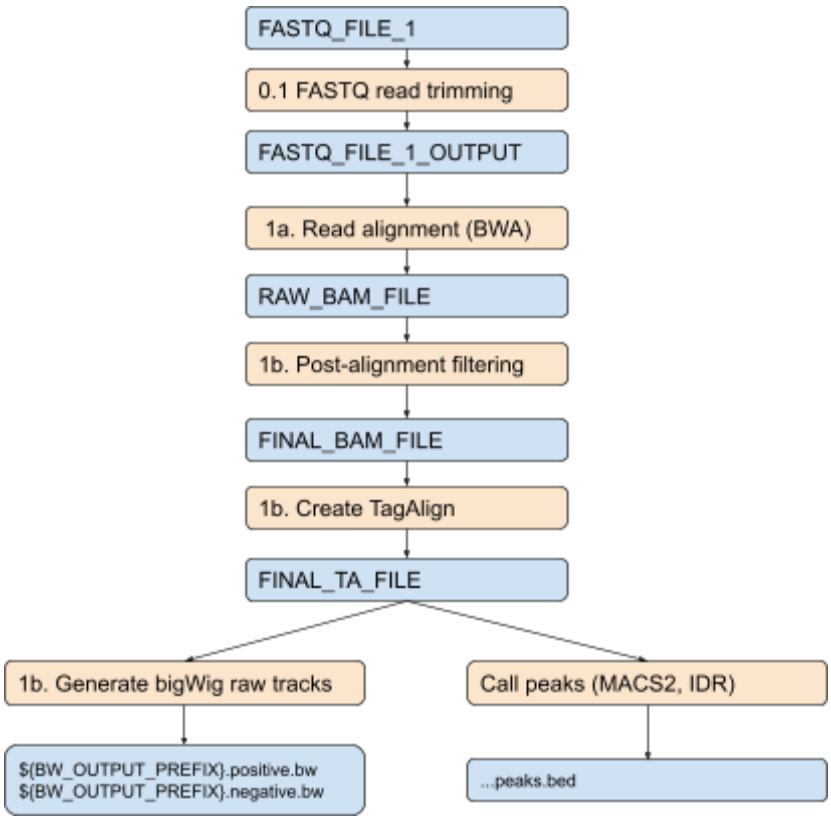
ENCODE3 pipeline v1 specifications

Pipeline Overview



0. FASTQ read quality filtering

Optional, was not specified

- FastQC can show the distribution of sequence quality scores and to help set an appropriate cutoff: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/3%20Analysis%20Modules/3%20Per%20Sequence%20Quality%20Scores.html)
- The -q parameter in BWA does read trimming so it can also be used to remove bad parts of reads

0.1 FASTQ read trimming

Single-End ChIP-nexus parameters

- Cutadapt Parameters:
  - Minimum read length after processing m=18
  - Minimum overlap for adapter to need to trim O=4
  - Maximum error rate allowed for adapter matching e=0.2
  - Suppress output --quiet
- **nimnexus trim [options] <barcode>**
  - **-t/--trim: Pre-trim all reads by this length before processing [default: 0]**
  - **-k/--keep: Minimum number of bases required after barcode to keep read [default: 18]**
  - **-r/--randombarcode: Number of bases at the start of each read used for random barcode [default: 5]**

Program(s)	• Nimnexus 0.1.1 (install via conda -c bioconda nimnexus==0.1.1) • Cutadapt 1.8.1 • Pigz (we can freeze the current version)
Input(s)	• FASTQ file ${FASTQ_FILE_1} • Fixed barcode sequences, comma delimited ${BARCODES} (required as CLI args) • Adapter sequence ${ADAPTER_SEQUENCE}, default=AGATCGGAAGAGCACACGTCTGGATCCACGACGCTCTTCC • Total number of available threads: ${THREADS}
Output(s)	• FASTQ file ${FASTQ_FILE_1_OUTPUT}
Commands	# == # Trim reads, filter based on presence of fixed barcode, assign random barcode to FASTQ read name, Trim adapters off FASTQ files output processed FASTQ file. # == pigz -cd ${FASTQ_FILE_1} | nimnexus trim ${BARCODES} | parallel -j ${THREADS} --pipe -L 4 --block 10M \\ cutadapt -m 22 --overlap=4 -e 0.2 --quiet \\ -a ${ADAPTER_SEQUENCE} \\ -a ${ADAPTER_SEQUENCE:1} \\ -a ${ADAPTER_SEQUENCE:2} \\ -a ${ADAPTER_SEQUENCE:3} \\ -a ${ADAPTER_SEQUENCE:4} - | pigz -c ${FASTQ_FILE_1_OUTPUT}
QC to report	Fraction of sequences trimmed
Status	

1a. Read alignment (BWA aligner)

Single-End ChIP-seq parameters

- With dynamic read trimming q = 5
- seed length l = 32
- max. mismatches in seed k = 2

