QTL pipeline : from fasta file to QTL mapping and going through DNA sequences

Gaëlle Bustarret

May 2024

1 Basic commands

1.1 How to use the terminal

Moving between directories :

```
cd directory/
```

Listing files inside a directory :

```
ls
```

Creating a new directory :

```
mkdir new_directory/
```

Creating a file :

```
echo Hello World > file.txt
```

Editing a file :

```
vim file.txt
```

To modify, you type "i" then change then ESC, then type " :x" to exit.

Deleting a file :

```
rm file.txt
```

1.2 How to connect to the cluster

```
$ ssh -Y gbustarr@cheops.rrz.uni-koeln.de
```

1.3 How to copy a file

1.3.1 Locally

```
$ cp /path/to/file /path/to/target/directory/
```

1.3.2 From cluster to local or vice-versa

```
$ scp -r gbustarr@cheops.rrz.uni-koeln.de:/path/to/file /path/to/target/directory/
```

Beware : You have to be doing it from your computer files and not from the cluster

1.3.3 From docker to local or vice-versa

```
$ docker cp 9bfc22450f7a:/path/to/file /path/to/target/directory/
```

Beware : You have to be doing it from your computer files and not from the cluster

1.4 Reducing image size on Mac

```
$ sips -Z 1200 *.png
```

1.5 Changing chromosome names

1.5.1 On BAM files

```
samtools view -H foo.bam > header

vim header

#change your chomosome names

samtools reheader header foo.bam > new_foo.bam
```

1.5.2 On VCF files

```
#Number of unique chromosome in the vcf file:

$ bcftools query -f '%CHROM\n' MIL2-KAD1_noindels_0.01.vcf.gz|uniq|wc -l

#Register the unique chromosome name in a file called
↪  "chrm_correspondance_name.txt"

$ bcftools query -f '%CHROM\n' MIL2-KAD1_noindels_0.01.vcf.gz|uniq >
↪  chrm_correspondance_name.txt
```

Open the text file (using vim) and add the new chromosome name, the text file should be :
old_name new_name
old_name2 new_name2

So for *A. Thaliana* :
NC_003070.9 1
NC_003071.7 2
NC_003074.8 3
NC_003075.7 4
NC_003076.8 5
NC_037304.1 MT
NC_000932.1 Pltd

For each vcf file, do :

```
#Changing the chromosome name

bcftools annotate --rename-chrs chrm_correspondance_name.txt
 ↪  MIL2-KAD1_noindels_0.01.vcf.gz -Oz -o
 ↪  MIL2-KAD1_noindels_0.01_chrname.vcf.gz

#checking the new name in the vcf

bcftools query -f '%CHROM\n' MIL2-KAD1_noindels_0.01_chrname.vcf.gz|uniq
```

2 FastQC - quality report for fasta files

You have to download FastQC if you want to use it on a computer. You can then do the quality control directly on the app. You can also do it on the cluster using the terminal and the following commands :

```
$ module load fastqc

$ mkdir -p ~/QTL_analysis/fastqc
```

To run the program :

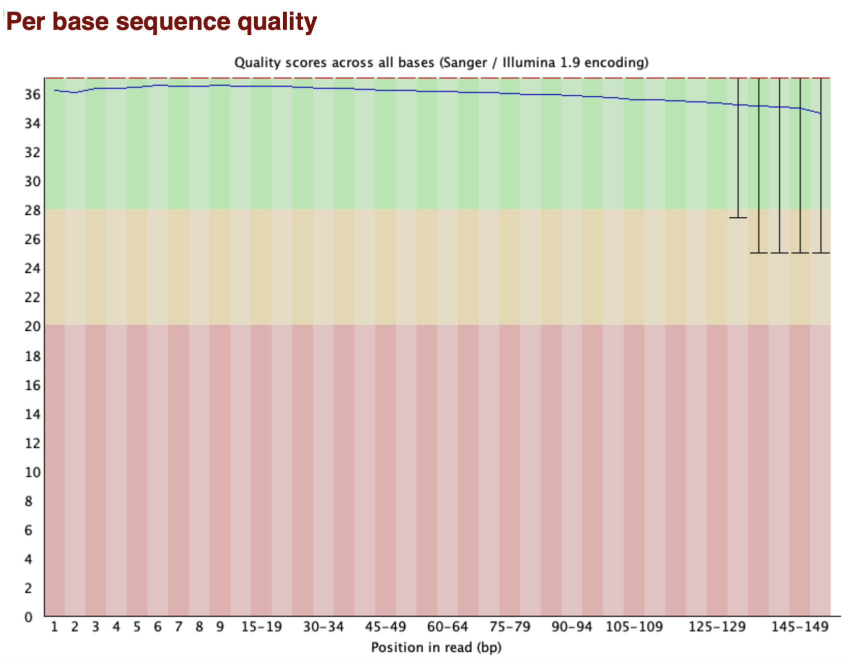
```
$ fastqc -o ~/QTL_analysis/fastqc/
 ↪  /projects/ag-demeaux/jfloret/QTL_FT/MIL2_KAD1/MIL2-KAD1_E_R1.fastq.gz
 ↪  /projects/ag-demeaux/jfloret/QTL_FT/MIL2_KAD1/MIL2-KAD1_E_R2.fastq.gz
```

To open the output file :

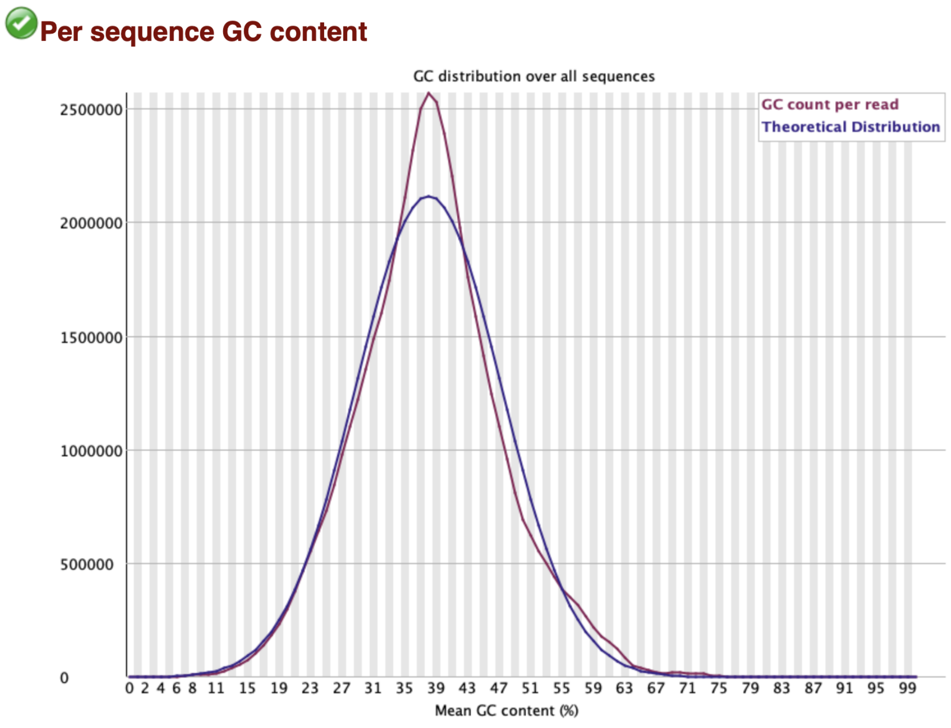
```
$ cd ~/QTL_analysis/fastqc/

$ firefox MIL2-KAD1_E_R1_fastqc.html
```

