Accompanying scripts for "Comparison of single-cell whole-genome amplification strategies"

Nuria Estévez-Gómez, Tamara Prieto, Amy Guillaumet-Adkins, Holger Heyn, Sonia Prado-López & David Posada

Contents

1 Pre-processing *bulk* NGS data

1.1 Clipping adaptors and removing reads shorter than 70bp

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

cutadapt --minimum-length 70 \
  -a IndexedAdapter=AGATCGGAAGAGCACACGTCTGAACTCCAGTCACATCTCGTATGCCGTCTTCTGCTTG \
  -A UniversalAdapter=AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTAGATCTCGGTGGTCGCCGTATCATT \
  -o ${WORKDIR}/${SAMPLE}.trimmed_1.fastq.gz \
  -p ${WORKDIR}/${SAMPLE}.trimmed_2.fastq.gz \
  ${ORIDIR}/${SAMPLE}_1.fastq.gz ${ORIDIR}/${SAMPLE}_2.fastq.gz
```

1.2 Mapping reads

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
ID=${SAMPLE}
sample=$(echo $SAMPLELIST | sed 's/Files.//')
SM=${sample}
PL="ILLUMINA"
if [ $SAMPLELIST == "Files.Caco-2" ]
then
        LB="TruSeqDNAPCR-Free"
elif [ $SAMPLELIST == "Files.Z-138" ]
then
        LB="CNAGin-house"
elif [ $SAMPLELIST == "Files.HDF" ]
then
        LB="NxSeqAmpFREELowDNA"
fi
PU=`zcat ${ORIDIR}${SAMPLE}_1.fastq.gz | head -1 | sed 's/[:].*//' | sed 's/@//'`


RG="@RG\\tID:${ID}\\tSM:${SM}\\tPL:${PL}\\tLB:${LB}\\tPU:${PU}"

bwa mem -t 8 -M -R ${RG} ${RESDIR}/${REF}.fa \
          ${WORKDIR}/${SAMPLE}.trimmed_1.fastq.gz \
          ${WORKDIR}/${SAMPLE}.trimmed_2.fastq.gz > ${WORKDIR}/${SAMPLE}.bam
```

1.3 Sorting reads

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1


source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

java -jar $PICARD SortSam \
        I=${WORKDIR}/${SAMPLE}.bam \
        TMP_DIR=${WORKDIR} \
        O=${WORKDIR}/${SAMPLE}.sorted.bam \
        CREATE_INDEX=true \
        SORT_ORDER=coordinate
```

1.4 Marking duplicates and merging different read groups

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1

samples=$(awk -v dir=$WORKDIR '{print "I="dir$0".sorted.bam"}' ${ORIDIR}/${SAMPLELIST} | \
  tr '\n' ' ')
SAMPLE=$(echo $SAMPLELIST | sed 's/Files.//')

java -jar $PICARD MarkDuplicates \
        ${samples} \
        OUTPUT=${WORKDIR}/${SAMPLE}.dedup.bam \
        CREATE_INDEX=true \
        TMP_DIR=${WORKDIR} \
        M=${WORKDIR}/Duplicates_${SAMPLE}.txt \
        VALIDATION_STRINGENCY=LENIENT
```

1.5 Recalibrating base quality scores

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 8

source ReadConfig.sh $1
SAMPLE=$(echo $SAMPLELIST | sed 's/Files.//')
echo $SAMPLE
```

```
java -jar $GATK \
        -T BaseRecalibrator \
        -nct 8 \
        -R ${RESDIR}/${REF}.fa \
        -I ${WORKDIR}/${SAMPLE}.dedup.bam \
        -knownSites ${RESDIR}/dbsnp_138.b37.vcf \
        -knownSites ${RESDIR}/Mills_and_1000G_gold_standard.indels.b37.vcf \
        -o ${WORKDIR}/RecalibrationReportI_${SAMPLE}.grp
```

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1
SAMPLE=$(echo $SAMPLELIST | sed 's/Files.//')
echo $SAMPLE

java -jar $GATK \
        -T PrintReads \
        -I ${WORKDIR}/${SAMPLE}.dedup.bam \
        -BQSR ${WORKDIR}/RecalibrationReportI_${SAMPLE}.grp \
        -o ${WORKDIR}/${SAMPLE}.recal.bam \
        -R ${RESDIR}/${REF}.fa
```

2 Pre-processing *single-cell* NGS data

2.1 Clipping adaptors and removing reads shorter than 70bp

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

if [[ ${SAMPLE} == *"NX"* ]]
then
        Adapter1="CTGTCTCTTATACACATCTCCGAGCCCACGAGAC"
        Adapter2="CTGTCTCTTATACACATCTGACGCTGCCGACGA"
elif [[ ${SAMPLE} == *"SS"* ]]
then
        Adapter1="CTGTCTCTTGATCACA"
        Adapter2="CTGTCTCTTGATCACA"
elif [[ ${SAMPLE} == *"CW"* ]]
then
        Adapter1="AGATCGGAAGAGCACACGTCTGAACTCCAGTCACNNNNNNATCTCGTATGCCGTCTTCTGCTTG"
        Adapter2="AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTAGATCTCGGTGGTCGCCGTATCATT"
elif [[ ${SAMPLE} == *"NS"* ]]
then
        Adapter1="AGATCGGAAGAGCACACGTCTGAACTCCAGTCACNNNNNNATCTCGTATGCCGTCTTCTGCTTG"
        Adapter2="AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTAGATCTCGGTGGTCGCCGTATCATT"
else
        echo "CutAdapt not needed"
        exit
fi


cutadapt --minimum-length 70 -a AdapterA=$Adapter1 \
        -A AdapterB=$Adapter2 \
        -o ${WORKDIR}/${SAMPLE}.trimmed_1.fastq.gz \
        -p ${WORKDIR}/${SAMPLE}.trimmed_2.fastq.gz \
        ${ORIDIR}/${SAMPLE}_1.fastq.gz ${ORIDIR}/${SAMPLE}_2.fastq.gz
```

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1


source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${IDLIST})
echo $SAMPLE


if [[ ${WGA_LIBRARY} == "AMPLI-1" ]]
```

```
then
        Adapter1="GCTGTCAGTTAA"
        Adapter2="TTAACTGACAGCAGGAATCCCACT"
        adapters_to_remove="-g Adapter5=${Adapter1} -G Adapter5=${Adapter1} \
        -a Adapter3=${Adapter2} -A Adapter3=${Adapter2}"
elif [[ ${WGA_LIBRARY} == "MALBAC" ]]
then
        Adapter1="GTGAGTGATGGTTGAGGTAGTGTGGAG"
        Adapter2="CTCCACACTACCTCAACCATCACTCAC"
        adapters_to_remove="-g Adapter5=${Adapter1} -G Adapter5=${Adapter1} \
        -a Adapter3=${Adapter2} -A Adapter3=${Adapter2}"
elif [[ ${WGA_LIBRARY} == "PICOPLEX" ]]
then
        Adapter1="TGTGTTGGGTGTGTTTGG"
        Adapter2="CCAAACACACCCAACACA"
        Adapter3="TGTTGTGGGTTGTGTTGG"
        Adapter4="CCAACACAACCCACAACA"
        Adapter5="TGTGTTGGGTGTGTTTGG"
        Adapter6="CCAAACACACCCAACACA"
        adapters_to_remove="-g Adapter5=${Adapter1} -G Adapter5=${Adapter1} \
        -a Adapter3=${Adapter2} -A Adapter3=${Adapter2} \
        -g Adapter5.2=${Adapter3} -G Adapter5.2=${Adapter3} \
        -a Adapter3.2=${Adapter4} -A Adapter3.2=${Adapter4} \
        -g Adapter5.3=${Adapter5} -G Adapter5.3=${Adapter5} \
        -a Adapter3.3=${Adapter6} -A Adapter3.3=${Adapter6}"
else
        echo "CutAdapt not needed for ${WGA_LIBRARY}?"
        ln -s ${WORKDIR}/${SAMPLE}.trimmed_1.fastq.gz \
        ${WORKDIR}/${SAMPLE}.trimmed2_1.fastq.gz
        ln -s ${WORKDIR}/${SAMPLE}.trimmed_2.fastq.gz \
        ${WORKDIR}/${SAMPLE}.trimmed2_2.fastq.gz
        exit
fi


cutadapt --minimum-length 70 ${adapters_to_remove} \
        -o ${WORKDIR}/${SAMPLE}.trimmed2_1.fastq.gz \
        -p ${WORKDIR}/${SAMPLE}.trimmed2_2.fastq.gz \
        ${WORKDIR}/${SAMPLE}.trimmed_1.fastq.gz \
        ${WORKDIR}/${SAMPLE}.trimmed_2.fastq.gz > ${WORKDIR}/${SAMPLE}_CutadaptWGA.txt
```

