BMMC Multi-scale Footprinting Vignette

Yan Hu

10/3/2022

BMMC Footprinting Tutorial

This document shows how to use our framework to run multi-scale footprinting and TF binding prediction. The tutorial data contains data of 10 CRE regions and 1000 pseudobulks, which is a very small subset of our human bone marrow dataset. If you have a large number of CREs and pseudobulks (like in the case of our full analysis with 200k CREs and 1000 pseudo-bulks), consider dividing CRE regions into chunks and running parallel jobs on a computing cluster. For examples, see the runTFBS.sh and TFBSLoop.sh scripts in /analyses/BMMC/.

1. Loading input data required for footprinting

We first source the scripts we need for footprinting

```
# If running in Rstudio, set the working directory to current path
if (Sys.getenv("RSTUDIO") == "1"){
  setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
}

source("../../code/utils.R")
source("../../code/getCounts.R")
source("../../code/getBias.R")
source("../../code/getFootprints.R")
source("../../code/getSubstructures.R")
source("../../code/visualization.R")
source("../../code/getGroupData.R")
source("../../code/footprintTracking.R")
source("../../code/getTFBS.R")
```

Next, we initialize an footprintingProject object. This would be the central object that we use from now on (similar to how you use SeuratObject for Seurat).

Here I need to briefly explain our folder structure. When the project is given a name, such as "BMMCTutorial", by default all data associated with this project will be stored in the ../../data/BMMCTutorial folder. When you want to create another project, say "XXX", with your own data, you need to create ../../data/XXX and and put all your input data (CRE bed files, fragments file, cell barcode grouping, metadata, etc. all in tha) in there. Typically I also like to put all the analyses script associated with the same project into ../../analyses/XXX

```
projectName <- "BMMCTutorial"
project <- footprintingProject(projectName = projectName,
                               refGenome = "hg38")
projectMainDir <- "../../"
projectDataDir <- paste0(projectMainDir, "data/", projectName, "/")
dataDir(project) <- projectDataDir
mainDir(project) <- projectMainDir
```

We load the genomic ranges of the regions we are interested in. Only the fragments in these regions will be used for footprinting.

```
regionsBed <- read.table(paste0(projectDataDir, "BMMCTutorialRegions.bed"), header = T)
regions <- GRanges(seqnames = regionsBed$chr,
                   ranges = IRanges(start = regionsBed$start, end = regionsBed$end))
regionRanges(project) <- regions
```

Make sure you first download our pre-computed whole-genome bias h5 files and put them in the ../../data/shared/precomputedTn5Bias/ directory. The below function loads the pre-computed bias in the regions of interest

```
if(file.exists(paste0(projectDataDir, "predBias.rds"))){
  regionBias(project) <- readRDS(paste0(projectDataDir, "predBias.rds"))
}else{
  project <- getPrecomputedBias(project, nCores = 8)
  saveRDS(regionBias(project), paste0(projectDataDir, "predBias.rds"))
}
```

Now we tell the program which cells belong to which pseudo-bulks. We load the file barcodeGroups.txt. This is a two-column table, where the first column is cell barcodes (should match the barcodes in the 4th column in the fragments file), and the second column is group IDs.

```
pathToFragGrouping <- paste0(projectDataDir, "barcodeGrouping.txt")
barcodeGroups <- read.table(pathToFragGrouping, header = T)
barcodeGrouping(project) <- barcodeGroups
groups(project) <- mixedsort(unique(barcodeGroups$group))
```

We now extract Tn5 insertion positions and re-format the data into a pseudobulk-by-region-by-position counts tensor. This facilitates fast data retrieval and computation for later.

```
pathToFrags <- paste0(projectMainDir, "data/", projectName, "/BMMCTutorialFragments.tsv")
project <- getCountTensor(project, pathToFrags, barcodeGroups, returnCombined = F)
```

```
## [1] "Using chunk size = 2000"
## Make 1bp step ..
## Reading in fragment file ..
## Reading only select barcodes specified within list from file ..
## Reformating counts data into a list (each element is data for a region) ..
## [1] "Re-organizing data into lists"
## [1] "2023-05-14 19:56:15 Processing chunk 1 out of 1 chunks"
## Done!
## Time elapsed:  0.6890538 secs
```

Load pseudo-bulk metadata.

```
groupInfo <- read.table(paste0(projectDataDir, "groupInfo.txt"), header = T,
                        comment.char = "")
groupCellType(project) <- groupInfo$cellType # Please make sure at least you provide this
groupUMAP(project) <- as.matrix(cbind(groupInfo$UMAP1, groupInfo$UMAP2)) # Optional
groupPseudoTime(project) <- groupInfo$Pseudotime # Only needed for object tracking
```