You can then check for quality of reads for example :



You want quality reads (within the green interval if possible). It is normal that quality decreases at the end of reads due to sequencing techniques. You can also check the GC content :



3 Quality trimming, with Fastp

All the following is done on the **cluster** (else it will be written).

3.1 Downloading Fastp

Download the latest build :

```
$ mkdir ~/QTL_analysis/tools

$ cd ~/QTL_analysis/tools

$ wget http://opengene.org/fastp/fastp

$ chmod a+x ./fastp

$ ./fastp
```

Setting up path variables permanently :

```
$ echo 'export PATH="$PATH:~/QTL_analysis/tools"' >> ~/.bashrc

$ source ~/.bashrc
```

Check if it is properly installed :

```
$ fastp -h
```

3.2 Using Fastp

Set up a new directory and run Fastp :

```
$ mkdir -p ~/QTL_analysis/fastp

$ fastp --in1
 ↪  /projects/ag-demeaux/jfloret/QTL_FT/MIL2_KAD1/MIL2-KAD1_E_R1.fastq.gz
 ↪  --in2
 ↪  /projects/ag-demeaux/jfloret/QTL_FT/MIL2_KAD1/MIL2-KAD1_E_R2.fastq.gz
 ↪  --out1 ~/QTL_analysis/fastp/MIL2-KAD1_E_trim_1.fastq --out2
 ↪  ~/QTL_analysis/fastp/MIL2-KAD1_E_trim_2.fastq -l 50 -h
 ↪  ~/QTL_analysis/fastp/report_html &> ~/QTL_analysis/fastp/log_file
```

4 Mapping trimmed FASTQ to reference

4.1 Downloading the reference genome

Where to find the reference genome : https://www.ncbi.nlm.nih.gov/datasets/genome/GCF_000001735.4/

Create new directories :

```
$ mkdir -p ~/QTL_analysis/{mapping_bwa,mapping_qc,reference_fasta}
```

Copy your reference fasta :
Beware : you have to be in the computer files for this step.

```
$ scp ~/QTL_analysis/reference_fasta/GCF_000001735.4_TAIR10.1_genomic.fna
 ↪  ~/QTL_analysis/reference_fasta/
```

4.2 Mapping to reference genome

Loading modules :

```
$ module load bwamem2/2.2.1

$ module load samtools/1.13
```

Creating an index/database of your reference :

```
$ bwa-mem2 index
 ↪  ~/QTL_analysis/reference_fasta/GCF_000001735.4_TAIR10.1_genomic.fna
```

As the reference genome uses different chromosome names, you should change the index file to **replace them by your chromosome names here using vim on the .fna.fai file**. *Beware to keep the same order when changing the chromosome names.*

Mapping to reference fasta :

```
$ bwa-mem2 mem -M -t 20
 ↪  ~/QTL_analysis/reference_fasta/GCF_000001735.4_TAIR10.1_genomic.fna
 ↪  ~/QTL_analysis/fastp/MIL2-KAD1_E_R_trim_1.fastq
 ↪  ~/QTL_analysis/fastp/MIL2-KAD1_E_R_trim_2.fastq >
 ↪  ~/QTL_analysis/mapping_bwa/MIL2-KAD1_E.sam
```

4.3 Convert SAM to BAM

```
$ samtools view -bS ~/QTL_analysis/mapping_bwa/MIL2-KAD1_E.sam >
↪  ~/QTL_analysis/mapping_bwa/MIL2-KAD1_E.bam
```

If you have a sample that has been separated into two files, you can combine them following these commands :

```
$ cd ~/QTL_analysis

$ mkdir merge

$ cp mapping_bwa/MIL2-KAD1_L1.bam merge/

$ cp mapping_bwa/MIL2-KAD1_L2.bam merge/

$ cd merge

$ samtools merge MIL2-KAD1_L.bam *.bam

$ cp MIL2-KAD1_L.bam ../mapping_bwa/
```

4.4 Sort BAM file

```
$ samtools sort -@ 20 -o ~/QTL_analysis/mapping_bwa/MIL2-KAD1_E_sorted.bam
↪  ~/QTL_analysis/mapping_bwa/MIL2-KAD1_E.bam
```

Index the sorted BAM file

```
$ samtools index ~/QTL_analysis/mapping_bwa/MIL2-KAD1_E_sorted.bam
```

Delete unsorted BAM and SAM files

```
$ rm ~/QTL_analysis/mapping_bwa/MIL2-KAD1_E.bam

$ rm ~/QTL_analysis/mapping_bwa/MIL2-KAD1_E.sam
```

5 BAM quality control

5.1 Downloading qualimap and packages

```
$ wget https://bitbucket.org/kokonech/qualimap/downloads/qualimap_v2.3.zip

$ unzip qualimap_v2.3.zip

$ cd qualimap_v2.3
```

Setting up path variables permanently :

```
$ echo 'export PATH="~/QTL_analysis/qualimap_v2.3:$PATH"' >> ~/.bashrc

$ source ~/.bashrc
```

Beware to use the right " and ' symbols for your terminal, or else you might get an error. To change the file, you can use vim.

Installing necessary R packages :

```
$ R
```

```
> install.packages("XML")
```

```
> install.packages("optparse")
```

```
> install.packages("BiocManager")
```

```
> BiocManager::install("NOISeq")
```

```
> q()
```

5.2 Running qualimap

```
$ qualimap bamqc -bam ~/QTL_analysis/mapping_bwa/MIL2-KAD1_E_sorted.bam
↪  -outdir ~/QTL_analysis/mapping_qc/MIL2-KAD1_E -outformat html -outfile
↪  MIL2-KAD1_E
```

If you run out of memory, you can add "–java-mem-size=2400M" at the end.

