Analysis of ATAC-seq data

Introduction

ATAC-seq (**A**ssay for **T**ransposase-**A**ccessible **C**hromatin using **seq**uencing) is a technique used to assess chromatin accessibility in a genome-wide manner. Compared with alternatives, such as MNaseq-seq or DNase-seq, ATAC-seq has multiple advantages: it requires significantly less input materials and has much shorter preparation times. In addition, it allows assessing regions of open chromatin and nucleosome positions from the same dataset, whereas the other techniques only allow for one or the other.

The advantages of ATAC-seq are based on the usage of an hyperactive mutant Tn5 transposase. In a process called *tagmentation* (tag + fragmentation), the Tn5 transposases cleaves the DNA double-strand and tags the ends of the cut DNA with sequencing adaptors. This leads to the production of short, double-stranded DNA fragments that have sequencing adaptors on each end. These fragments are then purified, PCR amplified and finally sequenced.

In this module you will learn how prepare ATAC-seq data, going from raw sequencing reads to calling regions of open chromatin. Along the way you will learn how to control the quality of your data and which special considerations are necessary when working with ATAC-seq data.

Practical 1: Pre-processing of reads

The first step in every analysis is controlling the quality of your sequencing data. This will allow you to spot any peculiarities of your data that might give you problems during later analysis steps and allow you to deal with them effectively. In addition, we will trim the sequencing data, removing adapter sequences, low-quality bases and other artefacts that complicate mapping the reads.

1) Create a new directory for your raw sequencing data and copy/symlink the files there

```
mkdir raw_data
cp <path-to-sequencing-data> raw_data
```

2) To avoid accidentally altering or deleting the raw data, remove write permissions from the files

```
chmod a-w raw_data/*
```

For quality control, we will be using `FastQC`. This simple tool will run a number of checks to give you a quick impression of your data and alert you to any problems your data might have.

1) Make sure that `fastqc` is accessible from the command line by running:

```
fastqc --version
```

This step is a simple way to test whether a tool is installed and accessible and important for making your analysis replicable: results can often change slightly between different versions of the same tool. Therefore, keeping track of which version you used for your analysis will allow others to replicate your results!

2) To run `fastqc` on your samples:

```
mkdir -p data/fastqc/untrimmed/
fastqc -t 4 -o data/fastqc/untrimmed/ raw_data/*
```

Using 4 threads in parallel (one file per thread) `fastqc` will produce quality reports for each file and store them in the `data/fastqc/untrimmed/` directory. One file will be an `.html` file, containing some basic information and plots of the quality metrics. The other will be a `.zip` archive containing the data underlying the report.

3) Open a report in your browser and familiarise yourself with the quality checks.
   - What is the sequence length for your data?
   - Is there a specific trend for the per base sequence quality?
   - What about the per base sequence content?
   - Do you reads have high levels of PCR duplicates?
   - Are the read contaminated with adapter sequences?

To remove any problematic basecalls and other sequencing artefacts, we will run `fastp` a very fast trimming tool. This will perform many trimming steps: - Remove low quality reads - Trim low quality base calls from the 3' end of reads - Remove adapter sequences from the 3' end of reads - Trim low-complexity and poly-X basecalls - Correct basecalls of overlapping reads (paired-end only)

4) Run `fastp` for your reads:

```
mkdir -p data/{trimmed,fastp}
fastp --thread 4 --in1 raw_data/${SAMPLE}_1.fastq.gz --in2 raw_data/${SAMPLE}_2.fastq.gz \
    --out1 data/trimmed/${SAMPLE}_1.fastq.gz --out2 data/trimmed/${SAMPLE}_2.fastq.gz \
    -3 --cut_tail_window_size 1 --cut_tail_mean_quality 20 \
    --low_complexity_filter --trim_poly_x --correction --detect_adapter_for_pe \
    --json data/fastp/${SAMPLE}.json --html data/fastp/${SAMPLE}.html
```