Now we load dispersion models. These models estimate the background dispersion of the footprinting test statistic and are used to compute a p-value when running footprinting on ATAC-Seq data

```
for(kernelSize in 2:100){
  dispModel(project, as.character(kernelSize)) <-
```

```
    readRDS(paste0("../../data/shared/dispModel/dispersionModel", kernelSize, "bp.rds"))
}
```

Also load the TF habitation model. It uses local multi-scale footprints to score suitability for TF binding

```
# Load TFBS prediction model
h5Path <- "../../data/TFBSPrediction/TFBS_model.h5"
TFBindingModel(project) <- loadTFBSModel(h5Path)
```

2. Running footprinting and TF binding/habitation scoring

Now we run footprinting at a specific scale across CREs and pseudo-bulks. Here we are using 50 bp scale, which corresponds to nucleosome footprints. You can change this number to obtain footprint signal at other scales

```
scale <- 50
project <- getFootprints(
  project,
  mode = as.character(scale),
  nCores = 16,
  footprintRadius = scale,
  flankRadius = scale)
```

```
## [1] "Using chunk size = 10"
## Computing footprinting scores of CREs
## Chunking data ..
## [1] "Processing region chunk 1 out of 1 chunks"
## [1] "2023-05-14 19:56:20 UTC"
```

Score TF binding for each CRE and pseudo-bulk

```
project <- getTFBS(project,
        nCores = 16)
```

```
## [1] "Using chunk size = 2000"
## Chunking data ..
## [1] "Processing region chunk 1 out of 1 chunks"
## [1] "2023-05-14 19:56:21 UTC"
```

3. Retrieving results

3.1 Footprinting results Let's take a look at what the results might look like. Since now we are only running on 10 regions, we have only 1 chunk. In practice if you have hundreds of thousands of regions, the results would be stored in multiple chunks. You can find all of them in the below folder.

```
scale <- 50
chunkResults <- readRDS(paste0("../../data/BMMCTutorial/chunkedFootprintResults/",
                               as.character(scale), "/chunk_1.rds"))
```

The results are stored in the form of a list. Each list element is all the results for the same CRE. Let's extract the results for the 10th CRE now.

```
regionFootprints <- chunkResults[[10]]

# Show the data slots stored
names(regionFootprints)
```

```
## [1] "regionID"             "summitPvalscores"     "aggregateATAC"
## [4] "footprintRanges"      "aggregateScores"      "summits"
```

```
## [7] "bias"                    "summitCellTypeScores"
```

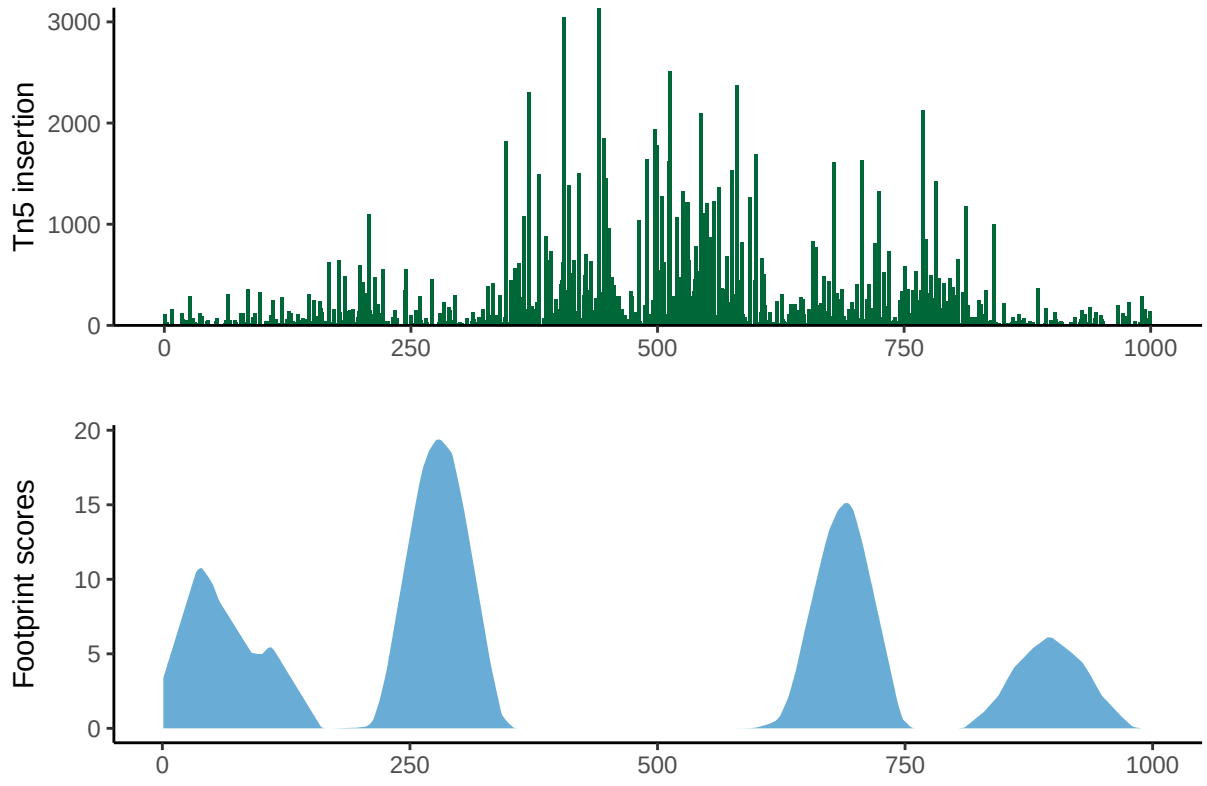
Check region ID, this by default stores the genomic ranges of the regions.

```
regionFootprints$regionID
```

```
## [1] "chr20:56411666-56412665"
```