Create a file "input_groups.txt" with three columns "# names path group"

```
$ cat > ~/QTL_analysis/mapping_qc/input_groups.txt
```

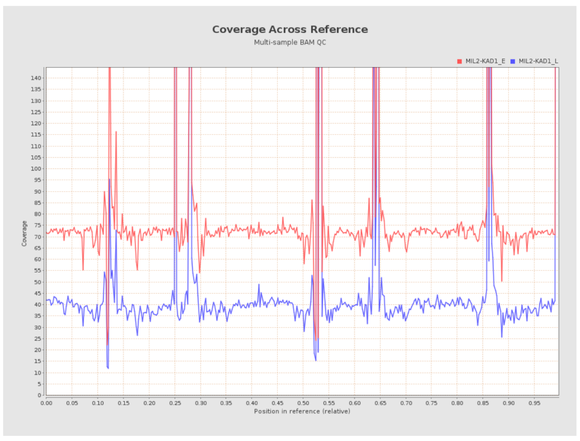
```
> MIL2-KAD1_E ~/QTL_analysis/mapping_qc/MIL2-KAD1_E/
```

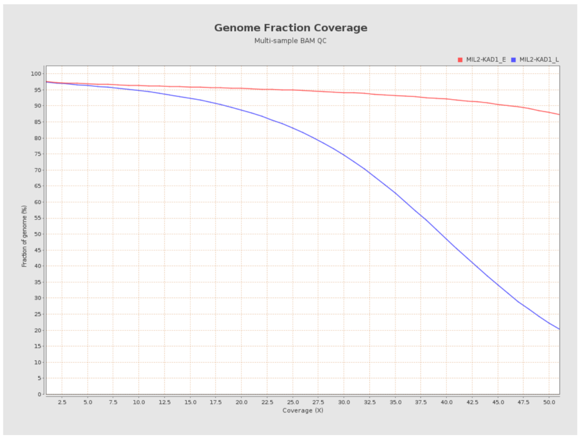
```
> MIL2-KAD1_L ~/QTL_analysis/mapping_qc/MIL2-KAD1_L/
```

use Ctrl+D to exit (do it twice).

```
$ qualimap multi-bamqc -d ~/QTL_analysis/mapping_qc/input_groups.txt
↪  -outdir ~/QTL_analysis/mapping_qc/multibamqc/ -outformat PDF
```

You can then check the quality of your mapping. For instance, you can check for coverage :





This gives you the depth of coverage across the genome and the average coverage. Peaks locations correspond to centromeres.

6 Filter and index BAM files

Create new directories

```
$ mkdir -p ~/QTL_analysis/{filtered_bam,variant_calling,variant_filtering}
```

```
$ module load samtools
```

Run the following commands using "for loop" to filter and index bam files based on mapping quality :

To list prefixes of all sorted bam files and save in variable

```
$ samples=$(ls /scratch/gbustarr/QTL_analysis/mapping_bwa/*_sorted.bam |
↪  xargs -n 1 basename | sed 's/\_sorted.bam//')
```

```
$ echo "$samples"
```

Loop through each sample

```
$ for sample in $samples
```

```
$ do
```

```
#Quality filtering and removing PCR duplicates with Samtools, keep only
↪  properly aligned paired-end reads
```

```
#(-f 3), Mapping quality (-q 30), Exclude secondary alignments and reads
↪  failing quality checks (-F 264)
```

```
#Remove duplicates (samtools rmdup)
```

```
$ samtools view -b -o /home/gbustarr/QTL_analysis/filtered_bam/${sample}_Ba⌋
↪  mQualFilt-f3Q30F264.bam -f3 -q 30 -F 264
↪  /scratch/gbustarr/QTL_analysis/mapping_bwa/${sample}_sorted.bam
```

```
$ samtools rmdup -s /home/gbustarr/QTL_analysis/filtered_bam/${sample}_BamQ⌋
↪  ualFilt-f3Q30F264.bam
↪  /home/gbustarr/QTL_analysis/filtered_bam/${sample}_BamQualFilt-f3Q30F26⌋
↪  4_nodup.bam
```

```
#Index the filtered_nodup.bam file
```

```
$ samtools index /home/gbustarr/QTL_analysis/filtered_bam/${sample}_BamQual⌋
↪  Filt-f3Q30F264_nodup.bam
```

```
$ done
```

7 Variants calling and filtering

Load required module :

```
$ module load bcftools/1.18
```

7.1 Preparing your reference file

For the analysis of a F2 population, you might want to use the parent genome as a reference, instead of the reference genome. If you want to use the reference genome, you can skip this part.

The parents genomes are supposed to be already sorted and index (if else, come back to the preceding steps).
First you have to load the necessary modules :

```
$ module load samtools/1.13
```

```
$ module load bcftools/1.18
```

Then you can prepare your files :

```
bcftools mpileup -f /scratch/gbustarr/QTL_analysis/reference_fasta/GCF_00000173⌋
↪  5.4_TAIR10.1_genomic.fna filtered_bam/MIL2.bam | bcftools call -c |
↪  vcfutils.pl vcf2fq > filtered_bam/MIL2_consensus.fa
```

Creating an index/database of your reference :

```
$ samtools faidx ~/QTL_analysis/filtered_bam/MIL2_consensus.fa
```

7.2 Variant calling with DP /AD in FORMAT column

DP : read depth at this position for this sample. This can be useful in the following studies.

