0}.tss_enrich.qc'.format(prefix) # Load the TSS file tss = pybedtools.BedTool(tss) tss_ext = tss.slop(b=bp_edge, g=chromsizes) # Load the bam file bam = metaseq.genomic_signal(bam_file, 'bam') # Need to shift reads and just get ends, just load bed file? bam_array = bam.array(tss_ext, bins=bins, shift_width = -read_len/2, # Shift to center the read on the cut site processes=processes, stranded=True) # Actually first build an "ends" file #get_ends = '"zcat {0} | awk -F '\t' 'BEGIN {{OFS="\t"}} {{if ($6 == "-") {{$2=$3-1; print}} else {{$3=$2+1; print}} }}' | gzip -c > {1}_ends.bed.gz"'.format(bed_file, prefix) #print(get_ends) #os.system(get_ends) #bed_reads = metaseq.genomic_signal('{0}_ends.bed.gz'.format(prefix), 'bed') #bam_array = bed_reads.array(tss_ext, bins=bins, # processes=processes, stranded=True)

```
# Normalization (Greenleaf style): Find the avg height
# at the end bins and take fold change over that
if greenleaf_norm:
    # Use enough bins to cover 100 bp on either end
    num_edge_bins = int(100/(2*bp_edge/bins))
    bin_means = bam_array.mean(axis=0)
    avg_noise = (sum(bin_means[:num_edge_bins]) +
                 sum(bin_means[-num_edge_bins:]))/(2*num_edge_bins)
    bam_array /= avg_noise
else:
    bam_array /= bam.mapped_read_count() / 1e6

# Generate a line plot
fig = plt.figure()
ax = fig.add_subplot(111)
x = np.linspace(-bp_edge, bp_edge, bins)

ax.plot(x, bam_array.mean(axis=0), color='r', label='Mean')
ax.axvline(0, linestyle=':', color='k')

# Note the middle high point (TSS)
tss_point_val = max(bam_array.mean(axis=0))

# write tss_point_val to file
with open(tss_log_file, 'w') as fp:
    fp.write(str(tss_point_val))

ax.set_xlabel('Distance from TSS (bp)')
if greenleaf_norm:
    ax.set_ylabel('TSS Enrichment')
else:
    ax.set_ylabel('Average read coverage (per million mapped reads)')
ax.legend(loc='best')

fig.savefig(tss_plot_file)

# Print a more complicated plot with lots of info

# Find a safe upper percentile - we can't use X if the Xth percentile is 0
upper_prct = 99
if mlab.prctile(bam_array.ravel(), upper_prct) == 0.0:
    upper_prct = 100.0

plt.rcParams['font.size'] = 8
fig = metaseq.plotutils.imshow(bam_array,
                               x=x,
                               figsize=(5, 10),
                               vmin=5, vmax=upper_prct, percentile=True,
                               line_kwargs=dict(color='k', label='All'),
                               fill_kwargs=dict(color='k', alpha=0.3),
                               sort_by=bam_array.mean(axis=1))

# And save the file
fig.savefig(tss_plot_large_file)

return tss_plot_file, tss_plot_large_file, tss_log_file
```

QC to report	

Status	Beta

7.2. Fraction of Reads in annotated regions
function: _annot_enrich()

Program(s)	• bedtools 2.26
Input(s)	TAG-ALIGN file: ${TA_FILE} Annotation BED file - DNase BED file: BED file of universal DHS regions - Promoter BED file: BED file of promoter regions - Enhancer BED file: BED file of enhancer regions - Blacklist BED file
Output(s)	Fraction of reads in each annotated region (for each BED file)
Python script	# For each annotation BED file, run the following command line bedtools sort -i ${EACH_ANNOT_BED} | bedtools merge -i stdin | bedtools intersect -u -nonamecheck -a ${TA_FILE} -b stdin | wc -l
QC to report	
Status	Beta