2.2 Mapping reads

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1


source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE
```

```
ID=${SAMPLE}

SM=${SAMPLE}

if [[ ${SAMPLE} == *"NX"* ]]
then
        LB="Nextera"
elif [[ ${SAMPLE} == *"SS"* ]]
then
        LB="SureSelectQXT"
elif [[ ${SAMPLE} == *"CW"* ]]
then
        LB="CNAGin-house"
elif [[ ${SAMPLE} == *"NS"* ]]
then
        LB="NxSeqAmpFREELowDNA"
else
        echo "not working"
        exit
fi


PL="ILLUMINA"
PU=`zcat ${ORIDIR}/${SAMPLE}_1.fastq.gz | head -1 | sed 's/[:].*//' | sed 's/@//'`
echo "@RG\tID:${ID}\tSM:${SM}\tPL:${PL}\tLB:${LB}\tPU:${PU}" > ${WORKDIR}/RG_${SAMPLE}
```

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 4


source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

read -r RG < ${WORKDIR}/RG_${SAMPLE}
bwa mem -t 4 -M -R ${RG} ${RESDIR}/${REF}.fa \
  ${WORKDIR}/${SAMPLE}.trimmed_1.fastq.gz \
  ${WORKDIR}/${SAMPLE}.trimmed_2.fastq.gz > ${WORKDIR}/${SAMPLE}.bam
```

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -t 02:00:00


source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

SM=${SAMPLE}
LB="IonPlus"
```

```
PU=$(grep -m 1 "^@" ${WORKDIR}/${SAMPLE}.fastq | sed 's/:.*//' | sed 's/@//')

tmap mapall \
 --fn-fasta ${RESDIR}/${REF}.fa \
 --fn-reads ${WORKDIR}/${SAMPLE}.fastq \
 --num-threads 4 \
 -R "ID:"${SAMPLE} -R "PL:IONTORRENT"   -R "SM:"${SAMPLE} -R "PU:"${PU} -R "LB:"${LB} \
 --reads-format fastq \
 --output-type 2 \
 --fn-sam ${WORKDIR}/${SAMPLE}.bam \
 -v stage1 map4 -Y -u
```

2.3 Sorting reads

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1


source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

java -jar $PICARD SortSam \
        I=${WORKDIR}/${SAMPLE}.bam \
        TMP_DIR=${WORKDIR} \
        O=${WORKDIR}/${SAMPLE}.sorted.bam \
        CREATE_INDEX=true \
        SORT_ORDER=coordinate
```

2.4 Marking duplicates

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1


source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

java -jar $PICARD MarkDuplicates \
        INPUT=${WORKDIR}/${SAMPLE}.sorted.bam \
        OUTPUT=${WORKDIR}/${SAMPLE}.dedup.bam \
        CREATE_INDEX=true \
        TMP_DIR=${WORKDIR} \
        M=${WORKDIR}/Duplicates_${SAMPLE}.txt \
        VALIDATION_STRINGENCY=LENIENT
```

2.5 Recalibrating base quality scores

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 8


source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

java -jar $GATK \
        -T BaseRecalibrator \
        -nct 8 \
        -R ${RESDIR}/${REF}.fa \
        -I ${WORKDIR}/${SAMPLE}.dedup.bam \
        -knownSites ${RESDIR}/dbsnp_138.b37.vcf \
        -knownSites ${RESDIR}/Mills_and_1000G_gold_standard.indels.b37.vcf \
        -o ${WORKDIR}/RecalibrationReportI_${SAMPLE}.grp
```

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

java -jar $GATK \
        -T PrintReads \
        -I ${WORKDIR}/${SAMPLE}.dedup.bam \
        -BQSR ${WORKDIR}/RecalibrationReportI_${SAMPLE}.grp \
        -o ${WORKDIR}/${SAMPLE}.recal.bam \
        -R ${RESDIR}/${REF}.fa
```

3 Predicting single-cell breadths at higher sequencing depths

3.1 Obtaining sequencing depth

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1


source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

if [[ $SAMPLE = *"IP"* ]]
then
        raw_reads=$(grep "TOTAL_READS" \
        ${WORKDIR}/${SAMPLE}.alignment_summary_metrics.txt | awk '{print $3}')
        aligned_soft_bases=$(samtools view $WORKDIR/$SAMPLE.dedup.bam \
        | cut -f10 | awk '{total+=length}END{print total}')
        hard_bases=$(samtools view $WORKDIR/$SAMPLE.dedup.bam | cut -f6 \
        | grep H | sed 's/\([A-Z]\)[0-9]/\1\n/g' | grep H | sed 's/H//' | \
        awk '{sum+=$1}END{print sum}')
        if [ -z "$hard_bases" ];then
                hard_bases=0
        fi
        raw_bases=$(($aligned_soft_bases + $hard_bases))
        genome_length=$(cat ${RESDIR}/${REF}.fai | cut -f2 | \
        awk '{sum+=$1}END{print sum}')
        sequencing_depth=$(awk -v genomelength=$genome_length \
        -v rawbases=$raw_bases 'BEGIN{print rawbases/genomelength}')
else
        raw_reads=$(zcat ${ORIDIR}/${SAMPLE}_1.fastq.gz | awk 'END{print NR/4}')
        genome_length=$(cat ${RESDIR}/${REF}.fai | cut -f2 | \
        awk '{sum+=$1}END{print sum}')
        sequencing_depth=$(zcat ${ORIDIR}/${SAMPLE}_1.fastq.gz | head -2 \
        | tail -n 1 | wc -c | awk -v rawreads=$raw_reads \
        -v genomelength=$genome_length '{print (($1 - 1)*(rawreads*2))/genomelength}')
fi
echo $SAMPLE $sequencing_depth > ${WORKDIR}/${SAMPLE}.seqdepth.txt
```