Load required module :

```
$ module unload vcftools/0.1.17
```

```
$ module unload gnu
```

```
$ module load bcftools/1.18
```

Set path for input and output directories, and reference fasta

```
$ input_dir="/home/gbustarr/QTL_analysis/filtered_bam/"
```

```
$ output_dir="/home/gbustarr/QTL_analysis/variant_calling/"
```

```
$ reference_fasta="/home/gbustarr/QTL_analysis/filtered_bam/MIL2_consensus.⌋
↪  fa"
```

Run bcftools :

```
$ bcftools mpileup -E -q 30 --threads 20 -o "${output_dir}MIL2-KAD1_ParentC⌋
↪  alling_BamQualFilt-f3Q30F264_nodup_DP.bcf" --annotate
↪  FORMAT/AD,FORMAT/DP -f "$reference_fasta" -b "${input_dir}bam_files.txt"
```

```
$ bcftools call -c -p 0.01 -O z --threads 20 -o "${output_dir}MIL2-KAD1_Par⌋
↪  entCalling_BamQualFilt-f3Q30F264_nodup_DP_raw.vcf.gz"
↪  "${output_dir}MIL2-KAD1_ParentCalling_BamQualFilt-f3Q30F264_nodup_DP.bc⌋
↪  f"
```

Basic filtering by removing all monomorphic variant sites

```
$ bcftools view -i 'AC>0' "${output_dir}MIL2-KAD1_ParentCalling_BamQualFilt⌋
↪  -f3Q30F264_nodup_DP_raw.vcf.gz" -o
↪  "${output_dir}MIL2-KAD1_ParentCalling_BamQualFilt-f3Q30F264_nodup_DP_On⌋
↪  lyPolymorphic.vcf.gz"
```

Simplify and shorten the names of samples

```
$ zcat "${output_dir}MIL2-KAD1_ParentCalling_BamQualFilt-f3Q30F264_nodup_DP⌋
↪  _OnlyPolymorphic.vcf.gz" | awk -F '\t' 'BEGIN{OFS="\t"} {if ($1 ~
↪  /^#CHROM/) {for (i=10; i<=NF; i++) {sub(".*/", "", $i);
sub("\\.BamQualFilt-f3Q30F264_nodup\\.bam", "", $i)}} print }' | gzip -c >
↪  "${output_dir}MIL2-KAD1_ParentCalling_BamQualFilt-f3Q30F264_nodup_DP_On⌋
↪  lyPolymorphic_shortNam.vcf.gz"
```

Do the following to load vcftools

```
$ module unload bcftools/1.18
```

```
$ module unload gnu
```

```
$ module load vcftools/0.1.17
```

Additional filters

```
$ vcftools --gzvcf "${output_dir}MIL2-KAD1_ParentCalling_BamQualFilt-f3Q30F⌋
↪  264_nodup_DP_OnlyPolymorphic_shortNam.vcf.gz" --minDP 10 --minGQ 20
↪  --minQ 30 --max-missing 0.80 --remove-indels --max-alleles 2 --recode
↪  --recode-INFO-all --stdout | gzip -c >
↪  "${output_dir}MIL2-KAD1_ParentCalling_BamQualFilt-f3Q30F264_nodup_DP_On⌋
↪  lyPolymorphic_shortNam_DP10GQ20Q30_Mis80NoIndel.vcf.gz"
```

8 From the VCF file to a csv / table file

In order to use the QTLseqr package on R, you have to have table files. You must thus convert your VCF files to csv or tables. This can be done using either GATK or R/vcfR. Note that the second (using R instead of gatk) can be practical to have a few filters and graphs, and more likely to work.

8.1 Using GATK

8.1.1 From terminal

The first way to achieve that is by using GATK on your terminal (easier).
Install Miniconda : `https://docs.anaconda.com/free/miniconda/`

Install GATK :

```
conda install bioconda::gatk4
```

8.1.2 From a docker image

Install Docker Desktop : `https://www.docker.com/products/docker-desktop/`

Install the docker image of GATK : `https://gatk.broadinstitute.org/hc/en-us/articles/360035889991--How-to-Run-GATK-in-a-Docker-container`

Import your files onto the docker image :

```
docker cp ~/QTL_analysis/variant_calling/MIL2-KAD1_onlyPolymorph_noindels_0⌋
 ↪  .01_chrname.vcf.gz
 ↪  9bfc22450f7a:/gatk/MIL2-KAD1_onlyPolymorph_noindels_0.01_chrname.vcf.gz
```

8.1.3 Indexing the reference file

You have to prepare your reference file by creating a .dict and a .fai (`https://gatk.broadinstitute.org/hc/en-us/articles/360035531652-FASTA-Reference-genome-format`).

```
gatk CreateSequenceDictionary -R GCF_000001735.4_TAIR10.1_genomic.fa

samtools faidx GCF_000001735.4_TAIR10.1_genomic.fa
```

8.1.4 Indexing your VCF file

```
gatk IndexFeatureFile -F
 ↪  MIL2-KAD1_onlyPolymorph_noindels_0.01_chrname.vcf.gz
```

If you have a problem with the format here, you can either try to download the file again from the cluster or re-do a few of the previous steps. A other solution can be to annotate your file, as if to change the chromosome names but without actually changing the names :

```
#Number of unique chromosome in the vcf file:

bcftools query -f '%CHROM\n' MIL2-KAD1_noindels_0.01.vcf.gz|uniq|wc -l

#Register the unique chromosome name in a file called
 ↪  "chrm_correspondance_name.txt"

bcftools query -f '%CHROM\n' MIL2-KAD1_noindels_0.01.vcf.gz|uniq >
 ↪  chrm_correspondance_name.txt
```

Open the text file (using vim) and add the new chromosome name, the text file should be :
1 1
2 2
3 3
4 4
5 5
MT MT
Pltd Pltd

For each vcf file, do :

```
#Changing the chromosome name

bcftools annotate --rename-chrs chrm_correspondance_name.txt
 ↪  MIL2-KAD1_noindels_0.01.vcf.gz -Oz -o
 ↪  MIL2-KAD1_noindels_0.01_chrname.vcf.gz
```

8.1.5 Creating your table file

```
$ REF=GCF_000001735.4_TAIR10.1_genomic.fa

$ NAME=MIL2-KAD1_noindels_0.01_chrname

$ gatk VariantsToTable \
-R ${REF} \
-V ${NAME}.vcf.gz \
-F CHROM -F POS -F REF -F ALT \
-GF AD -GF DP -GF GQ -GF PL \
-o ${NAME}.table
```

If you have an error concerning memory space, you can add "–java-options "-Xmx2g"".

