# ATAC-Seq pipeline v1 specifications

## 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)<br>• modules/align_trim_adapters.bds |
| Input(s) | • Input: $fastq |
| Output(s) | • log with detected adaptors: $log |
| Commands | log = "$prefix.adapter.txt"<br><br>python3 $script_dir/GGR_code/scripts/detect_adapter.py $fastq > $log<br><br># 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<br>• **modules/align_trim_adapters.bds** |
| **Input(s)** | • Input: $fastq<br>• Adapter sequence: $adapter<br>• Adapter error rate: $adapter_err_rate (0.2 by default) |
| **Output(s)** | • Trimmed fastq $trimmed_fastq |
| **Commands** | trimmed_fastq = "$prefix.trim.fastq.gz"<br><br>cutadapt -m 5 -e $adapter_err_rate -a $adapter $fastq \| gzip -c > $trimmed_fastq |
| **QC to report** | Output from last command i.e. samtools flagstat |
| **Status** | Frozen |

# 1a. Read alignment (Bowtie2 aligner)

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

■

| | |
|---|---|
| **Program(s)** | • Bowtie2 version 2.2.6<br>• SAMtools 1.2, sambamba 0.6.5<br>• **modules/align_bowtie2.bds** |
| **Input(s)** | • Input: $fastq<br>• Bowtie2 index: $bwt2_idx<br>• # alignments reported for multimapping (4 for ENCODE3, 0 by default): $multimapping |
| **Output(s)** | • BAM file $bam<br>• mapping stats from flagstat $log |
| **Commands** | bam = "$prefix.bam"<br>log = "$prefix.align.log"<br><br>bowtie2 -k $multimapping --local -x $bwt2_idx --threads $nth_bwt2 -U <(zcat -f $fastq) 2> $log \| \<br>samtools view -Su /dev/stdin \| samtools sort - $prefix |
| **QC to report** | Output from last command i.e. samtools flagstat |
| **Status** | Frozen |

~~**Single-End ATAC-seq parameters with CSEM : function _bowtie2_csem()**~~

~~■~~

| | |
|---|---|
| ~~**Program(s)**~~ | ~~• Bowtie2 version 2.2.6~~<br>~~• SAMtools 1.2~~<br>~~• CSEM 2.4~~<br>~~• **modules/align_bowtie2.bds**~~ |
| ~~**Input(s)**~~ | ~~• Input: $fastq~~<br>~~• Bowtie2 index: $bwt2_idx~~<br>~~• # alignments reported for multimapping (4 by default): $multimapping~~ |
| ~~**Output(s)**~~ | ~~• BAM file $bam~~<br>~~• mapping stats from flagstat $log~~ |
| ~~**Commands**~~ | ~~bam = "$prefix.bam"~~<br>~~log = "$prefix.align.log"~~<br><br>~~bowtie2 -k $multimapping -x $bwt2_idx --threads $nth_bwt2 -U <(zcat -f $fastq) 2> $log > $sam~~<br><br>~~run-csem --sam -p $nth_bwt2 $sam 100 $srt_bam_prefix~~<br><br>~~rm -f $sam~~ |
| ~~**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<br>• SAMtools 1.2<br>• **modules/align_bowtie2.bds** |
| **Input(s)** | • Input: $fastq1, $fastq2<br>• Bowtie2 index: $bwt2_idx<br>• # alignments reported for multimapping (4 by default): $multimapping |
| **Output(s)** | • BAM file $bam<br>• mapping stats from flagstat $log |
| **Commands** | bam = "$prefix.bam"<br>log = "$prefix.align.log"<br><br>bowtie2 -k $multimapping -X2000 --local --mm --threads $nth_bwt2 -x $bwt2_idx \<br>-1 $fastq1 -2 $fastq2 \<br>2>$log \| \<br>samtools view -Su /dev/stdin \| samtools sort - $prefix |
| **QC to report** | Output from last command i.e. samtools flagstat |
| **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 1796 or -F 1804 to keep it the same as PE)
- 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 (1.2)<br>• MarkDuplicates (Picard - latest version 1.126)<br>• bedtools 2.22 |
| **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}<br>• Flagstat Metric for filtered BAM file ${FINAL_BAM_FILE_MAPSTATS}<br>• Duplication metrics from MarkDuplicates ${DUP_FILE_QC}<br>• Library complexity measures ${PBC_FILE_QC} |
| **Commands** | # ============================<br># Remove unmapped, mate unmapped<br># not primary alignment, reads failing platform<br># ==================<br>QNAME_SORT_BAM_FILE="${OFPREFIX}.qnmsrt.bam"<br>FILT_BAM_PREFIX="${OFPREFIX}.filt.srt"<br>FILT_BAM_FILE="${FILT_BAM_PREFIX}.bam"<br><br>samtools sort -n ${RAW_BAM_FILE} ${QNAME_SORT_BAM_PREFIX}<br>samtools view -h ${QNAME_SORT_BAM} \| $(which assign_multimappers.py) -k $multimapping \| samtools view -bS - > ${QNAME_SORT_BAM_FILT}<br><br>samtools view -F 1804 -b ${QNAME_SORT_BAM_FILT} > ${FILT_BAM_FILE}<br><br>rm -f ${QNAME_SORT_BAM} ${QNAME_SORT_BAM_FILT}<br><br># ========================<br># Mark duplicates<br># ========================<br>module add picard-tools/1.126<br><br>TMP_FILT_BAM_FILE="${FILT_BAM_PREFIX}.dupmark.bam"<br>MARKDUP="/srv/gs1/software/picard-tools/1.126/MarkDuplicates.jar"<br>DUP_FILE_QC="${FILT_BAM_PREFIX}.dup.qc" # QC file<br><br>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<br><br>mv ${TMP_FILT_BAM_FILE} ${FILT_BAM_FILE} |