3.2 Downsampling to the lowest single-cell sequencing depth

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE
```

```
config=$(basename $1 | sed 's/.txt//' | sed 's/Config.//' | sed 's/.SC//')

sequencing_depth=$(awk '{print $2}' ${WORKDIR}/${SAMPLE}.seqdepth.txt)
echo $SAMPLE $sequencing_depth

subsampling_depth=$(awk '{print $2}' ${WORKDIR}/${config}.seqdepths.txt  \
| sort -n | head -n 1)
## Calculating subsampling probability
probability=`bc -l <<< "scale=3; $subsampling_depth / $sequencing_depth"`
echo "Downsampling probability: "$probability

strategy="Chained"
echo "Downsampling following "${strategy}" strategy. \
From "${sequencing_depth}"X to "${subsampling_depth}"X."
java -jar $PICARD DownsampleSam \
        INPUT=${WORKDIR}/${SAMPLE}.dedup.bam \
        OUTPUT=${WORKDIR}/${SAMPLE}.dedup.ds.bam \
        RANDOM_SEED=1 \
        PROBABILITY=${probability} \
        STRATEGY=$strategy
```

3.3 Running Preseq gc_extrap

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 4

source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

bam2mr -o ${WORKDIR}/${SAMPLE}.mr ${WORKDIR}/${SAMPLE}.sorted.ds.bam
preseq gc_extrap -v -o ${WORKDIR}/${SAMPLE}_inferredcov.txt ${WORKDIR}/${SAMPLE}.mr
```

4 Assessing coverage uniformity

4.1 Computing base depths along the genome

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

bedtools genomecov \
        -ibam ${WORKDIR}/${SAMPLE}.dedup.ps.bam | \
        grep "^genome" > ${WORKDIR}/${SAMPLE}_genome.ps.bed
```

4.2 Reconstructing lorenz curves and calculating gini index

```
library(ineq)
library(data.table)
library(ggplot2)
library(reshape)
library(ggpubr)
library(RColorBrewer)

Lorenz.Gini <-function(cell_line) {
genomebed<-fread(paste("LorenzCurvesIneq_",cell_line,".txt",sep=""),
                 colClasses = c("factor","numeric","numeric"),
                 col.names = c("Sample","reads", "times"))

# Plot the lorez curves
lcpersample<-function(sample) {
  genomebed<-genomebed[which(genomebed$Sample==sample),]
  lc<-Lc(genomebed$reads,genomebed$times/3137454505)
  sample<-rep(sample,length(lc$p))
  scWGAkit<-sub("GP","GenomiPhi",sub("AM","Ampli1",
            sub("TP","TruePrime",sub("PP","PicoPLEX",
            sub("MB","MALBAC",sub("RG","REPLIg",
            sub("-.*$","", sample)))))))
  mydata<-cbind.data.frame(sample,scWGAkit,lc$p, lc$L)
  colnames(mydata) <- c("Sample","scWGA kit","x","y")
  return(mydata)
}
lcs<-lapply(levels(as.factor(genomebed$Sample)), lcpersample)
# cbind all data.frames in one list
mydata<-merge_recurse(lcs)
library(RColorBrewer)
cols<- brewer.pal(8, name="Dark2")[c(1:8)]
mycols<-rep(cols,each=table(sapply(lcs,function(x) unique(x[,2]))))
lorenz<-ggplot(data=mydata) +
  geom_line(aes(x,y,col=`Sample`), alpha=0.5) +
  scale_color_manual(values = mycols) +
  scale_linetype_manual(values = c("dashed")) +
  coord_cartesian(xlim = c(0.55,1)) +
  labs(x="Cumulative fraction of reads",
       y="Cumulative fraction of \ngenome coverage breadth",
       title=cell_line) +
  theme(text=element_text(size=8, family="Arial"),
        axis.title=element_text (face="bold"),
        axis.line = element_line(colour = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_rect(fill = "white"),
        plot.title = element_text(hjust = 0.5),
        legend.position = "none")



# Calculate gini index and compare
gini<-function(sample) {
  genomebed<-genomebed[which(genomebed$Sample==sample),]
  x<-sample(x=genomebed$reads, size = 1000000,
            prob = genomebed$times, replace=TRUE)
  gini<-Gini(x, corr=TRUE)
  names(gini) <- sample
  return(gini)
}
ginis<-as.data.frame(unlist(lapply(levels(as.factor(genomebed$Sample)), gini)))
ginis$Samples<-rownames(ginis)
rownames(ginis) <- NULL
colnames(ginis) <- c("gini","Sample")
scWGAkit<-sub("GP","GenomiPhi",sub("AM","Ampli1",
          sub("TP","TruePrime",sub("PP","PicoPLEX",
          sub("MB","MALBAC",sub("RG","REPLIg",
          sub("-.*$","", ginis$Sample)))))))
gini<-cbind.data.frame(scWGAkit,ginis)
ginis<-ggplot(data=ginis) +
  geom_boxplot(aes(x=scWGAkit, y=gini, color=scWGAkit), outlier.shape = NA) +
  geom_jitter(aes(x=scWGAkit, y=gini, color=scWGAkit)) +
  scale_color_brewer(palette = "Dark2") +
  labs(x="scWGA kit", y="Gini index") +
  theme(text=element_text(size=8, family="Arial"),
        axis.title=element_text (face="bold"),
        axis.line = element_line(colour = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_rect(fill = "white"),
        plot.title = element_text(hjust = 0.5),
        legend.position = "none")

grid.arrange(lorenz, ginis, nrow = 1)
}
```

5 Testing for amplification recurrence

5.1 Filtering reads and obtaining sequencing depths

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1


source ReadConfig.sh $1
```

```
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE


samtools view -F 1284 -b ${WORKDIR}/${SAMPLE}.dedup.bam > ${WORKDIR}/${SAMPLE}.flt.cov.bam
samtools index ${WORKDIR}/${SAMPLE}.flt.cov.bam
raw_reads=$(samtools flagstat ${WORKDIR}/${SAMPLE}.flt.cov.bam | grep "in total" \
| awk '{print $1}')
genome_length=$(cat ${RESDIR}/${REF}.fai | cut -f2 | awk '{sum+=$1}END{print sum}')
sequencing_depth=$(awk -v rawreads=$raw_reads -v genomelength=$genome_length \
'BEGIN{print 150*rawreads/genomelength}')
echo $SAMPLE $sequencing_depth > ${WORKDIR}/${SAMPLE}.cor.seqdepth.txt
```

5.2 Obtaining read counts along 1Mb genome windows

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1


source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})

config=$(basename $1 | sed 's/.txt//' | sed 's/Config.//' | sed 's/.SC//')
sequencing_depth=$(awk '{print $2}' ${WORKDIR}/${SAMPLE}.cor.seqdepth.txt)
echo $sequencing_depth

subsampling_depth=0.1
probability=`bc -l <<< "scale=3; $subsampling_depth / $sequencing_depth"`
strategy="HighAccuracy"

echo "Downsampling following "${strategy}" strategy. \
From "${sequencing_depth}"X to "${subsampling_depth}"X."
java -jar $PICARD DownsampleSam \
        INPUT=${WORKDIR}/${SAMPLE}.flt.cov.bam \
        OUTPUT=${WORKDIR}/${SAMPLE}.cor.bam \
        RANDOM_SEED=1 \
        PROBABILITY=${probability} \
        STRATEGY=$strategy \
        CREATE_INDEX=true

python /home/uvi/be/tpf/apps/pysamstats/scripts/pysamstats \
        -t coverage_binned \
        ${WORKDIR}/${SAMPLE}.cor.bam \
        --max-depth=1000000000 \
        --fasta ${RESDIR}/${REF}.fa \
        --window-size=1000000 \
        --omit-header | cut -f4 > ${WORKDIR}/${SAMPLE}.1Mb.cov.txt
```

*Strategy followed for bulk downsampling was "Chained" instead of "HighAcurracy"

5.3 Obtaining gc content along 1Mb genome windows

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1


source ReadConfig.sh $1
cat ${RESDIR}/${REF}.fai | cut -f1-2 > ${WORKDIR}/chrom.sizes
bedtools makewindows -g ${WORKDIR}/chrom.sizes -w 1000000 \
> ${WORKDIR}/${REF}.1Mbwindows.bed
bedtools nuc -fi ${RESDIR}/${REF}.fa -bed ${WORKDIR}/${REF}.1Mbwindows.bed \
| cut -f4,5,10,12 | tail -n +2 \
| awk '{print $1"\t"$2"\t"$3/$4}' > ${WORKDIR}/GC.1Mbwindows.bed
```