8.2 Using R

8.2.1 Libraries needed

From this point on, you can do the commands directly on RStudio.

```
library(vcfR)
library(tidyverse)
library(reshape2)
library(ggplot2)
library(UpSetR)
library(venn)
```

8.2.2 Data set

Set your working directory and read your vcf file using read.vcfR() (returns a vcfR class object).

```
setwd("~/QTL_analysis")
vcf_file <- "variant_calling/MIL2-KAD1_DP_OnlyPolymorphic_NoIndel.vcf.gz"

vcf <- vcfR::read.vcfR(file = vcf_file)
```

If you want to check the type or class of an object, you can use : summary(), class(), is().

```
summary(vcf)
```

You can also look at slot names using :

```
slotNames(vcf)
```

8.2.3 Extracting data

Convert the slot gt to a tibble type data frame using extract_gt_tidy. You can choose the fields to keep (example : $extract\_gt\_tidy(vcf.snv, format\_fields = c(\imath AC\jmath, \imath AN\jmath, \imath MQ\jmath)))$. Get the tidy list using $vcfR2tidy(vcf.snv, format_fields = c(\imath GT\jmath, \imath AD\jmath, \imath DP\jmath), info\_fields = c(\imath AC\jmath, \imath AN\jmath, \imath MQ\jmath)))$.

```
extract_gt_tidy(vcf.snv)
```

```
vcf.tidy.list <- vcfR2tidy(vcf.snv)
```

Get the names and heads using :

```
names(vcf.tidy.list)
```

```
head(vcf.tidy.list$meta)
```

```
head(vcf.tidy.list$fix)
```

```
head(vcf.tidy.list$gt)
```

“ChromKey”, “POS” are common columns so they can be used to join data frames. Each location is unique for each variant but common between samples.

```
vcf.tidy <- dplyr::full_join(vcf.tidy.list$fix , vcf.tidy.list$gt, by =
↪  c(‘‘ChromKey’’, ‘‘POS’’))
```

You can get the head using :

```
head(vcf.tidy)
```

Each row has the following information :

```
# CHROM <- ‘‘Chr1’’
# POS <- 10000
# REF <- ‘‘A’’
# ALT <- ‘‘T’’
# ID  <- paste(CHROM, POS, REF, ALT, sep=‘‘_’’) # results: Chr1_10000_A_T
```

If the ID is NULL, you can create it as shown in the previous example.

```
vcf.tidy <- dplyr::mutate(vcf.tidy, ID = dplyr::if_else(is.na(ID),
↪  paste(CHROM, POS, REF, ALT, sep=‘‘_’’), ID))
```

```
head(vcf.tidy)
```

8.2.4 Filtering

To look for variants with more than one ALT allele :

```
vcf.tidy.multiAlt <- dplyr::filter(vcf.tidy, str_detect(string = ALT,
↪  pattern = ‘‘,’’))
```

```
vcf.tidy.multiAlt %>% select(ID, ALT, gt_AD) %>% head
```

Keep variants with only one alternative allele :

```
filter(vcf.tidy, !str_detect(string = ALT, pattern = ‘‘,’’)) %>% select(ID,
↪  ALT, gt_AD) %>% head
```

```
vcf.No.multiAlt <- dplyr::filter(vcf.tidy, !str_detect(string = ALT,
↪  pattern = ","))
```

Separate the column in two :

```
vcf.No.multiAlt <- tidyr::separate(data = vcf.No.multiAlt, col = gt_AD,
↪  c("gt_ref.AD", "gt_alt.AD"), sep = ",", remove=FALSE)
```

Convert to numeric format :

```
vcf.No.multiAlt <- vcf.No.multiAlt %>% dplyr::mutate(gt_ref.AD =
↪  as.numeric(gt_ref.AD), gt_alt.AD = as.numeric(gt_alt.AD))
```

```
dplyr::select(vcf.No.multiAlt, ID, Indiv, gt_AD, gt_ref.AD, gt_alt.AD) %>%
↪  head()
```

If the raw read depth (gt_DP) is =0 then the ratio is also =0 (no division by 0)

```
vcf.No.multiAlt <- dplyr::mutate(vcf.No.multiAlt, gt_AR = if_else(gt_DP ==
↪  0, 0, gt_alt.AD/gt_DP))
```

```
dplyr::select(vcf.No.multiAlt, ID, Indiv, gt_AD, gt_ref.AD, gt_alt.AD,
↪  gt_AR) %>% head()
```

8.2.5 Plots

```
p <- ggplot(data=vcf.No.multiAlt) +
  geom_histogram(aes(QUAL, fill=Indiv), bins = 35) +
  xlab("Quality score") + xlim(c(0,100)) +  theme_minimal()
p

# gt_DP : Number of high-quality bases
p <- ggplot(data=vcf.No.multiAlt) +
  geom_histogram(aes(gt_DP, fill=Indiv), bins = 35) +
  xlab("Variant coverage")+ xlim(c(0,300))  + theme_minimal()
p

p <- ggplot(data=vcf.No.multiAlt) +
  geom_histogram(aes(gt_DP, fill=Indiv), bins = 35) +
  xlab("Variant coverage")+ xlim(c(0,100))  + theme_minimal()
p

p <- ggplot(data=vcf.No.multiAlt) +
  geom_histogram(aes(gt_alt.AD, fill=Indiv), bins = 35) +
  xlab("Alternative allele depth") + xlim(c(0,50)) + theme_minimal()
p

p <- ggplot(data=vcf.No.multiAlt) +
  geom_histogram(aes(gt_AR, fill=Indiv), bins = 35) +
  xlab("Alternative allele ratio") + theme_minimal()
p
```

Remove variants with quality score below 30

```
vcf.No.multiAlt.Q30 <- dplyr::filter(vcf.No.multiAlt, QUAL >= 30)
```

Remove position covered by less than 4 reads (alternative allele need to be covered by more than 2 reads)

```
vcf.No.multiAlt.flt <- dplyr::mutate(vcf.No.multiAlt.Q30, gt_DP = if_else(gt_DP
↪  < 4, 0, as.numeric(gt_DP)), gt_alt.AD = if_else(gt_alt.AD < 2, 0,
↪  as.numeric(gt_alt.AD)))

p <- ggplot(data=vcf.No.multiAlt.flt) +
  geom_histogram(aes(QUAL, fill=Indiv), bins = 35) +
  xlab("Quality score") + xlim(c(0,100)) +  theme_minimal()
p

p <- ggplot(data=vcf.No.multiAlt.flt) +
  geom_histogram(aes(gt_DP, fill=Indiv), bins = 35) +
  xlab("Variant coverage")+ xlim(c(0,100))  + theme_minimal()
p

p <- ggplot(data=vcf.No.multiAlt.flt) +
  geom_histogram(aes(gt_alt.AD, fill=Indiv), bins = 35) +
  xlab("Alternative allele depth") + xlim(c(0,50)) + theme_minimal()
p

p <- ggplot(data=vcf.No.multiAlt.flt) +
  geom_histogram(aes(gt_AR, fill=Indiv), bins = 35) +
  xlab("Alternative allele ratio") + theme_minimal()
p
```

You can thus check the distribution of heterozygotes vs homozygotes :