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

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

# Index Final BAM file
samtools index ${FINAL_BAM_FILE} ${FINAL_BAM_INDEX_FILE}

samtools flagstat ${FINAL_BAM_FILE} > ${FINAL_BAM_FILE_MAPSTATS}

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

module add bedtools/2.22

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__":<br>'''<br>Runs the filtering step of choosing multimapped reads<br>'''<br><br>[alignment_cutoff, paired_ended] = parse_args()<br><br>if paired_ended:<br>alignment_cutoff = int(alignment_cutoff) * 2<br><br># Store each line in sam file as a list of reads,<br># where each read is a list of elements to easily<br># modify or grab things<br>current_reads = []<br>current_qname = ''<br><br>for line in sys.stdin:<br><br>read_elems = line.strip().split('\t')<br><br>if read_elems[0].startswith('@'):<br>sys.stdout.write(line)<br>continue<br><br># Keep taking lines that have the same qname<br>if read_elems[0] == current_qname:<br># Add line to current reads<br>current_reads.append(line)<br>pass<br>else:<br># Discard if there are more than the alignment cutoff<br>if len(current_reads) >= alignment_cutoff:<br>current_reads = [line]<br>current_qname = read_elems[0]<br>elif len(current_reads) > 0:<br># Just output all reads, which are then filtered with<br># samtools<br>for read in current_reads:<br>sys.stdout.write(str(read))<br><br># And then discard<br>current_reads = [line]<br>current_qname = read_elems[0]<br>else:<br># First read in file<br>current_reads.append(line)<br>current_qname = read_elems[0] |
| **QC to report** | (1) Flagstat output from Final filtered deduped BAM file ${FINAL_BAM_FILE_MAPSTATS}<br>(2) PICARD MarkDup output ${DUP_FILE_QC}<br>http://sourceforge.net/apps/mediawiki/picard/index.php?title=Main_Page#Q:_What_is_meaning_of_the_histogram_produced_by_MarkDuplicates.3F<br>(3) Library complexity measures ${PBC_FILE_QC}<br>• Format of file<br>TotalReads [tab] DistinctReads [tab] OneRead [tab] TwoRead [tab] NRF=Distinct/Total [tab] PBC1=OnePair/Distinct [tab] PBC2=OnePair/TwoPair<br>• NRF (non redundant fraction)<br>• PBC1 (PCR Bottleneck coefficient 1)<br>• PBC2 (PCR Bottleneck coefficient 2) |