Program(s)	• BWA version 0.7.13 • SAMtools (version 1.2)
Input(s)	• FASTQ file ${FASTQ_FILE_1} • hg19 male or female or mm9 reference sequence ${BWA_INDEX_NAME}
Output(s)	• BAM file ${RAW_BAM_FILE} • mapping stats from flagstat ${RAW_BAM_FILE_MAPSTATS}
Commands	# ==================================== # Map reads to create raw SAM file # ==================================== SAI_FILE_1="${OFPREFIX}.sai" RAW_BAM_PREFIX="${OFPREFIX}.raw.srt" RAW_BAM_FILE="${RAW_BAM_PREFIX}.bam" # To be stored RAW_BAM_FILE_MAPSTATS="${RAW_BAM_PREFIX}.flagstat.qc" # QC File module add bwa/0.7.13 module add samtools/1.2 bwa aln -q 5 -l 32 -k 2 -t ${NTHREADS} ${BWA_INDEX_NAME} ${FASTQ_FILE_1} > ${SAI_FILE_1} bwa samse ${BWA_INDEX_NAME} ${SAI_FILE_1} ${FASTQ_FILE_1} | samtools view -Su - | samtools sort - ${RAW_BAM_PREFIX} rm ${SAI_FILE_1} samtools flagstat ${RAW_BAM_FILE} > ${RAW_BAM_FILE_MAPSTATS}
QC to report	Output from last command i.e. samtools flagstat
Status	Frozen

1b. Post-alignment filtering

Single-End ChIP-nexus parameters

- **nimnexus dedup [options] <BAM>. Default parameters**
  - **-t: number of BAM decompression threads [default: 2]**

Program(s)	• Nimnexus 0.1.1 (install via conda -c bioconda nimnexus==0.1.1) • SAMtools (version 1.2) • MarkDuplicates (Picard - version 1.126) • bedtools 2.26.0
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 ${PBC_FILE_QC}
Commands	# ============================ # Remove unmapped, mate unmapped # not primary alignment, reads failing platform # Remove low MAPQ reads # ================== FILT_BAM_PREFIX="${OFPREFIX}.filt.srt" FILT_BAM_FILE="${FILT_BAM_PREFIX}.bam" MAPQ_THRESH=30 samtools view -F 1804 -q ${MAPQ_THRESH} -b ${RAW_BAM_FILE} -o ${FILT_BAM_FILE} # ======================== # Mark duplicates # ====================== module add picard-tools/1.126 TMP_FILT_BAM_FILE="${FILT_BAM_PREFIX}.dupmark.bam" MARKDUP="/srv/gs1/software/picard-tools/1.126/picard MarkDuplicates" 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}.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} -o ${FINAL_BAM_FILE}

	# Index Final BAM file samtools index ${FINAL_BAM_FILE} ${FINAL_BAM_INDEX_FILE} samtools flagstat ${FINAL_BAM_FILE} > ${FINAL_BAM_FILE_MAPSTATS} # ============================ # Compute library complexity # ============================ # sort by position and strand # Obtain unique count statistics module add bedtools/2.26.0 PBC_FILE_QC="${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}
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 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

Program(s)	• Nimnexus 0.1.1 (install via conda -c bioconda nimnexus==0.1.1) • Samtools
Input(s)	• Raw BAM file ${RAW_BAM_FILE} • Total number of available threads: ${THREADS}
Output(s)	• Filtered deduped position sorted BAM and index file ${FINAL_BAM_FILE} ${FINAL_BAM_INDEX_FILE}
Commands	# == # 1. Sort the bam file SORTED_RAW_BAM_FILE=${RAW_BAM_FILE}

	samtools sort -o ${SORTED_RAW_BAM_FILE} ${RAW_BAM_FILE} # 2. Run nimnexus dedup nimnexus dedup -t ${THREADS} ${SORTED_RAW_BAM_FILE} | \ samtools view -b > ${FINAL_BAM_FILE} # 3. Index the bam file samtools index ${FINAL_BAM_FILE} ${FINAL_BAM_INDEX_FILE}
QC to report	Fraction of reads removed
Status	

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

Program(s)	• bedtools 2.26.0 • gawk • shuf
Input(s)	• Filtered BAM file ${FINAL_BAM_FILE}
Output(s)	• tagAlign file ${FINAL_TA_FILE} • Subsampled tagAlign file for CC analysis ${SUBSAMPLED_TA_FILE}
Commands	# ==================== # Create tagAlign file # ==================== module add bedtools/2.26.0 # Create SE tagAlign file FINAL_TA_FILE="${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=15000000 SUBSAMPLED_TA_FILE="${OFPREFIX}.filt.nodup.sample.$((NREADS / 1000000)).SE.tagAlign.gz" zcat ${FINAL_TA_FILE} | grep -v “chrM” | shuf -n ${NREADS} --random-source=${FINAL_TA_FILE} | gzip -nc > ${SUBSAMPLED_TA_FILE}
QC to report	None
Status	Frozen