5.4 Testing for amplification correlation using 1Mb window read counts

```
library(data.table)
library(Hmisc)
library(RColorBrewer)
library(corrplot)


gc_content<-fread("/Users/tama/Downloads/GC.1Mbwindows.bed", \
                  col.names = c("AT","GC"), colClasses = c("numeric"), na.strings = "NA")
x <- fread("HDF.1Mb.June.Bulk.txt", col.names = c("Ampli1-10","Ampli1-2",
                                                  "Ampli1-5","Ampli1-9",
                                                  "GenomiPhi-1","GenomiPhi-6",
                                                  "GenomiPhi-7","GenomiPhi-9",
                                                  "Bulk","MALBAC-10",
                                                  "MALBAC-3","MALBAC-4",
                                                  "MALBAC-8","PicoPLEX-2",
                                                  "PicoPLEX-3","PicoPLEX-4",
                                                  "PicoPLEX-5","REPLIg-10",
                                                  "REPLIg-13","REPLIg-1",
                                                  "REPLIg-2","TruePrime-10",
                                                  "TruePrime-4","TruePrime-7",
                                                  "TruePrime-9"))
x <-x[,c(1:4,10:17,5:8,18:25,9)]
corrgc<-rcorr(x=as.matrix(x), y=as.matrix(gc_content[,-1]), type="pearson")
pvalgc<-corrgc$P
corgc<-corrgc$r
colnames(pvalgc)[26] <- "GC content"
colnames(corgc)[26] <- "GC content"
rownames(pvalgc)[26] <- "GC content"
rownames(corgc)[26] <- "GC content"

corrplot(as.matrix(corgc), type = "lower",
         sig.level = 0.05,
         col=brewer.pal(10,'RdYlBu'),
         method="circle", tl.col = "black",
         p.mat=as.matrix(pvalgc),
         insig = "blank",
         addgrid.col=NA,
         number.digits=NULL,
         diag=FALSE)
```

5.5 Creating coverage presence/absence matrix

```
#!/bin/sh

while read sample
do
  bedtools genomecov -ibam ${WORKDIR}/${SAMPLE}.cor.bam \
  -bg > ${WORKDIR}/${SAMPLE}.cor.bedGraph
  awk '{if ($4>0){print $1"\t"$2"\t"$3"\t"1}else {print $0}}' \
  $sample.cor.bedGraph > $sample.cor2.bedGraph
done < $GLUSTER/Single-Cell/SC_HDF.txt

# union all the bedgraph files
bedtools unionbedg -i *cor2.bedGraph | cut -f4- \
| tr -t "\t" "," > Intersection.genome.bedGraph

# remove singletons from the begraph file
bedtools unionbedg -i *cor2.bedGraph | cut -f1,4- \
| uniq -c | cut -f2- > Intersection.genome.notrep.bedGraph
```

5.6 Testing for amplification recurrence using Jaccard similarity coefficients (with permutations)

```
library(data.table)


# Read data
PresAbs <- fread("Intersection.genome.notrep.bedGraph",
                 col.names = c("Ampli1-10","Ampli1-2",
                               "Ampli1-5","Ampli1-9",
                               "GenomiPhi-1","GenomiPhi-6",
                               "GenomiPhi-7","GenomiPhi-9",
                               "MALBAC-10","MALBAC-3",
                               "MALBAC-4","MALBAC-8",
                               "PicoPLEX-2","PicoPLEX-3",
                               "PicoPLEX-4","PicoPLEX-5",
                               "REPLIg-10","REPLIg-13",
                               "REPLIg-1","REPLIg-2",
                               "TruePrime-10","TruePrime-4",
                               "TruePrime-7","TruePrime-9"))

PresAbs <-PresAbs[,c(1:4,9:16,5:8,17:24)]
```

```r
library(jaccard)
PresAbs<-as.data.frame(PresAbs)
jaccard_estimates <- apply(
  expand.grid(colnames(PresAbs),colnames(PresAbs)),
  1, FUN = function(x) {
  dat<-as.data.frame(PresAbs[,c(x[1],x[2])])
  names(dat) <- c("X","Y")
  jaccard(dat$X, dat$Y)
})

matrix_jaccard_estimates <- matrix(jaccard_estimates, ncol=length(colnames(PresAbs)))
colnames(matrix_jaccard_estimates) <- colnames(PresAbs)
rownames(matrix_jaccard_estimates) <- colnames(PresAbs)

write.table(as.data.frame(matrix_jaccard_estimates),"JaccardValues.txt", quote=FALSE)


# Get pvalues for jaccard through permutations

Combinations_required <-as.data.frame(matrix(combn(colnames(PresAbs), 2),
                                             nrow=276, byrow=T))
jaccard_pvalues_permutations <- apply(Combinations_required,
                                      1, FUN = function(x) {
    dat<-as.data.frame(PresAbs[,c(x[1],x[2])])
    names(dat) <- c("X","Y")
    total <- length(dat$X)
    times_0_1 <- table(dat$X)
    times_0 <- times_0_1[1]
    times_1 <- times_0_1[2]
    prob_0 <- times_0/total
    prob_1 <- times_1/total
    permut<-function (x) {
    jaccard(resampled <- sample(c(0,1), size = length(dat$X),
                                replace = TRUE, prob = c(prob_0,prob_1)), dat$Y)
    }
    resampled_jaccard<-sapply(1:1000,permut)
    observed<-jaccard(dat$X, dat$Y)
    # Test whether the observed value is higher
    # or equal than all of the numbers estimated randomly
    sum(resampled_jaccard >= observed)/length(resampled_jaccard) # alternative
})

out2 <- jaccard_pvalues_permutations
class(out2) <- "dist"
attr(out2, "Labels") <- as.character(colnames(PresAbs))
attr(out2, "Size") <- length(colnames(PresAbs))
attr(out2, "Upper") <- TRUE
attr(out2, "Diag") <- TRUE
out2 <- as.matrix(out2)
diag(out2) <- NA


write.table(as.data.frame(out2),"JaccardPValues.txt", quote=FALSE)
```

```r
# Plot Jaccard and remove non significant values
library(corrplot)
library(RColorBrewer)
png(height=1200, width=1200, pointsize=25, file="CorrMatrixJaccardPermutations.png")

col1=brewer.pal(9,'YlOrRd')[2]
col2=brewer.pal(8,'Set2')[5]
col3=brewer.pal(9,'BuGn')[5]
col4=brewer.pal(9,'Blues')[5]
col5=brewer.pal(9,'BuPu')[6]
col6 <- colorRampPalette(c("white","white","white","white","white",
                           col1,col2,col3,col4,col5))

corrplot(matrix_jaccard_estimates, type = "full",
    sig.level = 0.05,
    col=col6(20),
    method="circle",
    number.cex = .5,
    tl.col = "black",
    p.mat=out2,
    insig = "blank",
    is.corr = FALSE,
    number.digits=4,
    diag=FALSE)
dev.off()
```

6 Quantifying allelic imbalance (AI), allelic dropout (ADO) and false SNVs.

Realigning single-cell and bulk reads all together around known indels.