| | |
|---|---|
| | ● PBC1 is the primary measure. Provisionally,<br>○ 0-0.5 is severe bottlenecking<br>○ 0.5-0.8 is moderate bottlenecking<br>○ 0.8-0.9 is mild bottlenecking<br>○ 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 1804).
- Retain properly paired reads -f 2
- 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 (1.2)<br>• MarkDuplicates (Picard - latest version 1.126)<br>• bedtools 2.22 |
| **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}<br>• Filtered deduped name sorted BAM file ${FINAL_NMSRT_BAM_FILE}<br>• Flagstat Metric for filtered BAM file ${FINAL_BAM_FILE_MAPSTATS}<br>• Duplication metrics from MarkDuplicates ${DUP_FILE_QC}<br>• Library complexity measures ${PBC_FILE_QC} |
| **Commands** | # ============================<br># Remove unmapped, mate unmapped<br># not primary alignment, reads failing platform<br># Only keep properly paired reads<br># Obtain name sorted BAM file<br># ==================<br>FILT_BAM_PREFIX="${OFPREFIX}.filt.srt"<br>FILT_BAM_FILE="${FILT_BAM_PREFIX}.bam"<br>TMP_FILT_BAM_PREFIX="tmp.${FILT_BAM_PREFIX}.nmsrt"<br>TMP_FILT_BAM_FILE="${TMP_FILT_BAM_PREFIX}.bam"<br><br>samtools view -F 524 -f 2 -u ${RAW_BAM_FILE} \| samtools sort -n -<br>${TMP_FILT_BAM_MULTIMAP_PREFIX}<br><br>samtools view -h ${TMP_FILT_BAM_MULTIMAP} \| assign_multimappers.py -k $multimapping<br>--paired-end \| samtools view -bS - > ${TMP_FILT_BAM}<br><br>rm -f ${TMP_FILT_BAM_MULTIMAP}<br><br># Remove orphan reads (pair was removed)<br># and read pairs mapping to different chromosomes<br># Obtain position sorted BAM<br><br><br>samtools fixmate -r ${TMP_FILT_BAM_FILE} ${OFPREFIX}.fixmate.tmp<br>samtools view -F 1804 -f 2 -u ${OFPREFIX}.fixmate.tmp \| samtools sort - ${FILT_BAM_PREFIX}<br>rm ${OFPREFIX}.fixmate.tmp<br><br><br>rm ${TMP_FILT_BAM_FILE}<br><br># ==============<br># Mark duplicates<br># ============== |

```
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="${OFPREFIX}.filt.srt.nodup"
FINAL_BAM_FILE="${FINAL_BAM_PREFIX}.bam" # To be stored
FINAL_BAM_INDEX_FILE="${FINAL_BAM_PREFIX}.bai"
FINAL_BAM_FILE_MAPSTATS="${FINAL_BAM_PREFIX}.flagstat.qc" # QC file
FINAL_NMSRT_BAM_PREFIX="${OFPREFIX}.filt.nmsrt.nodup"
FINAL_NMSRT_BAM_FILE="${FINAL_NMSRT_BAM_PREFIX}.bam" # To be stored

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

samtools sort -n ${FINAL_BAM_FILE} ${FINAL_NMSRT_BAM_PREFIX}

# Index Final BAM file
samtools index ${FINAL_BAM_FILE} ${FINAL_BAM_INDEX_FILE}

samtools flagstat ${FINAL_BAM_FILE} > ${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/2.22

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} ${OFPREFIX}.srt.tmp
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))<br><br># And then discard<br>current_reads = [line]<br>current_qname = read_elems[0]<br>else:<br># First read in file<br>current_reads.append(line)<br>current_qname = read_elems[0] |
| **QC to report** | (1) Flagstat output from Final filtered deduped BAM file ${FINAL_BAM_FILE_MAPSTATS}<br>(2) PICARD MarkDup output ${DUP_FILE_QC}<br>http://sourceforge.net/apps/mediawiki/picard/index.php?title=Main_Page#Q:_What_is_meaning_of_the_histogram_produced_by_MarkDuplicates.3F<br>(3) Library complexity measures ${PBC_FILE_QC}<br>● Format of file<br>TotalReadPairs [tab] DistinctReadPairs [tab] OneReadPair [tab] TwoReadPairs [tab] NRF=Distinct/Total [tab] PBC1=OnePair/Distinct [tab] PBC2=OnePair/TwoPair<br>● NRF (non redundant fraction)<br>● PBC1 (PCR Bottleneck coefficient 1)<br>● PBC2 (PCR Bottleneck coefficient 2)<br>● PBC1 is the primary measure. Provisionally,<br>○ 0-0.5 is severe bottlenecking<br>○ 0.5-0.8 is moderate bottlenecking<br>○ 0.8-0.9 is mild bottlenecking<br>○ 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 2.22<br>• gawk<br>• shuf |
| **Input(s)** | • Filtered BAM file ${FINAL_BAM_FILE} |
| **Output(s)** | • tagAlign file ${FINAL_TA_FILE}<br>• Subsampled tagAlign file for CC analysis ${SUBSAMPLED_TA_FILE} |
| **Commands** | # ====================<br># Create tagAlign file<br># ====================<br>module add bedtools/2.22<br><br># Create SE tagAlign file<br>FINAL_TA_FILE="${FINAL_BAM_PREFIX}.SE.tagAlign.gz"<br><br>bedtools bamtobed -i ${FINAL_BAM_FILE} \| awk 'BEGIN{OFS="\t"}{$4="N";$5="1000";print $0}' \| gzip -c > ${FINAL_TA_FILE}<br><br># ================================<br># Subsample tagAlign file<br># ================================<br>NREADS=15000000<br>SUBSAMPLED_TA_FILE="${OFPREFIX}.filt.nodup.sample.$((NREADS / 1000000))M.SE.tagAlign.gz"<br><br>zcat ${FINAL_TA_FILE} \| grep -v “chrM” \| shuf -n ${NREADS} \| gzip -c > ${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 2.22<br>• gawk<br>• shuf |
| **Input(s)** | • Filtered BAM file ${FINAL_BAM_FILE} |
| **Output(s)** | • tagAlign file (virtual single end) ${FINAL_TA_FILE}<br>• BEDPE file (with read pairs on each line) ${FINAL_BEDPE_FILE}<br>• Subsampled tagAlign file for CC analysis ${SUBSAMPLED_TA_FILE} |
| **Commands** | # ====================<br># Create tagAlign file<br># ====================<br>module add bedtools/2.22<br><br># Create virtual SE file containing both read pairs<br>FINAL_TA_FILE="${FINAL_BAM_PREFIX}.PE2SE.tagAlign.gz"<br>bedtools bamtobed -i ${FINAL_BAM_FILE} \| awk 'BEGIN{OFS="\t"}{$4="N";$5="1000";print $0}' \| gzip -c > ${FINAL_TA_FILE} |

| | |
|---|---|
| | # ================<br># Create BEDPE file<br># ================<br>FINAL_BEDPE_FILE="${FINAL_NMSRT_BAM_PREFIX}.bedpe.gz"<br>bedtools bamtobed -bedpe -mate1 -i ${FINAL_NMSRT_BAM_FILE} \| gzip -c ><br>${FINAL_BEDPE_FILE}<br><br># ================================<br># Subsample tagAlign file<br># Restrict to one read end per pair for CC analysis<br># ================================<br>NREADS=15000000<br>SUBSAMPLED_TA_FILE="${OFPREFIX}.filt.nodup.sample.$((NREADS /<br>1000000)).MATE1.tagAlign.gz"<br><br>zcat ${FINAL_BEDPE_FILE} \| grep -v “chrM” \| shuf -n ${NREADS} \| awk 'BEGIN{OFS="\t"}{print<br>$1,$2,$3,"N","1000",$9}' \| gzip -c > ${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-2.10 and above, gawk, samtools, boost C++ libraries, R packages: SPP, caTools, snow