2a. Convert PE BAM to tagAlign (BED 3+3 format)

Program(s)	• bedtools 2.26.0 • gawk • shuf
Input(s)	• Filtered BAM file ${FINAL_BAM_FILE}
Output(s)	• tagAlign file (virtual single end) ${FINAL_TA_FILE} • BEDPE file (with read pairs on each line) ${FINAL_BEDPE_FILE} • Subsampled tagAlign file for CC analysis ${SUBSAMPLED_TA_FILE}
Commands	# ================== # Create tagAlign file # ================== module add bedtools/2.26.0 # ================ # Create BEDPE file # ================ FINAL_BEDPE_FILE="${FINAL_NMSRT_BAM_PREFIX}.bedpe.gz" bedtools bamtobed -bedpe -mate1 -i ${FINAL_NMSRT_BAM_FILE} | gzip -nc > ${FINAL_BEDPE_FILE} 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 > ${FINAL_TA_FILE} # ================================ # Subsample tagAlign file # Restrict to one read end per pair for CC analysis # ================================ NREADS=15000000 SUBSAMPLED_TA_FILE="${OFPREFIX}.filt.nodup.sample.$((NREADS / 1000000)).MATE1.tagAlign.gz" zcat ${FINAL_BEDPE_FILE} | grep -v “chrM” | shuf -n ${NREADS} --random-source=${FINAL_TA_FILE} | 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

- Code package: https://code.google.com/p/phantompeakqualtools/ (Updated version is imminent)
- Dependencies: unix, bash, R-3.20 and above, gawk, samtools, boost C++ libraries, R packages: SPP, caTools, snow

Program(s)	• phantompeakqualtools (v1.2)
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)

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 ${FINAL_TA_FILE} \| shuf --random-source=${FINAL_TA_FILE} | 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)

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/' > $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 ${joined} \| shuf --random-source=${FINAL_TA_FILE} | split -d -l ${nlines} - ${PR_PREFIX} # Will produce ${PR_PREFIX}00 and ${PR_PREFIX}01 # Convert fake BEDPE 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 -f ${joined}
QC to report	None
Status	Frozen

2d. Generate pooled dataset and pooled-pseudoreplicates

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.pr2.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

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 3 peak callers SPP, GEM and PeakSeq

Pooling controls: If control datasets (input DNA or Igg) have replicates as far as possible match ChIP replicates to appropriate control replicates. However, under some conditions listed below, its best to pool the control replicates.

- If the no. of reads per replicate is < the no. of reads per ChIP replicate then pool the control replicate reads into a single control
- If the no. of reads between control replicates differ by > a factor of 1.2, then pool replicates (this is to avoid artificial differences in peak scores due to sequencing depth differences in different control replicates)

Pooled-replicates or pooled-pseudoreplicates should always be compared to pooled controls
Self-pseudoreplicates for a particular ReplicateN should be compared to the same control that was used for ReplicateN.

Generate COUNT bigwig tracks for ONLY TRUE REPS and POOLED REPS

Program(s)	• bedtools • UCSC bedGraphToBigWig
Input(s)	• Genome file ${CHRSIZEFILE} • tagAlign file ${REP1_TA_FILE} ${REP2_TA_FILE} ${POOLED_TA_FILE}
Output(s)	• Single-resolution BigWig files for positive and negative strand coverage ${BW_OUTPUT_PREFIX}.positive.bw ${BW_OUTPUT_PREFIX} .negative.bw
Commands	# ========== # positive strand zcat ${FINAL_TA_FILE} | \ bedtools genomecov -5 -bg -strand + -g ${CHRSIZEFILE} -i stdin | \ sort -k1,1 -k2,2n > ${BW_OUTPUT_PREFIX}.positive.bedGraph # negative strand zcat ${FINAL_TA_FILE} | \ bedtools genomecov -5 -bg -strand - -g ${CHRSIZEFILE} -i stdin | \ sort -k1,1 -k2,2n > ${BW_OUTPUT_PREFIX}.negative.bedGraph bedGraphToBigWig ${BW_OUTPUT_PREFIX}.positive.bedGraph ${CHRSIZEFILE} ${BW_OUTPUT_PREFIX}.positive.bw bedGraphToBigWig ${BW_OUTPUT_PREFIX}.negative.bedGraph ${CHRSIZEFILE} ${BW_OUTPUT_PREFIX}.negative.bw

QC to report	NA
Status	

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

Use IDR to compare all pairs of matched replicates
(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.