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1

samples=$(awk -v dir=$WORKDIR '{print "-I "dir$0".recal.bam"}' \
${ORIDIR}/${SAMPLELIST} | tr '\n' ' ')
cell_line=HDF

java -jar $GATK \
  -T RealignerTargetCreator \
  ${samples} \
  ${cell_line} \
  -R ${RESDIR}/${REF}.fa \
  -known ${RESDIR}/Mills_and_1000G_gold_standard.indels.b37.vcf \
  -known ${RESDIR}/1000G_phase1.indels.b37.vcf \
  -o ${WORKDIR}/${cell_line}.intervals
```

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1

samples=$(awk -v dir=$WORKDIR '{print "-I "dir$0".recal.bam"}' \
${ORIDIR}/${SAMPLELIST} | tr '\n' ' ')
cell_line=HDF

cd ${WORKDIR}
awk -v cl=$cell_line \
'BEGIN{print cl".recal.bam\t"cl".real.bam"}{print $0".recal.bam\t"$0".real.bam"}'\
${ORIDIR}/${SAMPLELIST} > ${WORKDIR}/${SLURM_JOBID}.map

java -jar $GATK \
        -T IndelRealigner \
        -I ${WORKDIR}/${cell_line}.recal.bam \
        $samples \
        -known ${RESDIR}/Mills_and_1000G_gold_standard.indels.b37.vcf \
        -known ${RESDIR}/1000G_phase1.indels.b37.vcf \
        -targetIntervals ${WORKDIR}/${cell_line}.intervals \
        -R ${RESDIR}${REF}.fa \
        --nWayOut ${WORKDIR}/${SLURM_JOBID}.map

rm ${WORKDIR}/${SLURM_JOBID}.map
```

6.1 Quantifying AI

6.1.1 Calling heterozygous variants in PCR-free amplified bulk

```sh
#!/bin/sh

source ReadConfig.sh $1
SAMPLE=$(echo $SAMPLELIST | sed 's/.txt//' | sed 's/SC_//' | sed 's/Files.//')
java -jar $GATK \
       -T HaplotypeCaller \
       -nct 8 \
       --pcr_indel_model NONE \
       -R ${RESDIR}/${REF}.fa \
       -I ${WORKDIR}/${SAMPLE}.real.bam \
       --dbsnp ${RESDIR}/dbsnp_138.b37.vcf \
       -o ${WORKDIR}/HaplotypeCaller.${SAMPLE}.var_sites.vcf

java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/HaplotypeCaller.${SAMPLE}.var_sites.vcf \
        -o ${WORKDIR}/HaplotypeCaller.${SAMPLE}.het_sites.vcf \
        -selectType SNP \
        -select 'vc.getGenotype("HDF").isHet()'
```

6.1.2 Performing variant quality score recalibration

```sh
#!/bin/sh

source ReadConfig.sh $GLUSTER/Single-Cell/Config.HDF.SC
SAMPLE="HDF"
cd $GLUSTER/Single-Cell/RESULTS/

gatk VariantRecalibrator -R ${RESDIR}/${REF}.fa \
   --variant ${WORKDIR}/HaplotypeCaller.${SAMPLE}.het_sites.vcf \
   --resource \
   hapmap,known=false,training=true,truth=true,prior=15.0:hapmap_3.3.b37.vcf \
   --resource \
   omni,known=false,training=true,truth=false,prior=12.0:1000G_omni2.5.b37.vcf \
   --resource \
   1000G,known=false,training=true,truth=false,prior=10.0:\
   1000G_phase1.snps.high_confidence.b37.vcf \
   --resource \
   dbsnp,known=true,training=false,truth=false,prior=2.0:dbsnp_138.b37.vcf \
   --use-annotation QD \
   -an MQ \
   -an MQRankSum \
   -an ReadPosRankSum \
   -an FS \
   -an SOR \
   -an DP \
   -mode SNP \
   --output output.recal \
   --tranches-file output.tranches \
   --rscript-file output.plots.R

gatk ApplyVQSR \
   -R ${RESDIR}/${REF}.fa \
   -V ${WORKDIR}/HaplotypeCaller.${SAMPLE}.het_sites.vcf \
   -O ${WORKDIR}/HaplotypeCaller.${SAMPLE}.het_sites.VQSR.vcf \
   -ts-filter-level 99.0 \
   --tranches-file output.tranches \
   --recal-file output.recal \
   -mode SNP

grep -e "^#" -e "PASS" ${WORKDIR}/HaplotypeCaller.${SAMPLE}.het_sites.VQSR.vcf > \
${WORKDIR}/HaplotypeCaller.${SAMPLE}.het_sites.VQSR.PASS.vcf
```

6.1.3 Running mpileup on the bulk and the single-cells

```
#!/bin/sh

samtools mpileup \
  -C50 -Osf ${RESDIR}/${REF}.fa \
  ${WORKDIR}/${SAMPLE}.real.bam > \
  ${WORKDIR}/SC.${SAMPLE}.mpileup
```

6.1.4 Indexing VCF files

```
#!/bin/sh

bgzip -c ${WORKDIR}/HaplotypeCaller.${SAMPLE}.het_sites.VQSR.PASS.vcf > \
${WORKDIR}/HaplotypeCaller.${SAMPLE}.het_sites.VQSR.PASS.vcf.gz
tabix -p vcf ${WORKDIR}/HaplotypeCaller.${SAMPLE}.het_sites.VQSR.PASS.vcf.gz
gatk IndexFeatureFile \
     -F ${WORKDIR}/HaplotypeCaller.${SAMPLE}.het_sites.VQSR.PASS.vcf.gz
```