| | |
|---|---|
| **Program(s)** | • phantompeakqualtools (v1.1) |
| **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}<br>• cross-correlation plot ${CC_PLOT_FILE |
| **Commands** | CC_SCORES_FILE="${SUBSAMPLED_TA_FILE}.cc.qc"<br>CC_PLOT_FILE="${SUBSAMPLED_TA_FILE}.cc.plot.pdf"<br><br># CC_SCORE FILE format<br># 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<br><br>Rscript $(which run_spp_nodups.R) -c=${SUBSAMPLED_TA_FILE} -p=${NTHREADS} -filtchr=chrM -savp=${CC_PLOT_FILE} -out=${CC_SCORES_FILE}<br><br>sed -r 's/,[^\t]+//g' ${CC_SCORES_FILE} > temp<br>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<br>• Normalized strand cross-correlation coefficient (NSC) = col9 in outFile<br>• Relative strand cross-correlation coefficient (RSC) = col10 in outFile<br>• Estimated fragment length = col3 in outFile, take the top value<br>• 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<br>• UNIX split<br>• gawk |
| **Input(s)** | • TagAlign file ${FINAL_TA_FILE} |
| **Output(s)** | • 2 pseudoreplicate virtual SE tagAlign files ${PR1_TA_FILE}  ${PR2_TA_FILE} |
| **Commands** | # ========================<br># Create pseudoReplicates<br># ========================<br>PR_PREFIX="${OFPREFIX}.filt.nodup"<br>PR1_TA_FILE="${PR_PREFIX}.SE.pr1.tagAlign.gz"<br>PR2_TA_FILE="${PR_PREFIX}.SE.pr2.tagAlign.gz"<br><br># Get total number of read pairs<br>nlines=$( zcat ${FINAL_TA_FILE} \| wc -l )<br>nlines=$(( (nlines + 1) / 2 ))<br><br># Shuffle and split BED file into 2 equal parts<br>zcat ${FINAL_TA_FILE} \| shuf \| split -d -l ${nlines} - ${PR_PREFIX} # Will produce ${PR_PREFIX}00 and ${PR_PREFIX}01<br><br># Convert reads into standard tagAlign file<br>gzip -c “${PR_PREFIX}00" > ${PR1_TA_FILE}<br>rm "${PR_PREFIX}00"<br>gzip -c “${PR_PREFIX}01" > ${PR2_TA_FILE}<br>rm "${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<br>• UNIX split<br>• gawk |
| **Input(s)** | • BEDPE file ${FINAL_BEDPE_FILE} |
| **Output(s)** | • 2 pseudoreplicate virtual SE tagAlign files ${PR1_TA_FILE}  ${PR2_TA_FILE} |
| **Commands** | # ========================<br># Create pseudoReplicates<br># ========================<br>PR_PREFIX="${OFPREFIX}.filt.nodup"<br>PR1_TA_FILE="${PR_PREFIX}.PE2SE.pr1.tagAlign.gz"<br>PR2_TA_FILE="${PR_PREFIX}.PE2SE.pr2.tagAlign.gz"<br><br># Get total number of read pairs<br>nlines=$( zcat ${FINAL_BEDPE_FILE} \| wc -l )<br>nlines=$(( (nlines + 1) / 2 ))<br><br># Shuffle and split BEDPE file into 2 equal parts |

| | |
|---|---|
| | zcat ${FINAL_BEDPE_FILE} \| shuf \| split -d -l ${nlines} - ${PR_PREFIX} # Will produce ${PR_PREFIX}00 and ${PR_PREFIX}01<br><br># Convert read pairs to reads into standard tagAlign file<br>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,$10}' "${PR_PREFIX}00" \| gzip -c > ${PR1_TA_FILE}<br>rm "${PR_PREFIX}00"<br>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,$10}' "${PR_PREFIX}01" \| gzip -c > ${PR2_TA_FILE}<br>rm "${PR_PREFIX}01" |
| **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.<br>• 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 ${POOLED_TA_FILE}<br>• 2 pooled-pseudoreplicate tagAlign files |
| **Commands** | # ========================<br># Create pooled datasets<br># ========================<br>REP1_TA_FILE="${DATASET_PREFIX}.Rep1.tagAlign.gz"<br>REP2_TA_FILE="${DATASET_PREFIX}.Rep2.tagAlign.gz"<br>POOLED_TA_FILE="${DATASET_PREFIX}.Rep0.tagAlign.gz"<br><br>zcat ${REP1_TA_FILE} ${REP2_TA_FILE} \| gzip -c > ${POOLED_TA_FILE}<br><br># ========================<br># Create pooled pseudoreplicates<br># ========================<br>REP1_PR1_TA_FILE="${DATASET_PREFIX}.Rep1.pr1.tagAlign.gz"<br>REP1_PR2_TA_FILE="${DATASET_PREFIX}.Rep1.pr2.tagAlign.gz"<br><br>REP2_PR1_TA_FILE="${DATASET_PREFIX}.Rep2.pr1.tagAlign.gz"<br>REP2_PR2_TA_FILE="${DATASET_PREFIX}.Rep2.pr2.tagAlign.gz"<br><br>PPR1_TA_FILE="${DATASET_PREFIX}.Rep0.pr1.tagAlign.gz"<br>PPR2_TA_FILE="${DATASET_PREFIX}.Rep0.pr1.tagAlign.gz"<br><br>zcat ${REP1_PR1_TA_FILE} ${REP2_PR1_TA_FILE} \| gzip -c > ${PPR1_TA_FILE}<br>zcat ${REP1_PR2_TA_FILE} ${REP2_PR2_TA_FILE} \| gzip -c > ${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"<br><br>zcat $tag \| awk -F $'\t' 'BEGIN {OFS = FS}{ if ($6 == "+") {$2 = $2 + 4} else if ($6 == "-") {$3 = $3 - 5} print $0}' \| gzip -c > $shifted_tag |
| **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()**

- Two p-value thresholds (-p [P-VAL-THLD] in MACS2 parameter) are used (0.01 and 0.1).
- Used 0.01 for ATAQC and 0.1 for IDR and naive overlap thresholding.

| | |
|---|---|
| **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.<br>Also requires slopBed, bedClip and bedGraphToBigWig from KentTools (ucsc_tools) |
| **Input(s)** | RepN ChIP ${REP1_TA_FILE} ${REP2_TA_FILE}<br>Pooled replicate ${POOLED_TA_FILE}<br>RepN pseudoreplicate1 ${REP*_PR1_TA_FILE}<br>RepN pseudoreplicate2 ${REP*_PR2_TA_FILE}<br>Pooled-pseudoreplicate 1 ${PPR1_TA_FILE}<br>Pooled-pseudoreplicate 2 ${PPR2_TA_FILE} |
| **Output(s)** | Narrowpeak file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.narrowPeak.gz<br>Broadpeak file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.broadPeak.gz<br>Gappedpeak file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.gappedPeak.gz<br>Fold-enrichment bigWig file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.fc.signal.bw<br>-log10(pvalue) bigWig file ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.pval.signal.bw |
| **Commands** | prefix_sig := prefix<br><br>peakfile := "$prefix.narrowPeak.gz"<br>bpeakfile := "$prefix.broadPeak.gz" |