- ~~Code: https://sites.google.com/site/anshulkundaje/projects/idr~~
- ~~R scripts batch-consistency-analysis.r and batch-consistency-plot.r~~

~~THIS USES THE OLD IDR CODE (THIS IS DEPRECATED .. SEE NEXT SECTION FOR NEW COMMANDS USING IDR CODE)~~

~~Program(s)~~	• ~~batch-consistency-analysis.r **(DEPRECATED)**~~
~~Input(s)~~	• ~~a pair of narrowPeak files for replicates ${REP1_PEAK_FILE} ${REP2_PEAK_FILE}~~ • ~~a pooled-replicate narrowPeak file ${POOLED_PEAK_FILE}~~ • ~~For SPP use signal.value for ranking~~ • ~~For GEM use signal.value for ranking~~ • ~~For PeakSeq use q.value for ranking~~
~~Output(s)~~	• ~~The output from EM fitting: suffixed by -em.sav~~ • ~~The output for plotting empirical curves: suffixed by -uri.sav~~ ~~Note: 1 and 2 are objects that can be loaded back to R for plotting or other purposes (e.g. retrieve data)~~ • ~~The parameters estimated from EM and the log of consistency analysis, suffixed by -Rout.txt~~ • ~~The number of peaks that pass specific IDR thresholds for the pairwise analysis: suffixed by npeaks-aboveIDR.txt~~ • ~~The full set of peaks that overlap between the replicates with local and global IDR: suffixed by overlapped-peaks.txt~~ • ~~IDR output file ${POOLED_COMMON_PEAKS_IDR}~~ ○ ~~# Columns 1-10 are same as pooled common peaks narrowPeak columns~~ ○ ~~# Col 11: ranking measure from Rep1~~ ○ ~~# Col 12: ranking measure from Rep2~~ ○ ~~# Col 13: local IDR score~~ ○ ~~# Col 14: global IDR score~~ • ~~Final IDR thresholded file ${REP1_VS_REP2}.IDR0.02.narrowPeak.gz~~

Commands	
	# ========== # Find peaks in pooled set common to both replicates # ========== bedtools intersect -u -a ${POOLED_PEAK_FILE} -b ${REP1_PEAK_FILE} | bedtools intersect -u -a stdin -b ${REP2_PEAK_FILE} | sort -k7n,7n -k1,1 -k2n,2n -k3n,3n -k10n,10n | gzip -nc > ${POOLED_COMMON_PEAKS} # ========== # Create 2 new peak file per replicate with coordinates from common pooled set but scores from replicates by first computing overlaps, then then matching to closest summit, then recalibration coordinates to be +/- 2 bp from pooled set summit # ========== bedtools intersect -wa -wb -a ${POOLED_COMMON_PEAKS} -b ${REP1_PEAK_FILE} | awk 'BEGIN{OFS="\t"}{d=$2+$10-$12-$20;$21=sqrt(d^2);print $0}' | bedtools groupby -i stdin -g 1,2,3,10 -c 21 -o min -full | sort -k7n,7n -k1,1 -k2n,2n -k3n,3n -k10n,10n | awk 'BEGIN{OFS="\t"}{print $1,$2+$10-2,$2+$10+2,$4,$5,$6,$17,$18,$19,2}' | gzip -nc > ${COMMON_REP1_MATCH} bedtools intersect -wa -wb -a ${POOLED_COMMON_PEAKS} -b ${REP2_PEAK_FILE} | awk 'BEGIN{OFS="\t"}{d=$2+$10-$12-$20;$21=sqrt(d^2);print $0}' | bedtools groupby -i stdin -g 1,2,3,10 -c 21 -o min -full | sort -k7n,7n -k1,1 -k2n,2n -k3n,3n -k10n,10n | awk 'BEGIN{OFS="\t"}{print $1,$2+$10-2,$2+$10+2,$4,$5,$6,$17,$18,$19,2}' | gzip -nc > ${COMMON_REP2_MATCH} # ========== # Pass recalibrated peak files to IDR # Rscript batch-consistency-analysis.r [peakfile1] [peakfile2] [peak.half.width] [outfile.prefix] [min.overlap.ratio] [is.broadpeak] [ranking.measure] # For SPP & GEM use [ranking.measure] as signal.value. For PeakSeq use q.value # make sure the genome_table.txt file is set to the correct version of the genome # ========== Rscript batch-consistency-analysis.r ${COMMON_REP1_MATCH} ${COMMON_REP2_MATCH} -1 ${REP1_VS_REP2} 0 F signal.value # ========== # Convert IDR overlap file to narrowPeak format # ========== sed 1d ${REP1_VS_REP2}-overlapped-peaks.txt | sed -r 's/"//g' | sort -k11g,11g | awk '{if ($3 <=$7) st=$3 ; else st=$7 ; if ($4 >= $8) sto=$4 ; else sto=$8 ; printf "%s\t%d\t%d\t%s\t.\t%s\t%f\t%f\n",$2,st,sto,NR,$5,$9,$10,$11}' | gzip -nc > ${REP1_VS_REP2}.npk.gz # ========== # Create a recalibrated vesion of ${POOLED_COMMON_PEAKS} where coordinates are to be +/- 2 bp from pooled set summit. This is so we can match it with IDR output and retranslate back to original pooled set coordinates # ========== zcat ${POOLED_COMMON_PEAKS} | awk 'BEGIN{OFS="\t"}{$11=$2;$12=$3;$2=$2+$10-2;$3=$2+$10+2; print $0}' | gzip -nc >