5) Control quality after trimming with `fastqc`:

```
mkdir -p data/fastqc/trimmed/
fastqc -t 4 -o data/fastqc/trimmed/ data/trimmed/*
```

Practical 2: Alignment of trimmed reads to the genome

After trimming, we use `Bowtie2` to align the short reads to the genome. While `Bowtie2` offers many settings to fine-tune the alignment process, the default options work very well in almost all situations. We only set a few additional options to restrict the alignments that `Bowtie2` will output: - Limit the fragment size to 2,000bp - Ignore reads that do not align to the genome - Ignore reads that are not properly paired (only 1 read aligns, or reads that align too far from each other)

1) Align reads with `Bowtie2`:

```
mkdir -p data/alignment/${SAMPLE}
bowtie2 -p 4 --time \
    -1 data/trimmed/${SAMPLE}_1.fastq.gz -2 data/trimmed/${SAMPLE}_1.fastq.gz -x <index> \
    --maxins 2000 --no-mixed --no-discordant --no-unal > data/alignment/${SAMPLE}/${SAMPLE}
```

The output of `Bowtie2` are **S**AM files, but we prefer to work with **B**AM files, as they are compressed and therefore take up less space. Converting from SAM to BAM is a straightfoward process during which we can already remove low-quality alignments (this includes multi-mapping reads).

2) Convert from SAM to BAM:

```
samtools view -@ 4 -q 10 -b -o data/alignment/${SAMPLE}/${SAMPLE}.bam data/alignment/${SAMPL
```

Finally, we sort the alignments by their genomic coordinates. Many analyses require only the subset of reads orignating from a specific genomic position, e.g. loading the alignments into a genome browser. If the data is unsorted this would require searching through all alignments everytime we move our viewing window. Instead, if the data is sorted we only need to search until the last alignment in our viewing window.

3) Sort the alignments by genomic coordinates:

```
sambamba sort --threads 4 -m 20G -o data/alignment/${SAMPLE}/${SAMPLE}.sorted.bam data/align
```

To allow even faster access to alignments coming from random regions we need to index our BAM files. As the name suggests, this creates an index between genomic regions and the position of alignments from these regions in the BAM file. This can speed up random access by a lot and is required for many tools to work at all. The index is a separate file with the same name as your BAM file and an added `.bai` ending. As the index contains the correlation between genomic regiona and alignment position in the BAM file, whenever you modify the structure of the BAM file (i.e. remove alignments) you need to re-index.

4) Index the sorted alignments:

```
samtools index -@ 4 data/alignment/${SAMPLE}/${SAMPLE}.sorted.bam data/alignment/${SAMPLE}/$
```

Practical 3: Filtering to remove uniformative and artefactual alignments

After aligning the reads and converting them into BAM files we want to remove reads that are either uninformative or are artefacts of the library preparation and sequencing. Uninformative reads are those coming from the mitochondrial genome which is histone-free and therefore always open and usually not interesting to your research. Artefacts that arise during library preparation are PCR duplicates, as these do not give any additional information and actually bias your results towards regions that PCR-amplify more easily. Another class of duplicates are so-called optical duplicates that arise during sequencing, caused by a single cluster of fragments being interpreted as two or more separate clusters. Both types of duplicates appear the same, as alignments with exactly the same sequence, aligning to exactly the same position.

1) Mark PCR and optical duplicates and then remove them:

```
sambamba markdup --nthreads 4 --show-progress \
    data/alignment/${SAMPLE}/${SAMPLE}.bam data/alignment/${SAMPLE}/${SAMPLE}.dupmarked.bam

sambamba view --nthreads 4 --filter "not duplicate" --format bam \
    --output-filename data/alignment/${SAMPLE}/${SAMPLE}.noDups.bam data/alignment/${SAMPLE}
```

2) Remove mitochondrial alignments by filtering based on the name of the reference sequence:

```
sambamba view --nthreads 4 --filter "ref_name != 'chrM'" --format bam \
    --output-filename data/alignment/${SAMPLE}/${SAMPLE}.noM.bam data/alignment/${SAMPLE}/${
```

Lastly, we will remove alignment from so-called "blacklisted regions". These are regions that show extremly high signal independent of the experimental setup. For the most used model organisms (human, mouse, worm, fly) these can easily be downloaded.