6.1.5 Obtaining alternative allele fractions

```
#!/usr/bin/env python3

import re
import sys
import tabix

if sys.version_info[0] < 3:
    raise "Please, consider upgrading your version to 3.x.x in order to run this script"

if len(sys.argv) < 2:
    sys.exit('Please, provide a sample name as argument')
```

```
print ("\nSetting up variables\n")
workdir="/mnt/gluster/distributed/home/uvi/be/tpf/Single-Cell/RESULTS/"
sample=sys.argv[1]
prefix="SC."
suffix=".mpileup"
min_depth=6


mismatches=['A','C','G','T','N','a','c','t','g','n']

# pytabix command to load vcf file previously bgzipped and tabixed
tb = tabix.open(workdir+"HaplotypeCaller.HDF.het_sites.VQSR.PASS.vcf.gz")


mpileup_name= workdir + prefix + sample + suffix
print("Loading data frame ",mpileup_name,"\n")
out = open(workdir+"AllelicFrequency."+sample+".txt","w")
with open(mpileup_name, mode='r') as f: # open the pileup file
    for line in f:
        cols=re.split(r'\t+', line)
        pos_mpileup=str(cols[0])+':'+str(cols[1])
        if (int(cols[3]) >= min_depth): # Pass positions without reads
            try:
                records = tb.querys(str(cols[0])+ ':' + str(cols[1]) + '-' + str(cols[1]))
            except tabix.TabixError:
                print("Ignoring:",cols[0],cols[1],"TabixError")
            else:
                for record in records:
                    ref_vcf=record[3]
                    alt_vcf=record[4]
                    try:
                        alt_vcf #check the variable was defined
                        ref_vcf
                    except NameError:
                        continue # continue to next for iteration in for line
                    else: # else from the try when it is not except
                        if (len(alt_vcf)>1):
                            #ref_mpileup=cols[2]
                            list_characters=list(cols[4])
                            list_bq=list(cols[5])
                            print(alt_vcf)
                            if "," not in alt_vcf:
                                print('Indel detected. Position will not be considered\n')
                                continue
                            alleles=alt_vcf.split(',')
                            ref_vcf=alleles[0]
                            alt_vcf=alleles[1]
                            pos=0
                            index=0
                            ref_count=0
                            alt_count=0
                            while pos < len(list_characters):
                                if list_characters[pos]==".":
                                    pos += 1
                                    index+=1
                                elif list_characters[pos]==",":
                                    pos+=1
                                    index+=1
                                elif (list_characters[pos]=="^"):
                                    pos+=2
                                elif (list_characters[pos]=="-") \
                                | (list_characters[pos]=="+"):
                                    substring_list=list_characters[(pos+1):]
                                    substring=''.join(substring_list)
                                    num=re.search(r'(\d+)(\w+)', substring).groups()[0]
                                    pos+=1+len(str(num))+int(num)
                                elif list_characters[pos] in mismatches:
                                    base_mpileup=list_characters[pos]
                                    base_mpileup_upper=base_mpileup.upper()
                                    if (base_mpileup_upper==alt_vcf):
                                        alt_count+=1
                                    elif (base_mpileup_upper==ref_vcf):
                                        ref_count+=1
                                    pos+=1
                                    index+=1
                                else:
                                    pos += 1
                                    continue
                            total_count=alt_count+ref_count
                            if (total_count > 0):
                                allele_fraction=alt_count/total_count
                                out.write(sample+"\tchr"+pos_mpileup+\
                                "\t"+str(allele_fraction)+"\n")
                            continue
                        elif (len(alt_vcf)==1):
                            list_characters=list(cols[4])
                            list_bq=list(cols[5])
                            pos=0
                            index=0
                            ref_count=0
                            alt_count=0
                            while pos < len(list_characters):
                                if list_characters[pos]==".":
                                    pos += 1
                                    index+=1
                                    ref_count+=1
                                elif list_characters[pos]==",":
                                    pos+=1
                                    index+=1
                                    ref_count+=1
                                elif (list_characters[pos]=="^"):
                                    pos+=2
                                elif (list_characters[pos]=="-") | \
                                (list_characters[pos]=="+"):
                                    substring_list=list_characters[(pos+1):]
                                    substring=''.join(substring_list)
                                    num=re.search(r'(\d+)(\w+)', substring).groups()[0]
                                    pos+=1+len(str(num))+int(num)
                                elif list_characters[pos] in mismatches:
                                    alt_mpileup=list_characters[pos]
                                    alt_mpileup_upper=alt_mpileup.upper()
                                    if (alt_mpileup_upper==alt_vcf):
                                        alt_count+=1
                                    pos+=1
                                    index+=1
                                else:
                                    pos += 1
                                    continue
                            total_count=alt_count+ref_count
                            if (total_count > 0):
                                allele_fraction=alt_count/total_count
                                out.write(sample+"\tchr"+pos_mpileup+\
                                "\t"+str(allele_fraction)+"\n")
                        else:
                            continue

f.close()
```

6.2 Quantifying false SNVs and ADO

6.2.1 Running HaplotypeCaller on ERC mode for each single-cell

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1


source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})

java -jar $GATK \
        -T HaplotypeCaller \
        -nct 1 \
        --emitRefConfidence GVCF \
        --pcr_indel_model CONSERVATIVE \
        -R ${RESDIR}/${REF}.fa \
        -I ${WORKDIR}/${SAMPLE}.real.bam \
        --dbsnp ${RESDIR}/dbsnp_138.b37.vcf \
        -o ${WORKDIR}/HaplotypeCaller.${SAMPLE}.g.vcf
```

6.2.2 Runing HaplotypeCaller on ERC mode for the bulk. Set –pcr_indel_model to NONE as the sequencing library is PCR free.

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 8


source ReadConfig.sh $1
SAMPLE=$(echo $SAMPLELIST | sed 's/.txt//' | sed 's/SC_//' | sed 's/Files.//')

## IMPORTANT: when using PCR-free sequencing data we definitely
## recommend setting this argument to NONE
## HDF library: NxSeq AmpFREE Low DNA

java -jar $GATK \
        -T HaplotypeCaller \
        -nct 8 \
        --emitRefConfidence GVCF \
        --pcr_indel_model NONE \
        -R ${RESDIR}/${REF}.fa \
        -I ${WORKDIR}/${SAMPLE}.real.bam \
        --dbsnp ${RESDIR}/dbsnp_138.b37.vcf \
        -o ${WORKDIR}/HaplotypeCaller.${SAMPLE}.g.vcf
```

6.2.3 Running GenotypeGVCFs to perform a *joint calling* with the bulk and the 24 single-cells.

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1


source ReadConfig.sh $1
cell_line=$(echo $SAMPLELIST | sed 's/.txt//' | sed 's/SC_//' | sed 's/Files.//')
samples=$(awk -v dir=$WORKDIR '{print "--variant "dir"/HaplotypeCaller."$0".g.vcf"}' \
${ORIDIR}/${SAMPLELIST} | tr '\n' ' ')

java -jar $GATK \
        -T GenotypeGVCFs \
        -R ${RESDIR}/${REF}.fa \
        --variant ${WORKDIR}/HaplotypeCaller.${cell_line}.g.vcf \
        ${samples} \
        --dbsnp ${RESDIR}/dbsnp_138.b37.vcf \
        -newQual \
        -o ${WORKDIR}/GenotypedGVCFs.${cell_line}.Joint.vcf
```

6.2.4 Running GenotypeGVFs to perform a marginal calling.