	${RECAL_POOLED_COMMON_PEAKS} # ============================ # Overlap IDR output with ${RECAL_POOLED_COMMON_PEAKS} to add in IDR scores and switch back to original common pooled set coordinates # Columns 1-10 are same as pooled common peaks columns # Col 11: ranking measure from Rep1 # Col 12: ranking measure from Rep2 # Col 13: local IDR score # Col 14: global IDR score # IMPORTANT: DCC should store this file! All other files are temporary # ============================ bedtools intersect -wa -wb -a ${RECAL_POOLED_COMMON_PEAKS} -b ${REP1_VS_REP2}.npk.gz | awk 'BEGIN{OFS="\t"}{print $1,$11,$12,$4,$5,$6,$7,$8,$9,$10,$17,$19,$20,$21}' | gzip -nc > ${POOLED_COMMON_PEAKS_IDR} # ============================ # Get peaks passing IDR threshold of 2% # ============================ IDR_THRESH=0.02 zcat ${POOLED_COMMON_PEAKS_IDR} | awk 'BEGIN{OFS="\t"} $14 <="${IDR_THRESH}" {print $1,$2,$3,$4,$5,$6,$7,$8,$9,$10}' | sort -k7n,7n | gzip -nc > ${REP1_VS_REP2}.IDR0.02.narrowPeak.gz NPEAKS_IDR=$(zcat ${REP1_VS_REP2}.IDR0.02.narrowPeak.gz | wc -l)
Parameters	• [peakfile1] and [peakfile2] are the peak calls for the pair of replicates in narrowPeak format. They must be uncompressed files. e.g. /peaks/reps/chipSampleRep1_VS_controlSampleRep0.narrowPeak AND /peaks/reps/chipSampleRep2_VS_controlSampleRep0.narrowPeak • [peak.half.width]: Set this to -1 if you want to use the reported peak width in the peak files. (-1 is the only value that works for this parameter right now) • [outfile.prefix] is a prefix that will be used to name the output data for this pair of replicates. The prefix must also include the PATH to the directory where you want to store the output data. e.g. /consistency/reps/chipSampleRep1_VS_chipSampleRep2 • [min.overlap.ratio]: fractional bp overlap (ranges from 0 to 1) between peaks in replicates to be considered as overlapping peaks. Set to 0 if you want to allow overlap to be defined as >= 1 bp overlap. If set to say 0.5 this would mean that atleast 50% of the peak in one replicate should be covered by a peak in the other replicate to count as an overlap. IMPORTANT: This parameter has not been tested fully. It is recommended to set this to 0. • [is.broadpeak]: Is the peak file format narrowPeak or broadPeak. Set to F if it is narrowPeak/regionPeak or T if it is broadPeak. BroadPeak files do not contain Column 10. • [ranking.measure] is the ranking measure to use. It can take only one of the following values signal.value , p.value or q.value • IDR threshold ${IDR_THRESH}
QC to report	• Number of peaks passing IDR thresholds of 2% ${NPEAKS_IDR} • For each pairwise analysis, we have a *overlapped-peaks.txt file. The last column (Column 11) 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**~~	~~**(DEPRECATED)**~~

~~Plot IDR consistency plots~~

~~**Program(s)**~~	• ~~batch-consistency-plot.r~~
~~**Input(s)**~~	• ~~narrowPeak files from SPP peak caller, run for each biological replicate combinations (if more than 2 replicates)~~
~~**Output(s)**~~	• ~~Summary consistent plot in .ps format (suffixed _plot.ps)~~
~~**Commands**~~	~~# ============================~~ ~~# Format: Rscript batch-consistency-plot.r [npairs] [output.prefix] [input.file.prefix1] [input.file.prefix2] [input.file.prefix3] ....~~ ~~# ============================~~ ~~Rscript batch-consistency-plot.r 1 ${OUTPUT_PREFIX} ${REP1_VS_REP2}~~
~~**Parameters**~~	• ~~[n.pairs] is the number of pairs of replicates that you want to plot on the same plot e.g. 1 or 3 or ...~~ • ~~The prefix must also include the PATH to the directory where you want to store the output data. e.g. /consistency/plots/chipSampleAllReps~~
~~**QC to report**~~	• ~~IDR consistency plots between replicates~~
~~**Status**~~	~~**(DEPRECATED)**~~

- ~~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**~~

THIS IS THE NEW VERSION OF IDR (PLEASE USE THIS)

Program(s)	IDR (https://github.com/kundajelab/idr) 2.0.4 / 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 signal.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} | grep -P 'chr[\dXY]+[\t]' | awk 'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0}' | 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:
  hg19:
  http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeMapability/wgEncodeDacMapabilityConsensusExcludable.bed.gz
  hg38:

http://mitra.stanford.edu/kundaje/genome_data/hg38/hg38.blacklist.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>