```
gpeakfile    := "$prefix.gappedPeak.gz"

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"

shiftsize := round( smooth_window.parseReal()/2.0 )


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 -c > $bpeakfile
sort -k 14gr,14gr "$prefix"_peaks.gappedPeak | awk 'BEGIN{OFS="\t"}{$4="Peak_"NR ; print
$0}' | gzip -c > $gpeakfile
rm -f "$prefix"_peaks.broadPeak
rm -f "$prefix"_peaks.gappedPeak

macs2 callpeak \
   -t $tag -f BED -n "$prefix" -g "$gensz" -p $pval_thresh \
   --nomodel --shift -$shiftsize --extsize $smooth_window -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}' |
gzip -c > $peakfile
rm -f "$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 $pval_bedgraph_srt<br><br>rm -f "$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 2.22 (intersectBed)<br>blacklist filtering is also integrated in IDR (4.) and naive overlap thresholding (3c.). |
| **Input(s)** | Original peak: ${PEAK}=${PREFIX}.narrowPeak.gz (gappedPeak and broadPeak are also supported)<br>Blacklist: ${BLACKLIST}<br>hg19:<br>http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeMapability/wgEncodeDacMapabilityConsensusExcludable.bed.gz<br>mm9:<br>http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/mm9-mouse/mm9-blacklist.bed.gz<br>GRCh38:<br>http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/hg38-human/hg38.blacklist.bed.gz<br>mm10:<br>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 = "${PREFIX}.narrowPeak.gz"<br>FILTERED_PEAK = "${PREFIX}.narrowPeak.filt.gz"<br><br>bedtools intersect -v -a ${PEAK} -b ${BLACKLIST} \<br>\| awk 'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0}' \<br>\| grep -P 'chr[0-9XY]+(?!_)' \| gzip -nc > ${FILTERED_PEAK} |
| **QC to report** | |
| **Status** | Beta |