```
#!/bin/sh
#SBATCH -N 1
```

```
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
cell_line=$(echo $SAMPLELIST | sed 's/.txt//' | sed 's/SC_//' | sed 's/Files.//')

java -jar $GATK \
        -T GenotypeGVCFs \
        -R ${RESDIR}/${REF}.fa \
        --variant ${WORKDIR}/HaplotypeCaller.${cell_line}.g.vcf \
        --variant ${WORKDIR}/HaplotypeCaller.${SAMPLE}.g.vcf \
        --dbsnp ${RESDIR}/dbsnp_138.b37.vcf \
        -o ${WORKDIR}/GenotypedGVCFs.${SAMPLE}.vcf
```

6.2.5 Calling somatic variants within the 24 single-cells using MonoVar

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 16

source ReadConfig.sh $1

awk -v dir=$WORKDIR '{print dir"/"$0".recal.bam"}' ${ORIDIR}/${SAMPLELIST} \
> ${WORKDIR}/MonovarSamples.txt
n_samples=$(wc -l ${WORKDIR}/MonovarSamples.txt |  awk '{print $1}')

SAMPLE=$(echo $SAMPLELIST | sed 's/.txt//' | sed 's/SC_//' | sed 's/Files.//')

samtools mpileup \
        -BQ0 \
        -d10000 \
        -f ${RESDIR}${REF}.fa \
        -q 40 \
        -b ${WORKDIR}/MonovarSamples.txt | monovar.py \
        -p 0.002 \
        -a 0.2 \
        -t 0.05 \
        -m 16 \
        -c 0 \
        -f ${RESDIR}${REF}.fa \
        -b ${WORKDIR}/MonovarSamples.txt \
        -o ${WORKDIR}/MonoVarOutput.${SAMPLE}.vcf

fld=$(echo "$n_samples + 10" | bc)
awk -v fld=$fld '{if ($1 !~ /#/) {print $0"\t"gsub(/1|2/,"",$fld)} else {print $0}}' \
${WORKDIR}/MonoVarOutput.${SAMPLE}.vcf | \
awk '{if ($1 ~ /#/) {print $0} else if ($($fld+1) > 1){print $0}}' > \
${WORKDIR}/MonoVarOutput.${SAMPLE}.filtered.vcf
awk -v fld=$fld '{if ($1 !~ /#/) {print $0"\t"gsub(/1/,"",$fld)} else {print $0}}' \
${WORKDIR}/MonoVarOutput.${SAMPLE}.vcf | \
awk '{if ($1 ~ /#/) {print $0} else if ($($fld+1) > 1){print $0}}' > \
${WORKDIR}/MonoVarOutput.${SAMPLE}.filtered.more1het.vcf
```

6.2.6 Creating the files needed to quantify ADO and false SNVs

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1
cell_line=$(echo $SAMPLELIST | sed 's/.txt//' | sed 's/SC_//' | sed 's/Files.//')
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
sample=$SAMPLE


echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/GenotypedGVCFs.${SAMPLE}.vcf \
        -o ${WORKDIR}/6XPaired.${SAMPLE}.vcf \
        -selectType SNP \
          -restrictAllelesTo BIALLELIC \
        -select 'vc.getGenotype(\"$cell_line\").getDP() > 15 \
        && vc.getGenotype(\"$sample\").getDP() > 5'" | bash -

echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/GenotypedGVCFs.HDF.Joint.vcf \
        -o ${WORKDIR}/6XJoint.${SAMPLE}.vcf \
        -selectType SNP \
          -restrictAllelesTo BIALLELIC \
          -select 'vc.getGenotype(\"$cell_line\").getDP() > 15 && \
          vc.getGenotype(\"$sample\").getDP() > 5 '" | bash -



echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/6XPaired.${SAMPLE}.vcf \
        -o ${WORKDIR}/FP_00-01_Paired.${SAMPLE}.vcf \
        -select 'vc.getGenotype(\"$cell_line\").isHomRef() \
        && vc.getGenotype(\"$cell_line\").getAD().1 < 2 &&\
        vc.getGenotype(\"$sample\").isHet()'" | bash -
  # Exclude somatic mutations detected at least in two cells with monovar
  # monovar
vcftools --vcf ${WORKDIR}/FP_00-01_Paired.${SAMPLE}.vcf \
--exclude-bed ${WORKDIR}/MonoVarOutput.${cell_line}.filtered.bed \
--recode --stdout > ${WORKDIR}/FP_00-01_Paired.filtered.${SAMPLE}.vcf
```

```
echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/6XJoint.${SAMPLE}.vcf \
        -o ${WORKDIR}/FP_00-01_Joint.${SAMPLE}.vcf \
        -select 'vc.getGenotype(\"$cell_line\").isHomRef() && \
        vc.getGenotype(\"$cell_line\").getAD().1 < 2 && \
        vc.getGenotype(\"$sample\").isHet()'" | bash -
vcftools --vcf ${WORKDIR}/FP_00-01_Joint.${SAMPLE}.vcf \
--exclude-bed ${WORKDIR}/MonoVarOutput.${cell_line}.filtered.bed --recode \
--stdout > ${WORKDIR}/FP_00-01_Joint.filtered.${SAMPLE}.vcf
# 1/1 -> 0/1
echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/6XPaired.${SAMPLE}.vcf \
        -o ${WORKDIR}/FP_11-01_Paired.${SAMPLE}.vcf \
        -select 'vc.getGenotype(\"$cell_line\").isHomVar() && \
        vc.getGenotype(\"$cell_line\").getAD().0 < 2 && \
        vc.getGenotype(\"$sample\").isHet()'" | bash -
vcftools --vcf ${WORKDIR}/FP_11-01_Paired.${SAMPLE}.vcf \
--exclude-bed ${WORKDIR}/MonoVarOutput.${cell_line}.filtered.more1het.bed \
--recode --stdout > ${WORKDIR}/FP_11-01_Paired.filtered.${SAMPLE}.vcf
echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/6XJoint.${SAMPLE}.vcf \
        -o ${WORKDIR}/FP_11-01_Joint.${SAMPLE}.vcf \
        -select 'vc.getGenotype(\"$cell_line\").isHomVar() &&\
        vc.getGenotype(\"$cell_line\").getAD().0 < 2 && \
        vc.getGenotype(\"$sample\").isHet()'" | bash -
vcftools --vcf ${WORKDIR}/FP_11-01_Joint.${SAMPLE}.vcf \
--exclude-bed ${WORKDIR}/MonoVarOutput.${cell_line}.filtered.more1het.bed \
--recode --stdout > ${WORKDIR}/FP_11-01_Joint.filtered.${SAMPLE}.vcf


# Count homozygous genotypes for both the reference and
# alternative alleles in the bulk with enough coverage in single-cells
echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/6XPaired.${SAMPLE}.vcf \
        -o ${WORKDIR}/TotalHomozygous_6X_Paired.${SAMPLE}.vcf \
        -select 'vc.getGenotype(\"$cell_line\").isHomRef() || \
        vc.getGenotype(\"$cell_line\").isHomVar()'" | bash -
echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/6XJoint.${SAMPLE}.vcf \
        -o ${WORKDIR}/TotalHomozygous_6X_Joint.${SAMPLE}.vcf \
        -select 'vc.getGenotype(\"$cell_line\").isHomRef() ||\
        vc.getGenotype(\"$cell_line\").isHomVar()'" | bash -
```

```
# ADO
# 0/1 -> 0/0
echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/6XPaired.${SAMPLE}.vcf \
        -o ${WORKDIR}/ADO_01-00_Paired.${SAMPLE}.vcf \
        -select 'vc.getGenotype(\"$cell_line\").isHet() && \
        vc.getGenotype(\"$sample\").isHomRef()'" | bash -
echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/6XJoint.${SAMPLE}.vcf \
        -o ${WORKDIR}/ADO_01-00_Joint.${SAMPLE}.vcf \
        -select 'vc.getGenotype(\"$cell_line\").isHet() && \
        vc.getGenotype(\"$sample\").isHomRef()'" | bash -