4f. Compute Fraction of Reads in Peaks (FRiP) : function frip(ta_file, cc_qc_log, 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 2.26.0
Input(s)	Final tagAlign file for Rep1 ${REP1_TA_FILE} vs. IDR peak from pseudo replicates of Rep1 ${REP1_PR_IDR_PEAK_FILE} with estimated fragment length for replicate 1 Final tagAlign file for Rep2 ${REP2_TA_FILE} vs. IDR peak from pseudo replicates of Rep2 ${REP2_PR_IDR_PEAK_FILE} with estimated fragment length for replicate 2 Pooled tagAlign file ${POOLED_TA_FILE} vs. IDR peak from true replicates (Nt) ${CONS_IDR_PEAK_FILE} with mean estimated fragment length of rep1 and rep2 Pooled tagAlign file ${POOLED_TA_FILE} vs. IDR peak from pooled pseudo replicates (Np) ${OPTIMAL_IDR_PEAK_FILE} with mean estimated fragment length of rep1 and rep2
Output(s)	FRiP text file ${FRiP}
Commands	# get estimated fragment length from cross-corr. analysis log FRAGLEN=$(cat ${CC_QC_LOG} | awk '{print $3}') HALF_FRAGLEN=$(((FRAGLEN+1)/2)) # rounding to integer CHRSIZEFILE=<path_of_file_containing_chromosome_sizes> # This file is a tab delimited file with 2 columns Col1 (chromosome name), Col2 (chromosome size in bp). val1=$(bedtools slop -i ${TA_FILE} -g $CHRSIZEFILE -s -l -$HALF_FRAGLEN -r $HALF_FRAGLEN | \ awk '{if ($2>=0 && $3>=0 && $2<=$3) print $0}' | \ bedtools intersect -a stdin -b ${IDR_PEAK_FILE} -wa -u \| wc -l) val2=$(zcat $TA_FILE \| wc -l) awk 'BEGIN {print '${val1}'/'${val2}'}' > ${FRIP}
QC to report	Fraction of reads in peaks

Status	Frozen

<TODO>

4g. Compute Peak Overlap Statistics between peak callers

<TODO>

5. Create signal tracks

NOT FINALIZED

5a. Make signal files - WIGGLER

- Code package:
  - https://sites.google.com/site/anshulkundaje/projects/wiggler
  - https://code.google.com/p/align2rawsignal/
- Dependencies:
  - MATLAB runtime library: http://www.broadinstitute.org/~anshul/softwareRepo/MCR2010b.bin
  - SAMtools
  - At least 2 GB of memory - the more, the faster it runs
  - Appropriate sex and length matched mappability files: http://www.broadinstitute.org/~anshul/projects/umap/
- Install MCR into a directory of choice: ./MCR2010b.bin -console
- Modify .bashrc:
  - MCRROOT=<MCR_ROOT>/v714
  - LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${MCRROOT}/runtime/glnxa64
  - LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${MCRROOT}/bin/glnxa64
  - LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${MCRROOT}/sys/os/glnxa64
  - MCRJRE=${MCRROOT}/sys/java/jre/glnxa64/jre/lib/amd64
  - LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${MCRJRE}/native_threads
  - LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${MCRJRE}/server
  - LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${MCRJRE}
  - XAPPLRESDIR=${MCRROOT}/X11/app-defaults
  - export LD_LIBRARY_PATH
  - export XAPPLRESDIR
- or .cshrc:
  - setenv MCRROOT <MCR_ROOT>/v714
  - setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${MCRROOT}/runtime/glnxa64
  - setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${MCRROOT}/bin/glnxa64
  - setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${MCRROOT}/sys/os/glnxa64
  - setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${MCRROOT}/sys/java/jre/glnxa64/jre/lib/amd64/native_threads
  - setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${MCRROOT}/sys/java/jre/glnxa64/jre/lib/amd64/server
  - setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${MCRROOT}/sys/java/jre/glnxa64/jre/lib/amd64
  - setenv XAPPLRESDIR ${MCRROOT}/X11/app-defaults

- Generates genome-wide normalized (but not control normalized for ChIP-seq) signal tracks in bedGraph and/or bigWig format

Program(s)	• WIGGLER
Input(s)	• tagAlign/BAM files • directory of all fasta sequences by chromosome (hg19 male/female) • mappability files from http://www.broadinstitute.org/~anshul/projects/umap/
Output(s)	• Wiggle or bedGraph signal files