pseudo code

```
function atac_SE( replicate ) {

   if ( trimmed_fastq ) {
         p1 = fastq
   }
   else {
         adapter = _detect_adapter( fastq )
         p1 = _trim_adapters( fastq, adapter ) // using cutadapt
   }

   ( bam, samstat_qc, non_mito_samstat_qc) = _bowtie2_SE( p1, bowtie2_index )
   ( mito_bam, mito_only_samstat_qc, _ ) = _bowtie2_SE( p1, bowtie2_mito_only_index
)
```

```
    frac_mito_qc = _frac_mito( non_mito_samstat_qc, mito_only_samstat_qc )

    ( filt_bam, dup_qc, pbc_qc ) = _dedup_bam( bam )

    tag = _bam_to_tag( filt_bam )

    // use subsampled tagalign for steps downstream
    // (different from subsampling for cross-corr.)

    if ( subsample != 0 ) {
        subsampled_tag = _subsample_tag( tag, subsample )
    }
    else {
        subsampled_tag = tag
    }

    final_tag = _tn5_shift_tag( subsampled_tag )

    // make SPR (self pseudo replicates)
    ( final_tag_pr1, final_tag_pr2 ) = _spr( final_tag )

    // call peaks on pseudo replicates (p_val_thresh = 0.01)
    ( peak_pr1, ~~gpeak_pr1~~ ) = _macs2( final_tag_pr1 )
    ( peak_pr2, ~~gpeak_pr2~~ ) = _macs2( final_tag_pr2 )

    // call peaks on true replicate (with p_val_thresh = 0.01)
    ( peak, ~~gpeak~~, pval_bigwig ) = _macs2( final_tag )

    // subsample tagalign for cross-corr. analysis
    subsampled_tag_xcor = _subsample_tag( tag )

    ( xcor_qc, xcor_plot ) = _xcor( subsampled_tag_xcor )
}


function atac_PE( replicate ) {

    if ( trimmed_fastq ) {
        p1 = fastq1
```

```
        p2 = fastq2
    }
    else {
        adapter1 = _detect_adapter( fastq1 )
        adapter2 = _detect_adapter( fastq2 )
        p1 = _trim_adapters( fastq1, adapter1 ) // using cutadapt
        p2 = _trim_adapters( fastq2, adapter2 )
    }

    ( bam, samstat_qc, non_mito_samstat_qc) = _bowtie2_PE( p1, p2, bowtie2_index )
    ( mito_bam, mito_only_samstat_qc, _ ) = _bowtie2_PE( p1, p2,
bowtie2_mito_only_index )
    frac_mito_qc = _frac_mito( non_mito_samstat_qc, mito_only_samstat_qc)

    (filt_bam, dup_qc, pbc_qc ) = _dedup_bam_PE( bam )

    bedpe = _bam_to_bedpe( filt_bam )

    tag = _bedpe_to_tag( subsampled_bedpe )

    final_tag = _tn5_shift_tag( tag )

    // use subsampled tagalign for steps downstream
    // (different from subsampling for cross-corr.)
    if ( subsample != 0 ) {
        subsampled_tag = _subsample_tag_PE( final_tag , subsample )
    }
    else {
        subsampled_tag = final_tag
    }

    // make SPR (self pseudo replicates)
    (tag_pr1, tag_pr2) = _spr_PE( subsampled_tag )

    // call peaks on pseudo replicate (with p_val_thresh = 0.01)
    ( peak_pr1, gpeak_pr1 ) = _macs2( final_tag_pr1 )
    ( peak_pr2, gpeak_pr2 ) = _macs2( final_tag_pr2 )
```

```
    // call peaks on true replicate (with p_val_thresh = 0.01)
    ( peak, gpeak, pval_bigwig ) = _macs2( final_tag )

    // subsample tagalign for cross-corr. analysis (take one read end per pair)
    subsampled_tag_xcor = _subsample_bedpe_to_tag_xcor( bedpe )

    ( xcor_qc, xcor_plot ) = _xcor( subsampled_tag_xcor )