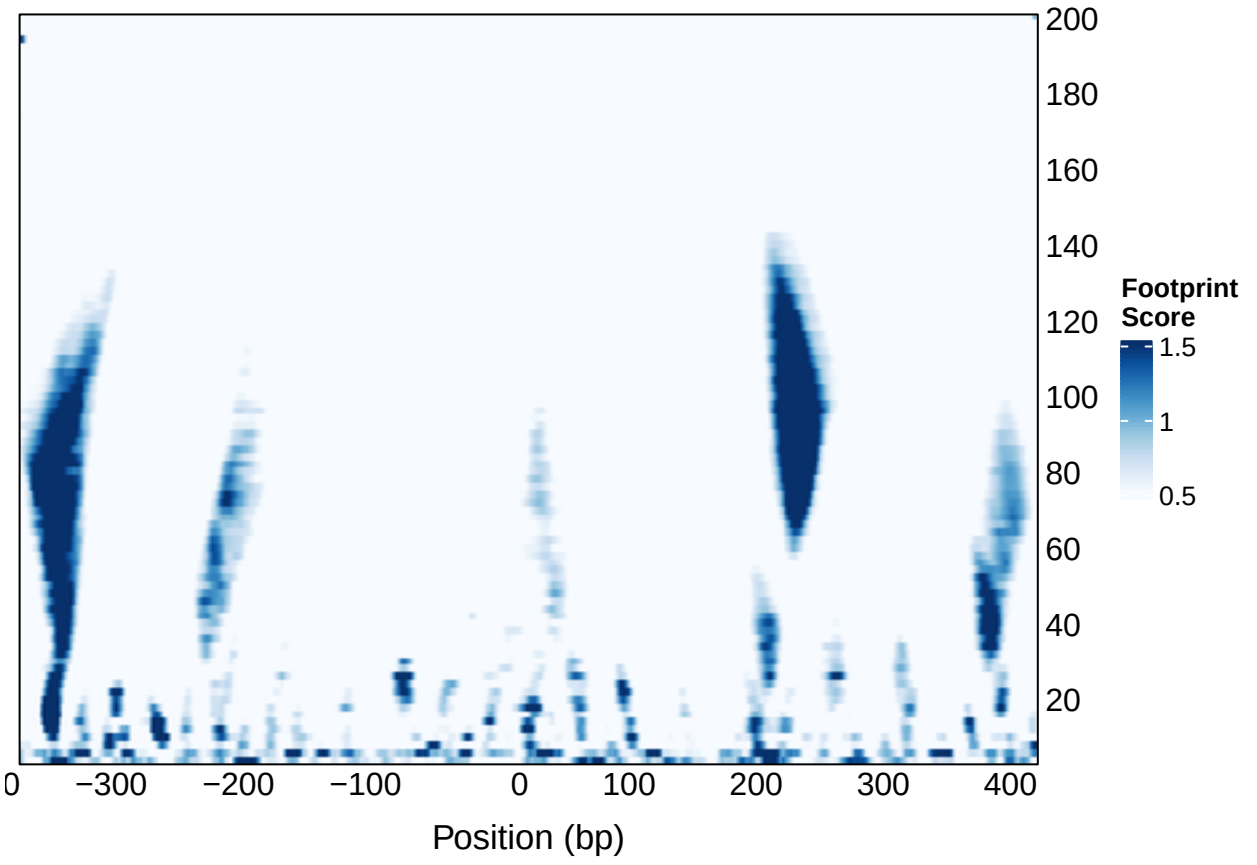
The "aggregateATAC" and "aggregateScores" slots store the Tn5 insertion counts and footprinting scores when aggregating all pseudo-bulks together.

```
positions <- 1:1000
Tn5Insertion = regionFootprints$aggregateATAC
barData <- data.frame(x1 = positions - 2, x2 = positions + 2,
                      y1 = 0, y2 = Tn5Insertion)
p1 <- ggplot(barData) +
  geom_rect(aes(xmin = x1, xmax = x2, ymin = y1, ymax = y2),
            fill = "#006838") +
  scale_y_continuous(expand = c(0, 0)) + xlab("") + ylab("Tn5 insertion") +
  theme_classic()

aggregateScores <- regionFootprints$aggregateScores
plotData <- data.frame(Position = positions,
                       aggregateScores = aggregateScores)
p2 <- ggplot(plotData) +
  geom_ribbon(aes_string(x = "Position", ymin = 0, ymax = "aggregateScores"),
              fill = "#69ACD5") + xlab("") + ylab("Footprint scores") +
  theme_classic()

library(patchwork)
p1 / p2
```