Commands	• Usage: align2rawsignal -i=<alignFname> -s=<seqDir> -of=bg -l=<fragLen> -mm=<memory>
Parameters	• -i=<alignFname> (MANDATORY, MULTIPLE ALLOWED). One or more tagAlign/BAM files (replicates) as input. • -s=<seqDir> (MANDATORY). Full path to directory containing chromosome fasta files (eg. chr1.fa ..) The file names MUST match the chromosome names used in the tagAlign files • -u=<uMapDir> (MANDATORY). Full path to directory containing binary mappability tracks. The directory name must be of the form [PATH]/globalmap_k<min>tok<max> • -o=<oFname> (OPTIONAL). Full path and name of output signal file. • -of=<outputFormat> (OPTIONAL). Output signal file format wiggle (wig) or bedGraph (bg) or matfile (mat) • -l=<fragLen> (OPTIONAL, MULTIPLE ALLOWED). Fragment-length == 2*Tag-shift. Default: 1 (no extension)* • -w=<smoothingWindow> (OPTIONAL). Smoothing window size for signal. Default: mean(1.5*fragLen) • -k=<smoothingKernel> (OPTIONAL). Smoothing kernel to use. Valid kernels (rectangular,triangular,epanechnikov,biweight,triweight,cosine,gaussian,tukey) Default: tukey (with taper ratio of max(0.25 , min (0.5,max(w-mean(l),0)/(2*mean(l)))) • -mm=<memory> (OPTIONAL). Total memory to use in GB. Default: 2
QC to report	• None.

- *NOTE: Multiple arguments of this type are allowed. In such a case, number of fragLen arguments MUST BE == no. of Align files. The ORDER of these arguments is important.
- e.g. The first <fragLen> is matched with the first tagAlign/BAM file and so on.
- It might make sense to use the estimated fragment lengths reported in 2b here (to be verified with Anshul).
- Use bedGraphToBigWig from the Kent source tree to convert into bigWig format

6. Signal tracks

Peak calling and signal tracks using MACSv2 for TFs and histone marks

- Use the estimated fragment length from column 3 from ${CC_SCORES_FILE}

Program(s)	MACSv2 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
Input(s)	RepN ChIP ${REP1_TA_FILE} ${REP2_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled replicate ${POOLED_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz RepN pseudoreplicate1 ${REP*_PR1_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz RepN pseudoreplicate2 ${REP*_PR2_TA_FILE} vs. appropriate control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled-pseudoreplicate 1 ${PPR1_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz Pooled-pseudoreplicate 2 ${PPR2_TA_FILE} vs. pooled control tagAlign ${CONTROL_TA_PREFIX}.tagAlign.gz
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	GENOMESIZE='hs' # for human GENOMESIZE='mm' #for mouse NPEAKS=500000 # capping number of peaks called from MACS2 ========== # Generate narrow peaks and preliminary signal tracks ========== macs2 callpeak -t ${REP1_TA_FILE}.tagAlign.gz -c ${CONTROL_TA_PREFIX}.tagAlign.gz -f BED -n ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX} -g ${GENOMESIZE} -p 1e-2 --nomodel --shift 0 --extsize ${FRAGLEN} --keep-dup all -B --SPMR # Sort by Col8 in descending order and replace long peak names in Column 4 with Peak_<peakRank> sort -k 8gr,8gr ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.narrowPeak | awk 'BEGIN{OFS="\t"}{$4="Peak_"NR ; print $0}' | head -n ${NPEAKS} | gzip -nc > ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.narrowPeak.gz # remove additional files rm -f ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.xls ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.narrowPeak ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_summits.bed ~~==========~~ ~~# Generate Broad and Gapped Peaks~~ ~~==========~~

~~macs2 callpeak -t ${REP1_TA_FILE}.tagAlign.gz -c ${CONTROL_TA_PREFIX}.tagAlign.gz -f BED -n ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX} -g ${GENOMESIZE} -p 1e-2 --broad --nomodel --shift 0 --extsize ${FRAGLEN} --keep-dup all~~

~~# Sort by Col8 (for broadPeak) or Col 14(for gappedPeak) in descending order and replace long peak names in Column 4 with Peak_<peakRank>~~
~~sort -k 8gr,8gr ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.broadPeak | awk 'BEGIN{OFS="\t"}{$4="Peak_"NR ; print $0}' | gzip -nc > ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.broadPeak.gz~~

~~sort -k 14gr,14gr ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.gappedPeak | awk 'BEGIN{OFS="\t"}{$4="Peak_"NR ; print $0}' | gzip -nc > ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.gappedPeak.gz~~

~~# remove additional files~~
~~rm -f ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.xls ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.broadPeak ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_peaks.gappedPeak ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_summits.bed~~

~~==========~~
For Fold enrichment signal tracks
~~==========~~
CHRSIZEFILE=<path_of_file_containing_chromosome_sizes>
This file is a tab delimited file with 2 columns Col1 (chromosome name), Col2 (chromosome size in bp).

macs2 bdgcmp -t ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_treat_pileup.bdg -c ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_control_lambda.bdg --outdir ${PEAK_OUTPUT_DIR} -o ${CHIP_TA_PREFIX}_FE.bdg -m FE