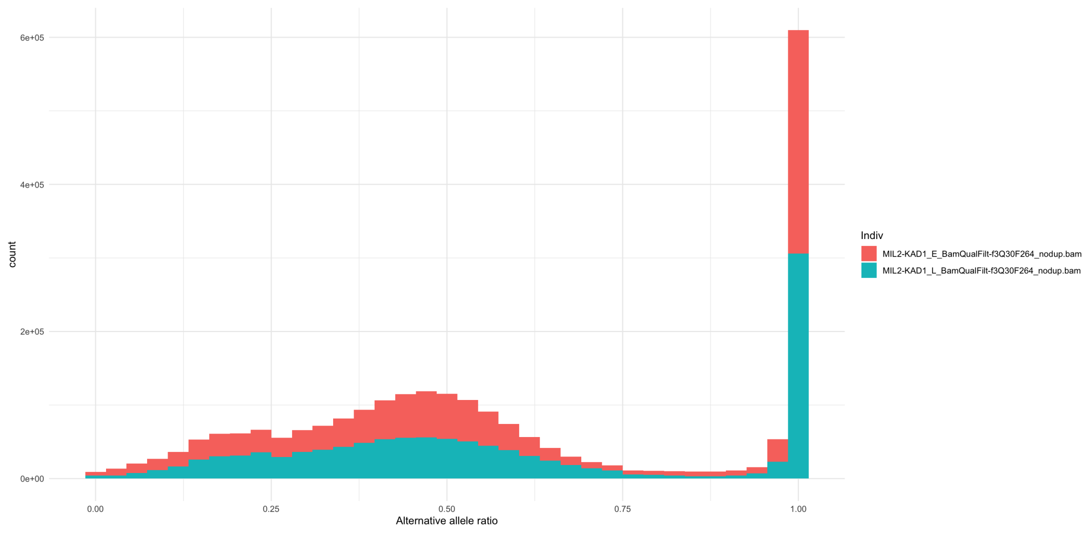


FIGURE 1 – A ratio of 0.5 means that the individual is heterozygous ; 1 means homozygous

If you want to calculate a fraction of count between values on the diagrams :

```
# Calculate counts for the specified range

count_range <- sum(vcf.No.multiAlt.flt$gt_AR >= 0.3 &
↪  vcf.No.multiAlt.flt$gt_AR <= 0.75)

# Calculate total count

total_count <- length(vcf.No.multiAlt.flt$gt_AR)

# Calculate fraction

fraction <- count_range / total_count

fraction
```

8.2.6 Filtering on allele ration

```
# Transform the data matrix to "wide" -> ID in row et Indiv in column

vcf.mat.AR <- reshape2::dcast(vcf.No.multiAlt.flt, ID~Indiv,
↪  value.var="gt_AR")

head(vcf.mat.AR)

# Remove the ID column

vcf.mat.AR <- dplyr::select(vcf.mat.AR, -ID)

# Replace NA by 0

vcf.mat.AR <- dplyr::mutate_at(vcf.mat.AR, .vars = names(vcf.mat.AR), .funs
↪  = function(x){dplyr::if_else(is.na(x), 0, as.double(x))})

head(vcf.mat.AR)
```

8.2.7 Venn and upset diagram

```
# Venn function needs a binary matrix (absence / presence)

vcf.mat.bin <- dplyr::mutate_at(vcf.mat.AR, .vars = names(vcf.mat.AR),
↪  .funs = function(x){dplyr::if_else(x > 0, 1, 0)})

#head(vcf.mat.bin)

venn::venn(vcf.mat.bin, zcolor = "style")

UpSetR::upset(vcf.mat.bin, sets=names(vcf.mat.AR), order.by = "freq",
↪  nintersects = NA, mainbar.y.label = "SNP Intersections", sets.x.label =
↪  "Number of SNP")
```

```
UpSetR::upset(vcf.mat.bin, sets=names(vcf.mat.AR), keep.order = T,
 ↪  nintersects = NA, mainbar.y.label = "SNP Intersections", sets.x.label =
 ↪  "Number of SNP")
```

8.2.8 Creating a table for QTLseqR

Getting libraries :

```
library(dplyr)

library(tidyr)
```

Group by CHROM, POS, REF, and ALT :

```
grouped <- vcf.tidy %>%
 group_by(CHROM, POS, REF, ALT)
```

Select necessary columns :

```
grouped <- select(grouped, CHROM, POS, REF, ALT, Indiv, gt_PL, gt_DP, gt_AD)
```

Pivot the data to wide format :

```
result <- grouped %>%
pivot_wider(names_from = Indiv, values_from = c("gt_PL", "gt_DP", "gt_AD"),
 ↪  names_sep = "_")
```

Rename the columns :

```
  result <- result %>%
rename("MIL2-KAD1_E.PL" =
 ↪  "gt_PL_MIL2-KAD1_E_BamQualFilt-f3Q30F264_nodup.bam") %>%
rename("MIL2-KAD1_L.PL" =
 ↪  "gt_PL_MIL2-KAD1_L_BamQualFilt-f3Q30F264_nodup.bam") %>%
rename("MIL2-KAD1_E.DP" =
 ↪  "gt_DP_MIL2-KAD1_E_BamQualFilt-f3Q30F264_nodup.bam") %>%
rename("MIL2-KAD1_L.DP" =
 ↪  "gt_DP_MIL2-KAD1_L_BamQualFilt-f3Q30F264_nodup.bam") %>%
rename("MIL2-KAD1_E.AD" =
 ↪  "gt_AD_MIL2-KAD1_E_BamQualFilt-f3Q30F264_nodup.bam") %>%
rename("MIL2-KAD1_L.AD" = "gt_AD_MIL2-KAD1_L_BamQualFilt-f3Q30F264_nodup.bam")
```

Writing a csv file :

```
write.csv(result,
 ↪  "variant_calling/MIL2-KAD1_DP_OnlyPolymorphic_NoIndel.csv", row.names =
 ↪  FALSE)
```

Writing .table file :

```
write.table(result,
 ↪  "variant_calling/MIL2-KAD1_DP_OnlyPolymorphic_NoIndel.table", sep =
 ↪  "\t", row.names = FALSE)
```

9 QTL mapping

Using the R package QTLseqR :

9.1 Installing and loading QTLseqR

To install a new package, run :

```
install.packages("QTLseqr")
```

To load the package :

```
library(QTLseqr)
library(ggplot2)
library(dplyr)
library(tidyr)
```

9.2 Loading data

Set sample and file names :

```
HighBulk <- "MIL2-KAD1_E"

LowBulk <- "MIL2-KAD1_L"

setwd("~/QTL_analysis")

table_file <- "variant_calling/MIL2-KAD1_DP_OnlyPolymorphic_NoIndel.table"

csv_file <- "variant_calling/MIL2-KAD1_DP_OnlyPolymorphic_NoIndel.csv"
```