3) Remove alignments coming from blacklisted regions

```
bedtools intersect -abam data/alignment/${SAMPLE}/${SAMPLE}.noM.bam -b blacklist.bed -v \
    > data/alignment/${SAMPLE}/${SAMPLE}.filtered.bam
```

One final step that is sometimes done is to shift the alignments. Due to the mechanisms by which the Tn5 transpose insert the sequencing adaptors the ends of the sequencing fragments are shifted by 9bp. To center the ends of the alignemnts on the center of the Tn5 binding site, reads need to be shifted by 5/-4bp. But for most analysis this nucleotide resolution is not necessary.

Practical 4: Calling regions of open chromatin (and nucleosomes)

One very common question for ATAC-seq data is to identify regions of open chromatin. This translates to finding regions where our reads pile up. In theory, this process is very similar to calling peaks, for example when analysing CUT&RUN data. Because of this, many people use peak callers originally developed for ChIP-seq data, with minor adaptions to make them work on ATAC-seq data. One often used peak caller is `MACS2`.
For this practical we will go a different route and use a tool developed specifically for ATAC-seq data, called `HMMRATAC`. It will take our alignments as input and call not only regions of open chromatin but is also able to detect the flanking nucleosomal regions.

1) Before calling actual regions of open chromatin, it is highly recommended to run a cutoff analysis first. This will allow us to pick optimal values for the 3 cutoffs:
   - `-u, --upper`: Upper limit on fold change range for choosing training sites. Should capture some extremly high enrichment and unusually wide peaks
   - `-l, --lower`: Lower limit on fold change range for choosing training sites. Should capture moderate number of peaks with normal width
   - `-c, --prescan-cutoff`: Fold change cutoff for prescanning candidate regions in the whole dataset. Should capture large number of possible peaks with normal width

```
mkdir data/peaks/${SAMPLE}
macs3 hmmratac -i data/alignment/${SAMPLE}/${SAMPLE}.filtered.bam -n ${SAMPLE} --outdir data
```

Once the cutoff analysis is done, have a look at the report to determine the optimal values for `-l`, `-u` and `-c`.

2) Run `HMMRATAC` on your alignments, using the cutoffs determined:

```
macs3 hmmratac -i data/alignment/${SAMPLE}/${SAMPLE}.filtered.bam -n ${SAMPLE} --outdir data
```

Practical 5: Quality control of alignments

With our alignments being filtered, we can now assess some additional quality metrics. This includes things like: - Number of non-duplicate, non-mitochondrial aligned fragments - Alignment rate - Library complexity - Correlation of replicates - Fragment size distribution - TSS enrichment

Whereas the first 4 metrics are useful to check for most sequencing experiments, the last 2 metrics (fragment size distribution and TSS enrichment) are especially important to check.

1) Count the number of sequenced fragments and alignments:

```
# The number of sequenced fragments is equal to the number of lines in the fastq divided by
zcat raw_data/*_1.fastq.gz | wc -l

zcat data/trimmed/*_1.fastq.gz | wc -l

# We can easily count the number of alignments with `sambamba`
# Directly after alignment
sambamba view -c data/alignment/${SAMPLE}/${SAMPLE}.bam

# After removing duplicates
sambamba view -c data/alignment/${SAMPLE}/${SAMPLE}.noDups.bam

# After removing mitochondrial reads
sambamba view -c data/alignment/${SAMPLE}/${SAMPLE}.noM.bam

# After removing alignments from blacklisted regions
sambamba view -c data/alignment/${SAMPLE}/${SAMPLE}.filtered.bam
```

From the number of alignments we can calculate the library complexity as the fraction of distinct (non-duplicated) alignments / total alignments. The alignment rate will be part of the output from `Bowtie2`.