**3c Naive overlap thresholding for MACS2 peak calls : 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. We recommend using these for discovery purposes. These are generally high to medium confidence peaks. If you need a very high confidence set use the IDR peaks in the next section.**

| | |
|---|---|
| **Program(s)** | bedtools 2.22 (intersectBed) |

| | |
|---|---|
| **Input(s)** | Peaks (narrowPeak or gappedPeak) for Pooled, Rep1, Rep2, PsRep1 (pseudo replicate 1) and PsRep2 (pseudo replicate 2). |
| **Output(s)** | For narrowPeak finalPeakList.narrowPeak.gz<br>For gappedPeak finalPeakList.gappedPeak.gz |
| **Commands** | # ======================<br># For narrowPeak files<br># ======================<br><br># 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<br><br>intersectBed -wo -a Pooled.narrowPeak.gz -b Rep1.narrowPeak.gz \|<br>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 \|<br>intersectBed -wo -a stdin -b Rep2.narrowPeak.gz \|<br>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<br><br># 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<br><br>intersectBed -wo -a Pooled.narrowPeak.gz -b PsRep1.narrowPeak.gz \|<br>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 \|<br>intersectBed -wo -a stdin -b PsRep2.narrowPeak.gz \|<br>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 > PooledInPsRep1AndPsRep2.narrowPeak.gz<br><br># Combine peak lists<br><br>zcat PooledInRep1AndRep2.narrowPeak.gz PooledInPsRep1AndPsRep2.narrowPeak.gz \|<br>sort \| uniq \| awk 'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0}' \\<br>\| grep -P 'chr[0-9XY]+(?!_)' > finalPeakList.narrowPeak.gz<br><br><br># ======================<br>For BroadPeak files (there is just a difference is the awk commands wrt the column numbers)<br># ======================<br><br># 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<br><br>intersectBed -wo -a Pooled.broadPeak.gz -b Rep1.broadPeak.gz \|<br>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 \|<br>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

The IDR peaks are a subset of the naive overlap peaks that pass a specific IDR threshold of 10%.

**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 10% 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.

- Top 500K lines are picked up from peaks generated from MACS2 (with p-value threshold 0.1)

| | |
|---|---|
| **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}<br>• a pooled-replicate narrowPeak file ${POOLED_PEAK_FILE}<br>• a blacklist ${BLACKLIST}<br>  ○ hg19: http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeMapability/wgEncodeDacMapabilityConsensusExcludable.bed.gz<br>  ○ mm9: http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/mm9-mouse/mm9-blacklist.bed.gz<br>  ○ GRCh38: http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/hg38-human/hg38.blacklist.bed.gz<br>  ○ mm10: http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/mm10-mouse/mm10.blacklist.bed.gz<br>• For SPP use signal.value for ranking with a parameter '--rank signal.value'<br>• 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<br>• The full set of peaks that overlap between the replicates with local and global IDR: suffixed by overlapped-peaks.txt ${IDR_OUTPUT}<br>• IDR output file ${IDR_OUTPUT}<br>  ○ # Columns 1-10 are same as pooled common peaks narrowPeak columns<br>  ○ # Col 11: -log10(local IDR value)<br>  ○ # Col 12: -log10(global IDR value)<br>  ○ # Col 15: ranking measure from Rep1<br>  ○ # Col 19: ranking measure from Rep2<br>• Final IDR thresholded file ${REP1_VS_REP2}.IDR0.05.narrowPeak.gz<br>• Final IDR thresholded file filtered using a blacklist ${REP1_VS_REP2}.IDR0.05.filt.narrowPeak.gz |
| **Commands** | IDR_THRESH=0.1 # OTHERWISE<br>IDR_THRESH=0.05  # FOR ENCODE3<br><br># ==============================<br># Perform IDR analysis.<br># Generate a plot and IDR output with additional columns including IDR scores.<br># ==============================<br>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 ${IDR_THRESH} --plot --use-best-multisummit-IDR<br><br># ==============================<br># Get peaks passing IDR threshold of 10%<br># ==============================<br>IDR_THRESH_TRANSFORMED=$(awk -v p=${IDR_THRESH} 'BEGIN{print -log(p)/log(10)}')<br><br>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 -c > ${REP1_VS_REP2}.IDR0.05.narrowPeak.gz<br><br>NPEAKS_IDR=$(zcat ${REP1_VS_REP2}.IDR0.05.narrowPeak.gz \| wc -l)<br><br># ==============================<br># Filter using black list<br># ==============================<br>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[0-9XY]+(?!_)'  \| gzip -c > ${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<br>• --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<br>• --soft-idr-threshold : IDR threshold, Set to ${IDR_THRESH} |
| **QC to report** | • Number of peaks passing IDR thresholds of 10% ${NPEAKS_IDR}<br>• 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.<br>• Also store ${POOLED_COMMON_PEAKS_IDR}<br>• 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 10% 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)**
You compute the fraction of reads from each replicate tagAlign and pooled tagAlign that fall within the Np and Nt peak sets
Inputs:
Final tagAlign file for Rep1 ${REP1_TA_FILE}
Final tagAlign file for Rep2 ${REP2_TA_FILE}
Pooled tagAlign file zcat ${REP1_TA_FILE} ${REP2_TA_FILE} | gzip -c > ${POOLED_TA_FILE}
(Nt) Conservative IDR peak set ${CONS_IDR_PEAKS}
(Np) Optimal IDR peak set ${OPTIMAL_IDR_PEAKS}