You can also visualize the multi-scale footprints within this region. Note that for the largest scale, we need to see +/- 100 bp for each position for footprinting. As a result, the leftmost and rightmost 100 bp within this 1 kb can't be used for footprinting. We therefore show the center 800 bp in the plot.

```
plotMultiScale(project = project, regionInd = 1, vmax = 1.5)
```



3.2 TFBS results Save as above, we take a look at the results for one chunk for now.

```
chunkResults <- readRDS("../../data/BMMCTutorial/chunkedTFBSResults/chunk_1.rds")
```

The results are stored in the form of a list. Each list element is all the results for the same CRE. Let's extract the results for the 10th CRE now.

```
regionTFBS <- chunkResults[[10]]

# Show the data slots stored
names(regionTFBS)
```

```
## [1] "position"   "region"     "sites"      "TFBSScores"
```

The $region slot stores the genomic range of the corresponding region.

```
regionTFBS$region
```

```
## GRanges object with 1 range and 0 metadata columns:
##       seqnames            ranges strand
##          <Rle>         <IRanges>  <Rle>
##   [1]    chr20 56411666-56412665      *
##   -------
```

```
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

The $sites slot stores the genomic range of the intervals for which TF binding habitation are computed. When we were running getTFBS(), if you provide a PWMatrixList object for the "motifs" parameter, then we will find all motif matched positions within each CRE and score TF binding for these motif sites. In this case $sites will store the motif matched positions. By default, if we don't provide TF motifs to the getTFBS() function, it will automatically divide each CRE region into 10bp bins and calculate TF habitation scores for each bin. Therefore, if you start with a 1kb CRE region and 1000 pseudo-bulks, you are supposed to get back a 100bin-by-1000pseudobulk matrix. However, since the model need input from +/- 100 bp local window, anything within 100 bp to the edge of the CRE will not be able to get a valid prediction result. Therefore we lose 10 bins on each side, and we end up with a 80bin-by-1000pseudobulk matrix.

```
regionTFBS$sites
```

```
## GRanges object with 80 ranges and 2 metadata columns:
##        seqnames            ranges strand |     score          TF
##           <Rle>         <IRanges>  <Rle> | <numeric> <character>
##    [1]    chr20 56411766-56411775      * |         1
##    [2]    chr20 56411776-56411785      * |         1
##    [3]    chr20 56411786-56411795      * |         1
##    [4]    chr20 56411796-56411805      * |         1
##    [5]    chr20 56411806-56411815      * |         1
##    ...      ...               ...    ... .       ...         ...
##   [76]    chr20 56412516-56412525      * |         1
##   [77]    chr20 56412526-56412535      * |         1
##   [78]    chr20 56412536-56412545      * |         1
##   [79]    chr20 56412546-56412555      * |         1
##   [80]    chr20 56412556-56412565      * |         1
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