2) Estimate the correlation between replicates by calculating the pairwise correlation between samples:

```
multiBamSummary -p 4 bins --samFlagExclude 1024 data/alignment/*/*.filtered.bam -out data/ba

mkdir results

plotCorrelation -in data/bamsummary.npz --corMethod pearson --skipZeros --whatToPlot scatter
```

There are a range of tools that allow you to easily assess the fragment size distribution and TSS enrichment. For this practical we will use `ataqv`, which can produce interactive HTML report you can view in your browser.

3) Create `JSON` reports for each sample with `ataqv`:

```
mkdir -p data/ataqv

ataqv --peak-file data/peaks/${SAMPLE}/${SAMPLE}.gappedPeaks \
    --name ${SAMPLE} \
    --metrics-file data/ataqv/${SAMPLE}.ataqv.json.gz \
    --tss-file ${TSSFILE} \
    mouse \
    data/alignment/${SAMPLE}/${SAMPLE}.dupmarked.bam
```

4) Combine all `.json` reports into the `.html` report:

```
mkarv results/ataqv_report data/ataqv/*
```

5) Open the interactive report in your browser:

```
xdg-open results/ataqv_report/index.html
```

Practical 6: Converting alignments into coverage tracks for visualization

With all basic analysis steps done and after having checked the quality of our data we can finally have a look at our data in a genome browser. For this we could use the final BAM files, but this has 2 big drawbacks: - BAM files tend to contain a lot of information not needed for visualising the signal, making the files quite big - Signal strength directly depends on sequencing depth, making comparisons between samples difficult

To solve these problems we can convert our alignments into simpler coverage tracks that only contain information on how strong the signal is at a specific point in the genome. In addition, this allows us to scale the signal based on the sequencing depth, making samples more comparable.
For this step we will use one of `Deeptools` many useful tools: `bamCoverage`.

1) Convert your `BAM` files into `bigWig` files:

```
mkdir -p data/coverage

bamCoverage -p 4 --binSize 5 \
    --normalizeUsing CPM \
    --ignoreForNormalization chrX chrY \
    --bam ${SAMPLE}.bam -o data/coverage/${SAMPLE}.all.bw
```

This will create coverage tracks of all alignments, summarising the signal into bins of 50bp and normalising it using **C**ounts **P**er **M**illion (CPM). With this normalisation strategy, a signal of 1 means you would expect to find 1 alignment at this position for every 1 Million alignments in your sample.

But as you remember, depending on the size of the sequencing fragment it contains different information, e.g. whether we have a nucleosome free region (NFR) or mono-nucleosomes. Luckily `bamCoverage` has options to filter alignments by size before creating the coverage tracks. With those, we can create coverage tracks for just the NFR or mono-nucleosomal alignments.

2) Create coverage tracks for the NFR (fragment length < 147bp):

```
bamCoverage -p 4 --binSize 5 \
    --normalizeUsing CPM \
    --ignoreForNormalization chrX chrY \
    --extendReads \
    --maxFragmentLength 135 \
    --bam ${SAMPLE}.bam -o data/coverage/${SAMPLE}.nfr.bw
```

3) Create coverage tracks for mono-nucleosomal regions (147bp < fragment length < 294 bp):

```
bamCoverage -p 4 --binSize 5 \
    --normalizeUsing CPM \
    --ignoreForNormalization chrX chrY \
    --extendReads \
    --minFragmentLength 150 \
    --maxFragmentLength 290 \
    --bam ${SAMPLE}.bam -o data/coverage/${SAMPLE}.mono.bw
```

4) Visualize your coverage tracks in the genome browser, together with the regions of open chromatin you called before.