<TODO>

# 5. ATAQC

**ATAQC : function _ataqc()**

| | |
|---|---|
| **Program(s)** | ATAQC: Computes a variety of QC metrics and generates user friendly HTML report |
| **Input(s)** | • for SE: fastq, bam, align_log, pbc_qc, dup_qc, filt_bam, tagalign, pval_signal, peak, peak_overlap, idr_opt<br>• for PE: fastq1, fastq2, bam, align_log, pbc_qc, dup_qc, filt_bam, tagalign, pval_signal, peak, peak_overlap, idr_opt |
| **Output(s)** | • HTML, log |
| **Commands** | # sort bam first<br><br>srt_bam = “$prefix.srt.bam”<br>srt_bam_prefix = “$prefix.srt”<br><br>samtools sort $bam $srt_bam_prefix<br>samtools index $srt_bam<br><br># run ATAQC<br>run_ataqc.py \<br>--workdir $o_dir \<br>--outdir $o_dir \<br>--outprefix $prefix_basename \<br>--genome $species \ |

| | |
|---|---|
| | --ref $ref_fa \<br>--tss $tss_enrich \<br>--dnase $dnase \<br>--blacklist $blacklist \<br>--prom $prom \<br>--enh $enh \<br>--reg2map $reg2map \<br>--meta $roadmap_meta \<br>--pbc $pbc_log\<br>--alignedbam $bam \<br>--alignmentlog $align_log \<br>--coordsortbam $srt_bam \<br>--duplog $dup_log \<br>--finalbam $filt_bam \<br>--finalbed $bed \<br>--bigwig $bigwig \<br>--peaks $peak \<br>--naive_overlap_peaks $peak_overlap \<br>--idr_peak $idr_opt \<br>--fastq1 $fastq1 --fastq2 $fastq2<br><br># for SE, the last parameter line must be --fastq1 $fastq1 |
| **Parameters** | • --workdir: where all the output files are<br>• --outdir: where all the QC files should be stored<br>• --outprefix: prefix for output QC files<br>• --genome: which genome to compare against (currently supported: hg19, mm9)<br>• --ref: reference fasta which MUST be the one used in alignment<br>• --tss: a BED file of TSS's, used for getting the TSS enrichment<br>• --dnase: BED file of universal DHS regions<br>• --blacklist: BED file of blacklist regions<br>• --prom: BED file of promoter regions<br>• --enh: BED file of enhancer regions<br>• --reg2map: matrix file of signal across DNase samples<br>• --meta: metadata file (only needed for human)<br>• --pbc: PBC QC output from BDS pipeline<br>• --alignedbam: mapped file with no filtering<br>• --alignmentlog: Bowtie alignment log<br>• --coordsortbam: sorted BAM file<br>• --duplog: duplicates log from Picard MarkDuplicates<br>• --finalbam: final filtered BAM file<br>• --finalbed: final filtered BED file, shifted for tn5 if ATAC (not for DNase)<br>• --bigwig: signal file (pval signal)<br>• --peaks: raw peaks<br>• --naive_overlap_peaks: naive overlap peaks<br>• --idr_peaks: IDR peaks<br>• --fastq1: fastq file (read 1)<br>• |
| **QC to report** | • Alignment statistics: mapping rate, etc<br>• Filtering statistics: how many reads are lost due to various filters (duplicates, chrM for example)<br>• Library complexity statistics: ENCODE (NRF, PBC), Picard EstimateLibraryComplexity, Preseq |

| | |
|---|---|
| | • Fragment length statistics: paired end data only<br>• Peak statistics: quartiles, peak numbers, region size distributions<br>• GC bias<br>• TSS enrichment<br>• Annotation enrichments: universal DHS, blacklist, promoter, enhancer, called peaks<br>• Comparison to available DNase data<br>• |
| **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
   }

   if ( csem ) {
         ( bam, align_log ) = _bowtie2_csem( p1 )
   }
   else {
         ( bam, align_log ) = _bowtie2( p1 )
   }

   ( 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 = 0.1)
    ( 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 = 0.1)
    ( peak, gpeak, pval_bigwig ) = _macs2( final_tag )

    // call peaks on true replicate (with p_val_thresh = 0.01)
    ( peak001, gpeak001, pval001_bigwig ) = _macs2( final_tag, 0.01 )

    // 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, align_log ) = _bowtie2_PE( p1, p2 )

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

bedpe = _bam_to_bedpe( filt_bam )