# 0/1 -> 1/1
echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/6XPaired.${SAMPLE}.vcf \
        -o ${WORKDIR}/ADO_01-11_Paired.${SAMPLE}.vcf \
        -select 'vc.getGenotype(\"$cell_line\").isHet() && \
        vc.getGenotype(\"$sample\").isHomVar()'" | bash -
echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/6XJoint.${SAMPLE}.vcf \
        -o ${WORKDIR}/ADO_01-11_Joint.${SAMPLE}.vcf \
        -select 'vc.getGenotype(\"$cell_line\").isHet() && \
        vc.getGenotype(\"$sample\").isHomVar()'" | bash -


# Count heterozygous genotypes in the bulk with enough coverage in single-cells
echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/6XPaired.${SAMPLE}.vcf \
        -o ${WORKDIR}/TotalHeterozygous_6X_Paired.${SAMPLE}.vcf \
        -select 'vc.getGenotype(\"$cell_line\").isHet()'" | bash -
echo "java -jar $GATK \
        -T SelectVariants \
        -R ${RESDIR}/${REF}.fa \
        -V ${WORKDIR}/6XJoint.${SAMPLE}.vcf \
        -o ${WORKDIR}/TotalHeterozygous_6X_Joint.${SAMPLE}.vcf \
        -select 'vc.getGenotype(\"$cell_line\").isHet()'" | bash -
```

6.2.7 Counting the number of positions showing ADO, false SNVs and passing filters

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

while read sample
do
FP1=$(grep -v "^#" FP_00-01_Joint.filtered.$sample.vcf | wc -l)
FP2=$(grep -v "^#" FP_11-01_Joint.filtered.$sample.vcf | wc -l)
total=$(grep -v "^#" TotalHomozygous_6X_Joint.$sample.vcf | wc -l)
bc=$(bc -l <<< "(($FP1+$FP2)/$total)*100")
echo $FP1 $FP2 $total
printf "%.2f\n" $bc >> FP.Joint.txt
done < /mnt/gluster/distributed/home/uvi/be/tpf/Single-Cell/SC_HDF.txt

while read sample
do
FP1=$(grep -v "^#" FP_00-01_Paired.filtered.$sample.vcf | wc -l)
FP2=$(grep -v "^#" FP_11-01_Paired.filtered.$sample.vcf | wc -l)
total=$(grep -v "^#" TotalHomozygous_6X_Paired.$sample.vcf | wc -l)
bc=$(bc -l <<< "(($FP1+$FP2)/$total)*100")
echo $FP1 $FP2 $total
printf "%.2f\n" $bc >> FP.Paired.txt
done < /mnt/gluster/distributed/home/uvi/be/tpf/Single-Cell/SC_HDF.txt

while read sample
do
ADO1=$(grep -v "^#" ADO_01-00_Joint.$sample.vcf | wc -l)
ADO2=$(grep -v "^#" ADO_01-11_Joint.$sample.vcf | wc -l)
total=$(grep -v "^#" TotalHeterozygous_6X_Joint.$sample.vcf | wc -l)
bc=$(bc -l <<< "(($ADO1+$ADO2)/$total)*100")
echo $sample $ADO1 $ADO2 $total $bc
printf "%.2f\n" $bc >> ADO.Joint.txt
done < /mnt/gluster/distributed/home/uvi/be/tpf/Single-Cell/SC_HDF.txt

while read sample
do
ADO1=$(grep -v "^#" ADO_01-00_Paired.$sample.vcf | wc -l)
ADO2=$(grep -v "^#" ADO_01-11_Paired.$sample.vcf | wc -l)
total=$(grep -v "^#" TotalHeterozygous_6X_Paired.$sample.vcf | wc -l)
bc=$(bc -l <<< "(($ADO1+$ADO2)/$total)*100")
echo $sample $ADO1 $ADO2 $total $bc
printf "%.2f\n" $bc  >> ADO.Paired.txt
done < /mnt/gluster/distributed/home/uvi/be/tpf/Single-Cell/SC_HDF.txt
```

7 Quantifying chimeric amplicons

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1


source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

java -jar $PICARD CollectAlignmentSummaryMetrics \
        R=${RESDIR}/${REF}.fa \
        I=${WORKDIR}/${SAMPLE}.dedup.bam \
        O=${WORKDIR}/${SAMPLE}.alignment_summary_metrics_all.txt \
        MAX_INSERT_SIZE=1000

awk '/^CATEGORY/ {split($0,header);n=1;next; } {if(n!=1) next; for(i=2;i<=NF;++i) \
  printf("%s\t%s\t%s\n",$1,header[i],$i);}' \
  ${WORKDIR}/${SAMPLE}.alignment_summary_metrics_all.txt | \
  column -t | grep "^PAIR" > ${WORKDIR}/${SAMPLE}.alignment_summary_metrics.txt

# Select first in paired reads (64)
split_1=$(samtools view -F 64 ${WORKDIR}/${SAMPLE}.dedup.bam | \
  grep "SA:Z" | awk '{print $1}' | sort -k1,1 | uniq | wc -l)
# Select second in paired reads (128)
split_2=$(samtools view -F 128 ${WORKDIR}/${SAMPLE}.dedup.bam | \
  grep "SA:Z" | awk '{print $1}' | sort -k1,1 | uniq | wc -l)
split=$(echo "$split_1+$split_2" | bc -l)
discordant=$(grep "PCT_CHIMERAS" ${WORKDIR}/${SAMPLE}.alignment_summary_metrics.txt | \
  awk '{print $3*100}') # Although the variable name is PCT_ADAPTER,
                        # it is a proportion, so I multiply by 100

echo $split $discordant
```

8 Estimating copy-number profiles and MAD values

Downsample bam files to the lowest sequencing depth found for each cell line

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE

config=$(basename $1 | sed 's/.txt//' | sed 's/Config.//' | sed 's/.SC//')
sequencing_depth=$(awk '{print $2}' ${WORKDIR}/${SAMPLE}.seqdepth.txt)
# Obtain the lowest seqdepth of all cells from the same cell line
# by cat *HDF*.seqdepth.txt > HDF.seqdepths.txt
subsampling_depth=$(awk '{print $2}' ${WORKDIR}/${config}.seqdepths.txt  \
| sort -n | head -n 1)   to create the file
## Calculating subsampling probability
probability=`bc -l <<< "scale=3; $subsampling_depth / $sequencing_depth"`
echo "Downsampling probability: "$probability
strategy="Chained"
echo "Downsampling following "${strategy}" strategy. \
From "${sequencing_depth}"X to "${subsampling_depth}"X."
java -jar $PICARD DownsampleSam \
        INPUT=${WORKDIR}/${SAMPLE}.dedup.bam \
        OUTPUT=${WORKDIR}/${SAMPLE}.dedup.ps.bam \
        RANDOM_SEED=1 \
        PROBABILITY=${probability} \
        STRATEGY=$strategy
```

Create ginkgo input

```
#!/bin/sh
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task 1

source ReadConfig.sh $1
SAMPLE=$(sed "${SLURM_ARRAY_TASK_ID}q;d" ${ORIDIR}/${SAMPLELIST})
echo $SAMPLE


samtools view -bq 20 ${WORKDIR}/${SAMPLE}.dedup.ps.bam | \
  bedtools bamtobed -i stdin | gzip > \
  ${WORKDIR}/${SAMPLE}.dedup.ps.bed.gz
```

8.1 Running ginkgo on the web server

We added bed files to the ginkgo web server, selected all cells, wrote down a job name and modified some advanced parameters. We adjusted *Binning Simulation Options* to *150 bp* reads and *BWA mapping*.