Choose which chromosomes will be included in the analysis (i.e. exclude smaller contigs). Beware to write the name same as in your files !

```
Chroms <- paste0(rep("", 5), 1:5)
```

Import SNP data from file

```
df <- importFromGATK(
    file = table_file,
    highBulk = HighBulk,
    lowBulk = LowBulk,
    chromList = Chroms
)
```

9.3 Analysis

Filter SNPs based on some criteria :

```
df_filt <- filterSNPs(
    SNPset = df,
    refAlleleFreq = 0.20,
    minTotalDepth = 100,
    maxTotalDepth = 400,
    minSampleDepth = 40,
    minGQ = 99
)
```

Run G' analysis

```
df_filt <- runGprimeAnalysis(SNPset = df_filt, windowSize = 1e6, outlierFilter
↪  = "deltaSNP")
```

Run QTLseq analysis

```
df_filt <- runQTLseqAnalysis(
    SNPset = df_filt,
    windowSize = 1e6,
    popStruc = "F2",
    bulkSize = 250,
    replications = 10000,
    intervals = c(95, 99)
)
```

9.4 Plots and mapping

SNP-index :

```
### To create the basic plot
# Convert data to long format for ggplot2
df_long <- df_filt %>%
    pivot_longer(cols = starts_with("SNPindex"),
                 names_to = "phenotype",
                 values_to = "SNP_index") %>%
    mutate(phenotype = ifelse(phenotype == "SNPindex.HIGH", "Late", "Early"))
    ↪  %>%
    mutate(POS_Mb = POS / 1000000)

# Create the plot
p <- ggplot(df_long, aes(x = POS_Mb, y = SNP_index, color = phenotype)) +
    geom_smooth(method = "auto") +
    facet_wrap(~ CHROM, scales = "free_x", nrow = 1) +
    scale_color_manual(values = c("Late" = "skyblue", "Early" = "orange")) +
    labs(x = "Position (Mb)", y = "SNP Index", color = "phenotype") +
    theme(legend.position = c(.9, .85))

### To add gene positions
# Data frame containing gene positions (middle of the gene), chromosomes, and
↪  names. You can download it from csv directly
markers <- data.frame(
  CHROM = c("4", "4", "4", "4", "4", "4", "5", "5", "5", "5"),
  POS_Mb = c(0.2702020, 1.1259825, 1.2412465, 5.7254595, 9.0143655, 9.1975435,
  ↪  3.1764150, 5.1716235, 5.3453005, 5.8284525),
  name = c("FRI", "LD", "GA1", "CRY1", "ESD4", "PHYD", "FLC", "CO", "FRL1",
  ↪  "TFL2"),
  line_number = 1:10 #the number of genes
)

# Add vertical lines and labels to the plot
p +
  geom_vline(data = markers, aes(xintercept = POS_Mb, linetype =
  ↪  as.factor(line_number)), color = "black") +
  geom_label(data = markers, aes(x = POS_Mb, y = max(df_long$SNP_index), vjust
  ↪  = c(0, 1.1, 2.2, 0, 0, 1.1, 0, 1.1, 2.2, 3.3), label = line_number),
             label.padding = unit(0.1, "lines"), size = 2.5,
             fill = "white", color = "black") +
  scale_linetype_manual(values = rep("dashed", 10), labels = paste(1:10,
  ↪  markers$name, sep = ": "), name = "Markers") +
  guides(linetype = guide_legend(override.aes = list(color = "black"))) +
  facet_wrap(~ CHROM, scales = "free_x", nrow = 1) +
  theme(legend.position = "right")

### To extract data from the plot
# Build the plot to extract the data
plot_data <- ggplot_build(p)

# Extract the smoothed data
smooth_data <- plot_data$data[[1]]

# Specify the x value and chromosome you are interested in
x_value <- 8.3573750 # replace with your specific x value
chromosome <- "5" # replace with your specific chromosome value

# Filter the smoothed data for the specific x value and chromosome
smoothed_values <- smooth_data %>%
  filter(PANEL == chromosome) %>%
  group_by(group) %>%
  filter(abs(x - x_value) == min(abs(x - x_value))) %>%
  ungroup()

# Display the smoothed values
print(smoothed_values)
```

Delta SNP :

```
plotQTLStats(
    SNPset = df_filt,
    var = "deltaSNP",
    plotIntervals = TRUE)
```

G prime :

```
plotQTLStats(
    SNPset = df_filt,
    var = "Gprime",
    plotThreshold = TRUE,
    q = 0.01
)
```

p-value :

```
plotQTLStats(
    SNPset = df_filt,
    var = "negLog10Pval",
    plotThreshold = TRUE,
    q = 0.01
)
```

9.5 Export QTLs information into a csv

```
setwd("~/QTL_analysis/R_analysis")

#export summary CSV

getQTLTable(
    SNPset = df_filt,
    alpha = 0.01,
    export = TRUE,
    fileName = "MIL2-KAD1_F2_QTLs.csv"
)
```

10 QTL analysis

10.1 Getting all candidate genes inside the QTL

With the QTLseqR package, you can download a csv file with chromosome name, start and end of your QTLs. You have to delete the column names and make sure that your first three columns have chromosome name start and end (*beware : the chromosome names must be the same as in your annotation file*). You can then convert your csv file to a bed file :

```
cat MIL2-KAD1_F2_QTLs_norownames.csv | tr '[,]' '[\t]' >
 ↪  MIL2-KAD1_F2_QTLs_norownames.bed
```

You can then intersect your bed file to the annotation file. To download the annotation file, you can go on this site for *Arabidopsis thaliana*. You have to download the gff file. Then you have to install bedtools. For Mac users, it can be done with :

```
brew tap homebrew/science
brew install bedtools
```

You can then perform the intersection :

```
intersectBed -a MIL2-KAD1_F2_QTLs_norownames.bed -b
 ↪  Arabidopsis_thaliana.gff -wa -wb >
 ↪  MIL2-KAD1_F2_QTLs_norownames_annotation_intersect.bed
```

And then you can convert the file back to csv format :

```
bedtools sort -i MIL2-KAD1_F2_QTLs_norownames_annotation_intersect.bed | tr
 ↪  '\t' ';' > MIL2-KAD1_F2_QTLs_norownames_annotation_intersect.csv
```