}


function main() {

    // By default, smoothing window for MACS2 is 150. IDR threshold is 0.05
    smooth_win = 150
    idr_thresh = 0.05

    // align, call peaks, do cross-corr. analysis and ataqc on each replicate

    for ( rep = 1; rep <= num_rep; rep++ ) {

            if ( se ) { // for single-ended replicate
                    atac_SE( rep )
            }
            else { // for paired end replicate
                    atac_PE( rep )
            }

           // GC bias
           // ref_fa: reference genome FASTA file
           _gc_bias() ( nodup_bam, ref_fa )
           // fragment length statistics
           _frag_len_stat( nodup_bam )
          // count signal track generation          _count_signal_track ( tag )    }


    // make pooled replicates and PPR (pooled pseudo replicates)
    ( tag_pooled, tag_ppr1, tag_ppr2 ) = _ppr( tag_rep1, tag_pr1_rep1, tag_pr2_rep1, \
                                               tag_rep2, tag_pr1_rep2, tag_pr2_rep2 )
```

```
    _count_signal_track ( tag_pooled )

    // Jensen-Shannon distance
    _jsd() ( nodup_bam_rep1, nodup_bam_rep2, mapq_thresh, blacklist_bed )

   // fraction of reads in annotated regions
   // BED files for TSS, DNase (universal DHS), Enhancer, Promoter, Blacklist
    _annot_enrich( tag_rep1, TSS_bed, dnase_bed, enh_bed, prom_bed, blacklist_bed )
    _annot_enrich( tag_rep2, TSS_bed, dnase_bed, enh_bed, prom_bed, blacklist_bed )


    // call peaks on pooled pseudo replicates

    ( peak_ppr1, ~~gpeak_ppr1~~ ) = _macs2( tag_ppr1 )
    ( peak_ppr2, ~~gpeak_ppr2~~ ) = _macs2( tag_ppr2 )
    ( peak_pooled, ~~gpeak_pooled~~ ) = _macs2( tag_pooled )


    // naive overlap
    overlap_tr = naive_overlap( peak_rep1, peak_rep2, peak_pooled )
    overlap_pr_rep1 = naive_overlap( peak_pr1_rep1, peak_pr2_rep1, peak_rep1 )
    overlap_pr_rep2 = naive_overlap( peak_pr1_rep2, peak_pr2_rep2, peak_rep2 )
    overlap_ppr = naive_overlap( peak_ppr1, peak_ppr2, peak_pooled )
    ( overlap_qc, overlap_opt, overlap_consv ) = overlap_final_qc( overlap_tr,
overlap_pr_rep1, overlap_pr_rep2, overlap_ppr )

    // FRiP (fraction of reads in peaks)
    frip_rep1 = _frip( tag_rep1, overlap_pr_rep1 )
    frip_rep2 = _frip( tag_rep2, overlap_pr_rep2 )
    frip_Nt = _frip( tag_pooled, overlap_tr )
    frip_Np = _frip( tag_pooled, overlap_ppr )

~~    // IDR (blacklist filtering is included in the function _idr() )~~
~~    idr_tr = _idr( peak_rep1, peak_rep2, peak_pooled )~~
~~    idr_pr_rep1 = _idr( peak_pr1_rep1, peak_pr2_rep1, peak_rep1 )~~
~~    idr_pr_rep2 = _idr( peak_pr1_rep2, peak_pr2_rep2, peak_rep2 )~~
~~    idr_ppr = _idr( peak_ppr1, peak_ppr2, peak_pooled )~~
~~    ( idr_qc, idr_opt, idr_consv ) = _idr_final_qc( idr_tr, idr_pr_rep1, idr_pr_rep2, idr_ppr )~~
```

```
    // FRiP (fraction of reads in peaks)
    frip_rep1 = _frip( tag_rep1, idr_pr_rep1 )
    frip_rep2 = _frip( tag_rep2, idr_pr_rep2 )
    frip_Nt = _frip( tag_pooled, idr_tr )
    frip_Np = _frip( tag_pooled, idr_ppr )

}
```