Remove coordinates outside chromosome sizes (stupid MACS2 bug)
slopBed -i ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_FE.bdg -g ${CHRSIZEFILE} -b 0 | awk '{if ($3 != -1) print $0}' | bedClip stdin ${CHRSIZEFILE} ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.fc.signal.bedgraph

rm -f ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_FE.bdg

Convert bedgraph to bigwig
bedGraphToBigWig ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.fc.signal.bedgraph ${CHRSIZEFILE} ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.fc.signal.bw

rm -f ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.fc.signal.bedgraph

~~==========~~
For -log10(p-value) signal tracks
~~==========~~

Compute sval = min(no. of reads in ChIP, no. of reads in control) / 1,000,000
chipReads=$(zcat ${REP1_TA_FILE}.tagAlign.gz | wc -l | awk '{printf "%f", $1/1000000}');

controlReads=$(zcat ${CONTROL_TA_FILE}.tagAlign.gz | wc -l | awk '{printf "%f", $1/1000000}');

sval=$(echo "${chipReads} ${controlReads}" | awk '$1>$2{printf "%f",$2} $1<=$2{printf "%f",$1}');

	macs2 bdgcmp -t ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_treat_pileup.bdg -c ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_control_lambda.bdg --outdir ${PEAK_OUTPUT_DIR} -o ${CHIP_TA_PREFIX}_ppois.bdg -m ppois -S ${sval} # Remove coordinates outside chromosome sizes (stupid MACS2 bug) slopBed -i ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_ppois.bdg -g ${CHRSIZEFILE} -b 0 | awk '{if ($3 != -1) print $0}' | bedClip stdin ${CHRSIZEFILE} ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.pval.signal.bedgraph rm -rf ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_ppois.bdg # Convert bedgraph to bigwig bedGraphToBigWig ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.pval.signal.bedgraph ${CHRSIZEFILE} ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.pval.signal.bw rm -f ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}.pval.signal.bedgraph rm -f ${PEAK_OUTPUT_DIR}/${CHIP_TA_PREFIX}_treat_pileup.bdg ${peakFile}_control_lambda.bdg
QC to report	
Status	Beta

6. Peak calling for Histone Marks

Naive overlap thresholding for histone peak calls

NOTE: We haven't yet finalized an IDR protocol for histone marks. For now this is a simple overlap version that works reasonably well. IDR protocol for histone marks is in development

But here we do a similar analysis as IDR described in Section 4. Repeat the same procedure for the following set of combination of (Rep1, Rep2 and Pooled) to do reproducibility QC.

(Rep1, Rep2, Pooled_rep)
(Rep1-PR1, Rep1-PR2, Rep1)
(Rep2-PR1, Rep2-PR2, Rep2)
(PPR1,PPR2,Pooled_rep)

(I've just split the piped commands on separate lines for clarity)

```
# ======================
# 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 through blacklist
zcat PooledInRep1AndRep2.narrowPeak.gz PooledInPsRep1AndPsRep2.narrowPeak.gz | sort | uniq | awk
'BEGIN{OFS="\t"} {if ($5>1000) $5=1000; print $0}' \
| grep -P 'chr[\dXY]+[ \t]' > 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 > 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 > finalPeakList.gappedPeak.gz

Paired-End Steps

Paired-End ChIP-nexus parameters

- **Rscript scripts/preprocess_paired_fastq.r**
- **Default parameters:**
    - **-t/--trim: Pre-trim all reads by length before processing [default=0]**
    - **-k/--keep: Minimum number of bases post-processed to keep read [default=18]**
    - **-b/--barcode: Comma separated fixed barcode sequences that follow random barcode [default=CTGA]**
    - **-r/--randombarcode: Length of random barcode [default=5]**
    - **-c/--chunksize: Number of FASTQ reads to process at once (in thousands) [default=1000]**
    - **-p/--processors: Number of parallel cores to use [default=2]**

Program(s)	• R 3.2.5 • Bioconductor 3.2
Input(s)	• FASTQ Read 1 file ${FASTQ_FILE_1} • FASTQ Read 2 file ${FASTQ_FILE_2} • Fixed barcode sequences, comma delimited ${BARCODES} (required as CLI args) • Output file prefix ${FASTQ_OUTPUT_PREFIX}
Output(s)	• FASTQ Read 1 file ${FASTQ_FILE_1_OUTPUT} • FASTQ Read 2 file ${FASTQ_FILE_2_OUTPUT}
Commands	# == # Trim reads, filter based on presence of fixed barcode, assign random barcode to FASTQ read name, and output processed FASTQ file. # == #1. Run using R (paired and single end) Rscript scripts/preprocess_fastq.r -f #{FASTQ_FILE_1} -s #{FASTQ_FILE_2} -b ${BARCODES} -o #{FASTQ_OUTPUT_PREFIX}
QC to report	?Fraction of sequences trimmed
Status	