10.2 Filtering to get only genes that are linked with a specific phenotype

The following was done using a python script. I wanted to select the useful information from my csv and filter to keep only genes linked with flowering time :

```
import pandas as pd
import os

os.chdir("~/QTL_analysis/Data/QTL_F2")

FT_genes = ['PHYA', 'CRY1', 'CRY2', 'PHYB', 'PHYD', 'PHYE', 'PFT1', 'FKF1',
↪  'GI', 'CO', 'CDF1', 'SPY', 'LD', 'FVE', 'FLK', 'FRL1', 'FRL2', 'FES1',
↪  'FRI', 'HUA2', 'HOS1', 'PIE1', 'ESD4', 'ELF5', 'FLC', 'VIN3', 'VIN3-L',
↪  'TFL2', 'SVP', 'GA1', 'GA', 'GID1', 'RGL1', 'RGL2', 'SLY1', 'FPF1', 'GAI',
↪  'RGA', 'ATMYB33', 'EBS', 'AGL24', 'SOC1', 'FT', 'FD', 'FDP', 'TSF', 'TFL1',
↪  'ATC', 'BFT', 'MFT', 'AP1', 'CAL', 'LFY', 'AP3', 'PI']

# Read the DataFrame from the CSV file
dfIntersect =
↪  pd.read_csv("MIL2-KAD1_F2_QTLs_norownames_annotation_intersect_test.csv",
↪  sep=";", dtype=str)

# Create a mapping dictionary for chromosome names
chromosome_mapping = {
    'CP002684.1': '1',
    'CP002685.1': '2',
    'CP002686.1': '3',
    'CP002687.1': '4',
    'CP002688.1': '5'
}

# Filter for rows where 'type' is 'gene'
exon_df = dfIntersect[dfIntersect['type'] == 'gene']

# Filter rows where the column "Unnamed: 14" starts with 'Note='
filtered_df = exon_df[exon_df['Unnamed: 14'].str.startswith('Note=', na=False)]

# Extract the gene and locus tag information from the filtered DataFrame
locus_tags = filtered_df['Unnamed: 11'].str.replace('ID=gene-', '')
genes = filtered_df['Unnamed: 13'].str.replace('Name=', '')
notes = filtered_df['Unnamed: 14'].str.replace('Note=', '')

# Extract the chromosome information
chromosomes = filtered_df['chrom'].map(chromosome_mapping)

# Create a DataFrame with genes, locus tags, and chromosome names
df_genes = pd.DataFrame({
    'Gene': genes,
    'Chromosome': chromosomes,
    'Locus_Tag': locus_tags,
    'Start_Position': filtered_df['pos_start'],
    'End_Position': filtered_df['pos_end'],
    'Note': notes
})
```

```
# Select rows where "Note" contains "flower" or the gene is known to be linked
↪  with FT
df_genes_flower = df_genes[(df_genes['Note'].str.contains('flower', case=False,
↪  na=False)) | (df_genes['Gene'].isin(FT_genes))]

# Export the DataFrame to a CSV file
df_genes_flower.to_csv('MIL2-KAD1_F2_QTL_FT_genes.csv', index=False)
```

This gives you the list of all the genes that are comprised in the QTLs and that are linked with the phenotype under study. Theses genes can then be analysed in IGV.

10.3 IGV - DNA sequences comparison

IGV can be used to compare sequences on specific genes. To find genes, you can use the research bar on the TAIR website (https ://www.arabidopsis.org/search/genes).

10.3.1 From VCF files

You can upload sequences from vcf files directly. This allows from quick SNP comparison.
For parents lines, I have selected MIL2 and KAD1 lines using gatk :

```
gatk SelectVariants -V
↪  CologneLines12_noindel_onlyvar_minqual25_minDP5.recode.vcf.gz -sn
↪  A006200066_116048_S97_L004 -sn A006200066_116050_S98_L004 -O
↪  MIL2_and_KAD1_noindel_onlyvar_minqual25_minDP5.recode.vcf.gz
```

Beware, you have to index vcf files :

```
gatk IndexFeatureFile -I
↪  MIL2_and_KAD1_noindel_onlyvar_minqual25_minDP5.recode.vcf.gz
```

If you want to change the samples' names :

```
bcftools reheader -s samples.txt -o new.bcf old.bcf
```

If you want the list of the samples' names :

```
bcftools query -l file.bcf
```

10.3.2 From BAM files

This is to compare sequences in full. Can be used to look at a specific gene.
You have to index your bam files first, using :

```
samtools index ~/QTL_analysis/mapping_bwa/MIL2-KAD1_E_sorted.bam
```

Then you can upload them into IGV. You may need to change the chromosome names.

11 Determining PCR primer sequence

I have found a deletion on the frigida gene that we want to use as target sequence for a PCR to segregate genotypes. It is possible to separate genotypes using 10% difference between the two sequences (wild type and mutant with deletion). PCR sequences used must be around 200bp (so 170bp with a 30bp deletion). In order to get primer sequences, we will use primer3. We will first get DNA consensus sequences from MIL2 and KAD1 (see 7.1).

You can then extract sequences from the gene you want to target :

```
samtools faidx KAD1_consensus.fa 4:268901-271503 > KAD1_FRI.fa
```

In order to use primer3, we need to add brackets to the deletion sequence to define it as target sequence for the PCR. You can either do it by hand or use a python script. You will need to run :

```
pip install biopython
```

Then run the following script. The parent line that you want to use is the one that doesn't have to deletion (or that has the insertion in that case).

```
from Bio import SeqIO
import os

os.chdir("~/QTL_analysis/filtered_bam/")

# Input and output file paths
input_fasta = "MIL2_FRI.fa"
output_fasta = "MIL2_FRI_brackets.fa"

# Define the positions and range
chromosome = "4"
start_range = 268901
end_range = 271503
bracket_start = 270693
bracket_end = 270723

# Read the FASTA file
with open(output_fasta, "w") as output_file:
    for record in SeqIO.parse(input_fasta, "fasta"):
        sequence = record.seq  # Get the sequence in the specified range

        # Calculate relative positions within the extracted sequence
        relative_start = bracket_start - start_range
        relative_end = bracket_end - start_range + 1  # +1 to include the end
         ↪  position

        # Modify the sequence to include brackets
        modified_sequence = (
            sequence[:relative_start] +
            "[" + sequence[relative_start:relative_end] + "]" +
            sequence[relative_end:]
        )
```

```
# Write to the output file
output_file.write(f">{chromosome}:{start_range}-{end_range}\n")
for i in range(0, len(modified_sequence), 60):  # Format sequence to 60
↪  bases per line
    output_file.write(str(modified_sequence[i:i+60]) + "\n")
```