In fact, you can use the getBindingSE() function to easily retrieve the site-by-pseudobulk SummarizedExperiment object of TF habitation scores

```
TFBindingSE <- getTFBindingSE(project)
print(paste("Shape of object", dim(TFBindingSE)))
```

```
## [1] "Shape of object 800"  "Shape of object 1000"
```

```
siteRanges <- rowRanges(TFBindingSE)
scoreMatrix <- assay(TFBindingSE)
siteRanges
```

```
## GRanges object with 800 ranges and 2 metadata columns:
##         seqnames            ranges strand | regionInd          TF
##            <Rle>         <IRanges>  <Rle> | <integer> <character>
##     [1]    chr20 56358479-56358488      * |         1
##     [2]    chr20 56358489-56358498      * |         1
##     [3]    chr20 56358499-56358508      * |         1
##     [4]    chr20 56358509-56358518      * |         1
##     [5]    chr20 56358519-56358528      * |         1
##     ...      ...               ...    ... .       ...         ...
##   [796]    chr20 56412516-56412525      * |        10
##   [797]    chr20 56412526-56412535      * |        10
##   [798]    chr20 56412536-56412545      * |        10
##   [799]    chr20 56412546-56412555      * |        10
##   [800]    chr20 56412556-56412565      * |        10
```

```
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

4. Visualize example regions

We have implemented visualization functions for easy visualization of specific examples.

First we provide some additional information for plotting

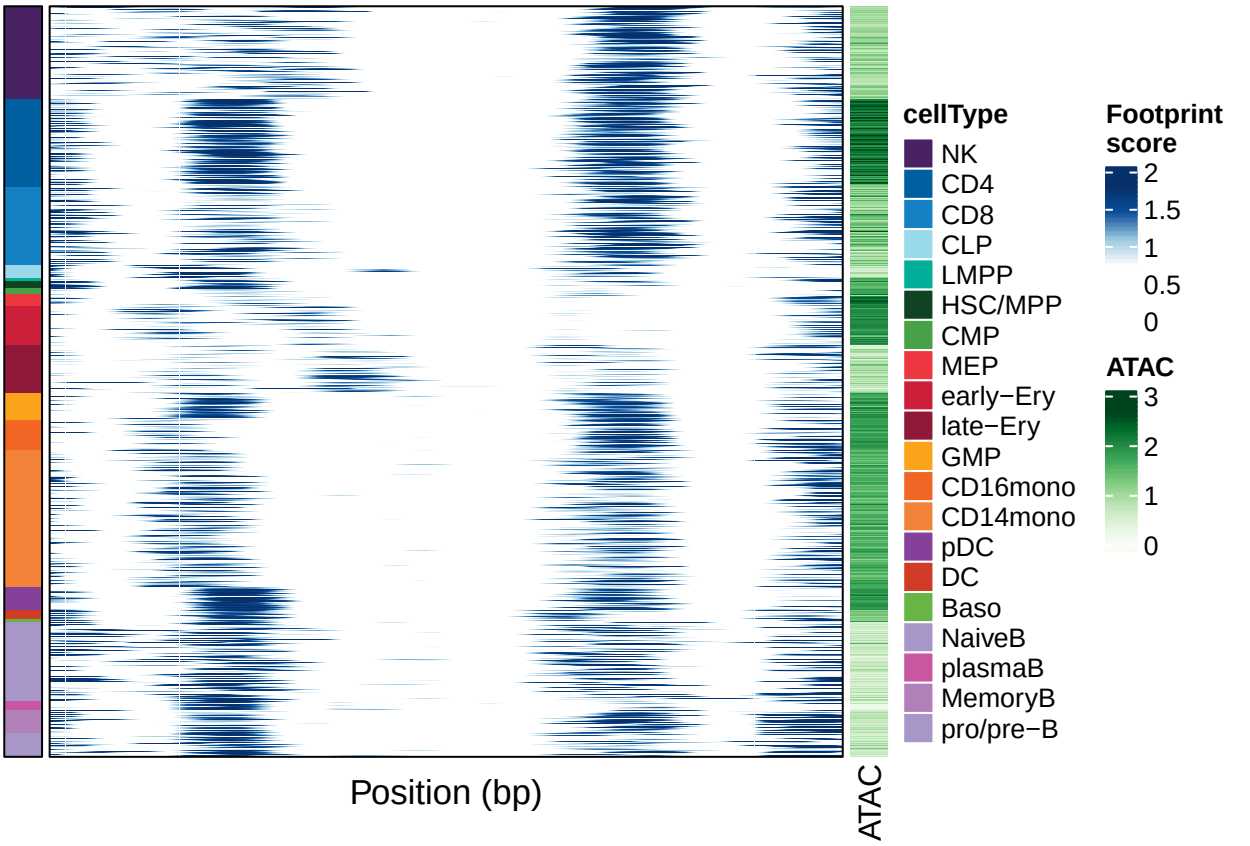
```
cellTypeOrder <- c("NK", "CD4", "CD8",
                   "CLP", "LMPP", "HSC/MPP", "CMP", "MEP", "early-Ery", "late-Ery",
                   "GMP", "CD16mono", "CD14mono", "pDC", "DC", "Baso",
                   "NaiveB", "plasmaB", "MemoryB", "pro/pre-B")
rowOrder <- order(match(groupCellType(project), cellTypeOrder))

# Get color mapping for cell types
cellTypes <- unique(groupInfo$cellType)
cellTypeColors <- groupInfo$color[match(cellTypes, groupInfo$cellType)]
names(cellTypeColors) <- cellTypes
```

We also need to load the region-by-pseudobulk ATAC matrix and gene-by-pseudobulk RNA matrix. The latter is too big to upload to Github but you can find it in our Zenodo repository.