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

tag = _bedpe_to_tag( subsampled_bedpe )

final_tag = _tn5_shift_tag( tag )

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

final_tag_pr1 = _tn5_shift_tag( tag_pr1 )
final_tag_pr2 = _tn5_shift_tag( tag_pr2 )

// call peaks on pseudo replicate (with p_val_thresh = 0.1)
( 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 = 0.1)
( peak, gpeak, pval_bigwig ) = _macs2( final_tag )

// call peaks on true replicate (with p_val_thresh = 0.01)
( peak001, gpeak001, pval_bigwig001 ) = _macs2( final_tag, 0.01 )

// 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 )
```

```
}


void ataqc( replicate ) {

        sorted_bam = _srt_bam( bam )

        if ( no_rep == 1 ) idr_peak = idr_pr_rep1
        else               idr_peak = idr_opt

        if ( se ) { // for single-ended data set
                _ataqc( fastq, "", bam, align_log, pbc_qc, sorted_bam, dup_qc, \
                        filt_bam, final_tag, pval_bigwig001, peak001, \
                        idr_peak, peak_overlap  )
        }
        else { // for paired end data set
                _ataqc( fastq1, fastq2, bam, align_log, pbc_qc, sorted_bam, dup_qc, \
                        filt_bam, final_tag, pval_bigwig001, peak001, \
                        idr_peak, peak_overlap )
        }
}

function main() {

    // By default, smoothing window for MACS2 is 150. IDR threshold is 0.1
    // for ENCODE3, smoothing window for MACS2 is 73. IDR threshold is 0.05
    if ( ENCODE3 ) {
        smooth_win = 73
        idr_thresh = 0.05
    }
    else {
        smooth_win = 150
        idr_thresh = 0.1
    }

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

    for ( rep = 1; rep <= no_rep; rep++ ) {
```

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


// 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 )

// 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 )

// take top 500K peaks

filt_peak_rep1     = _filt_top_peaks( peak_rep1 )
filt_peak_rep2     = _filt_top_peaks( peak_rep2 )
filt_peak_pooled    = _filt_top_peaks( peak_pooled )

filt_peak_pr1_rep1 = _filt_top_peaks( peak_pr1_rep1 )
filt_peak_pr2_rep1 = _filt_top_peaks( peak_pr2_rep1 )
filt_peak_pr1_rep2 = _filt_top_peaks( peak_pr1_rep2 )
filt_peak_pr2_rep2 = _filt_top_peaks( peak_pr2_rep2 )

// take top 500K gapped peaks

filt_gpeak_ppr1             = _filt_top_peaks( gpeak_ppr1 )
filt_gpeak_ppr2             = _filt_top_peaks( gpeak_ppr2 )

filt_gpeak_rep1     = _filt_top_peaks( gpeak_rep1 )
filt_gpeak_rep2     = _filt_top_peaks( gpeak_rep2 )
filt_gpeak_pooled    = _filt_top_peaks( gpeak_pooled )
```

```
filt_gpeak_pr1_rep1 = _filt_top_peaks( gpeak_pr1_rep1 )
filt_gpeak_pr2_rep1 = _filt_top_peaks( gpeak_pr2_rep1 )
filt_gpeak_pr1_rep2 = _filt_top_peaks( gpeak_pr1_rep2 )
filt_gpeak_pr2_rep2 = _filt_top_peaks( gpeak_pr2_rep2 )

filt_gpeak_ppr1          = _filt_top_peaks( gpeak_ppr1 )
filt_gpeak_ppr2          = _filt_top_peaks( gpeak_ppr2 )

// naive overlap

if ( no_rep == 1 ) {
    peak_overlap = _naive_overlap_peak( peak_rep1, \
                                        peak_pr1_rep1, peak_pr2_rep1 )
    gpeak_overlap = _naive_overlap_peak( gpeak_rep1, \
                                        gpeak_pr1_rep1, gpeak_pr2_rep1 )
}
else {
    peak_overlap = _naive_overlap_peak( peak_pooled, \
                                        peak_rep1, peak_rep2, \
                                        peak_ppr1, peak_ppr2 )
    gpeak_overlap = _naive_overlap_peak( gpeak_pooled, \
                                        gpeak_rep1, gpeak_rep2, \
                                        gpeak_ppr1, gpeak_ppr2 )
}

// IDR

idr_tr = _idr( filt_peak_rep1, filt_peak_rep2, filt_peak_pooled )

idr_pr_rep1 = _idr( filt_peak_pr1_rep1, filt_peak_pr2_rep1, filt_peak_rep1 )
idr_pr_rep2 = _idr( filt_peak_pr1_rep2, filt_peak_pr2_rep2, filt_peak_rep2 )

idr_ppr = _idr( filt_peak_ppr1, filt_peak_ppr2, filt_peak_pooled )

( idr_qc, idr_opt, idr_consv ) = _idr_final_qc( idr_tr, idr_pr_rep1, idr_pr_rep2, idr_ppr )

// ATAQC

for ( rep = 1; rep <= no_rep; rep++ ) ataqc( rep )
```

```
}
```