```
# Get region-by-pseudobulk ATAC matrix
pseudobulkATACPath <- paste0(projectDataDir, "pseudobulkATAC.rds")
groupATAC(project) <- readRDS(pseudobulkATACPath)

# Get gene-by-pseudobulk RNA matrix, followed by LogNormalization
pseudobulkRNAPath <- paste0(projectDataDir, "pseudobulkRNA.rds")
groupRNA(project) <- readRDS(pseudobulkRNAPath)
```

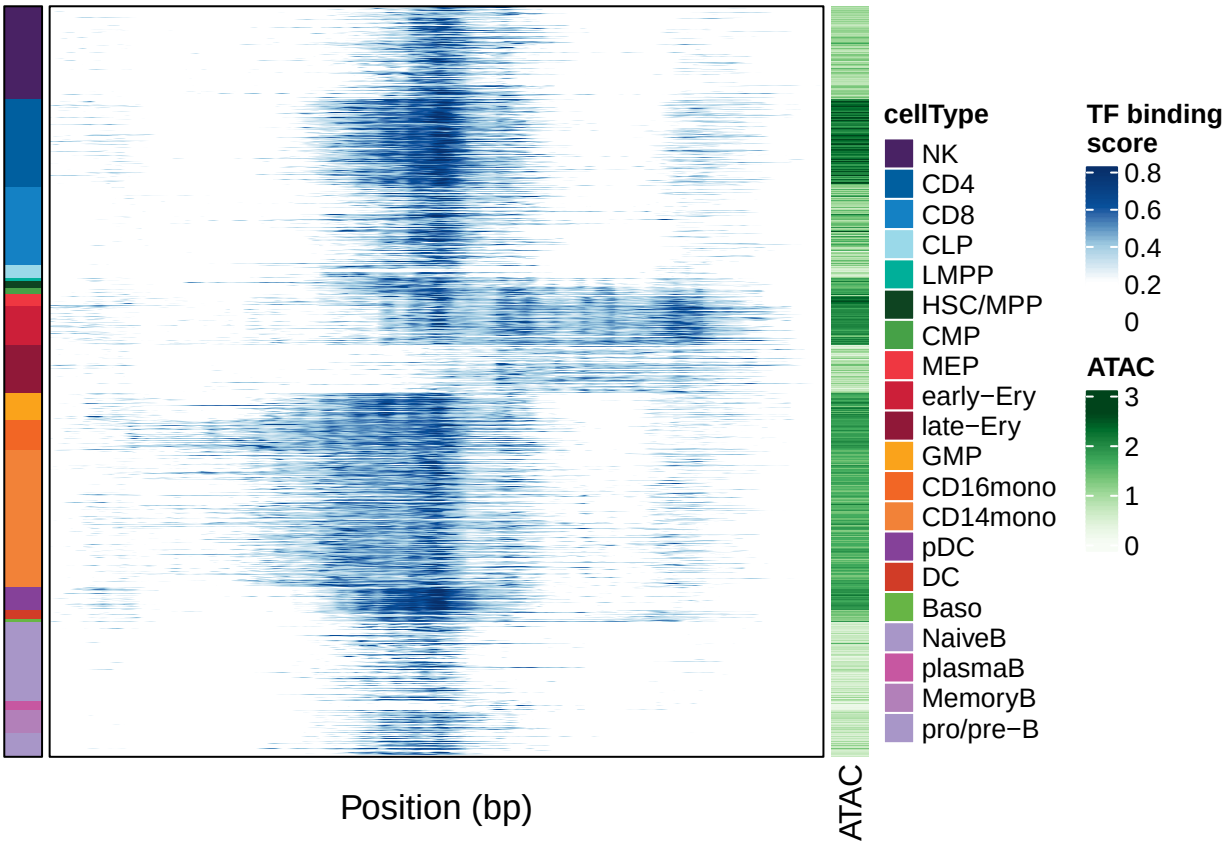
To visualize the footprint score pattern within a single region across pseudo-bulks:

```
regionInd <- 10
plotFeatureHeatmap(project,
                   regionInd = regionInd,
                   feature = "footprint",
                   footprintRadius = 50,
                   cellTypeColors = cellTypeColors,
                   cellTypeOrder = cellTypeOrder,
                   rowOrder = rowOrder)
```



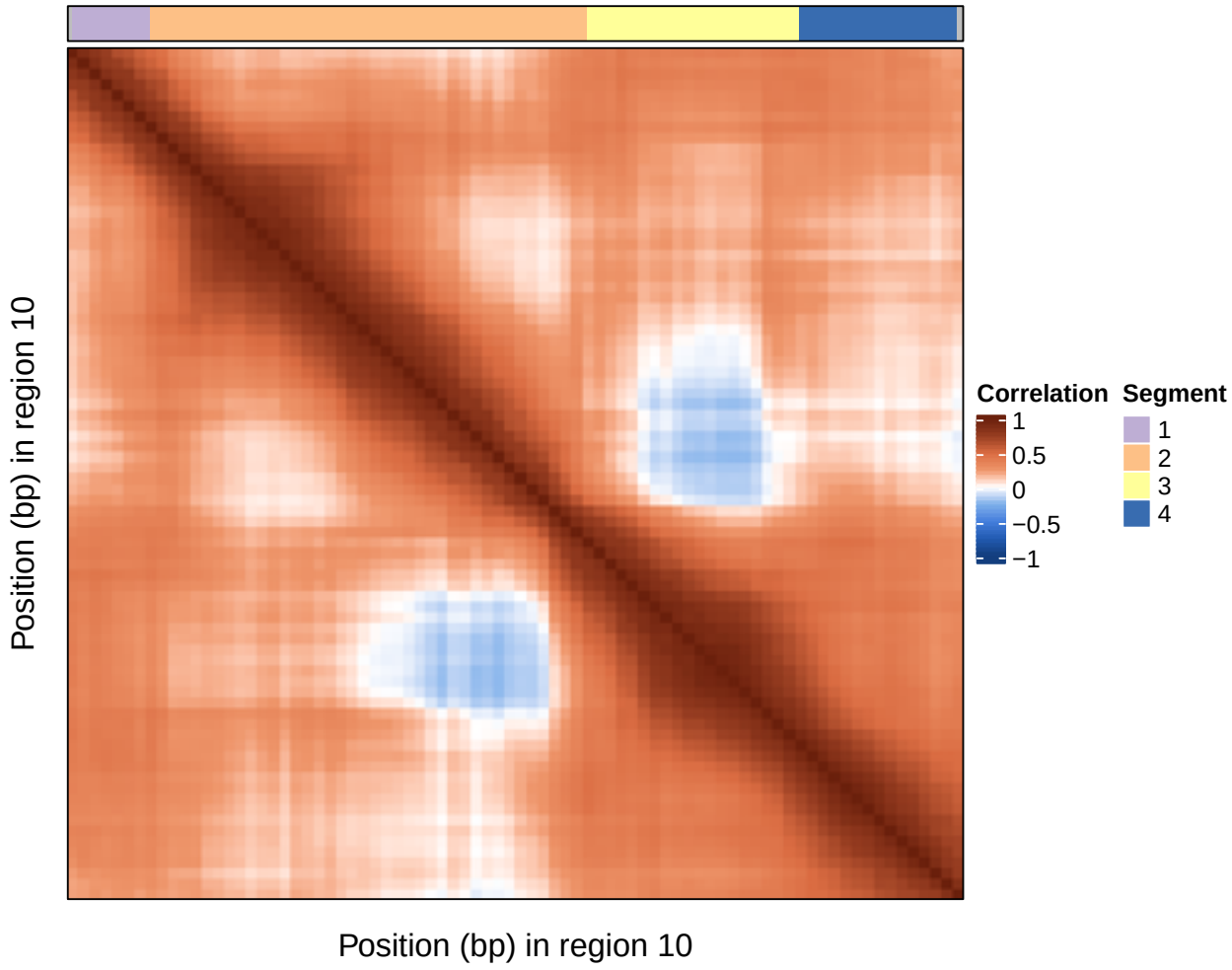
To visualize the TF habitation score pattern within a single region across pseudo-bulks:

```
regionInd <- 10
plotFeatureHeatmap(project,
                   regionInd = regionInd,
                   feature = "TFBS",
                   cellTypeColors = cellTypeColors,
                   cellTypeOrder = cellTypeOrder,
                   rowOrder = rowOrder)
```



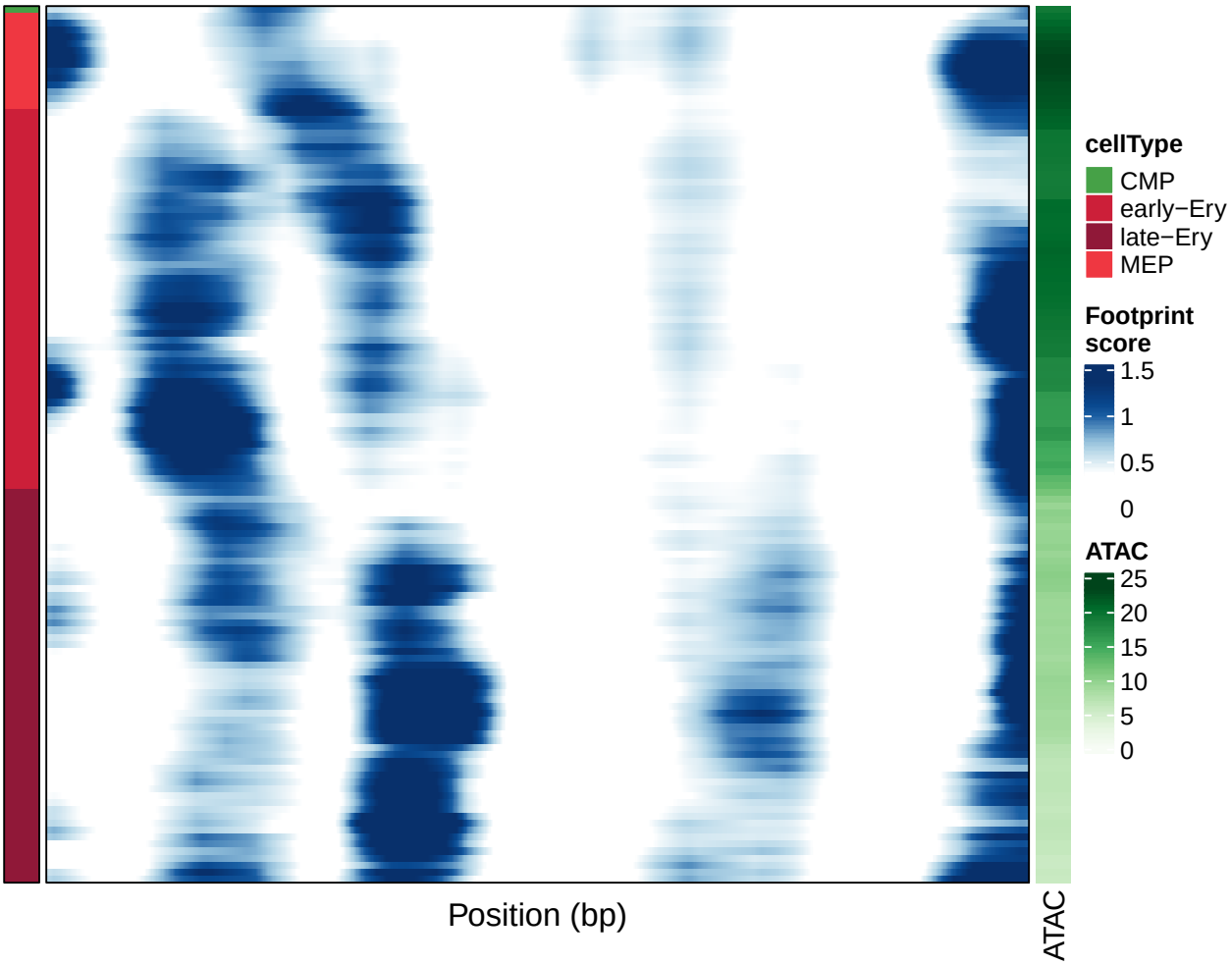
To visualize the TF habitation score correlation map and segmentation results: (Note that the below output only show the bins at least 100bp away from the edge of the region. For details see section 3.2)

```
regionInd <- 10
plotSegmentHeatmap(project,
                   regionInd = regionInd)
```



To visualize footprint movements across pseudotime (make sure you loaded psuedotime from groupInfo.txt as instructed previously):

```
LineageInd <- 2
lineageProbs <- groupInfo[, c("MyeloidProbs", "ErythroidProbs",
                              "BLymphoidProbs", "TLymphoidProbs")]
lineageIdentities <- sapply(1:dim(lineageProbs)[1],
                            function(x){which(lineageProbs[x,] == max(lineageProbs[x,]))})
lineageGroups <- which(lineageIdentities %in% LineageInd)
regionInd <- 10
plotPseudotimeHeatmap(
  project,
  regionInd = regionInd,
  feature = "footprint",
  lineageGroups = lineageGroups,
  cellTypeColors = cellTypeColors,
  heatmapPalette = colorRampPalette(c("white",  "#9ECAE1", "#08519C", "#08306B"))(9),
  footprintRadius = 50
)
```


cellType
CMP
early−Ery
late−Ery
MEP
Footprint score
1.5
1
0.5
0
ATAC
25
20
15
10
5
0
Position (bp)
ATAC