Peak co-accessibility - Cicero

Ivan Berest and Christian Arnold

Contents

Goals

This vignette will introduce you to `Cicero`, an R package that provides tools for analyzing single-cell chromatin accessibility experiments such as scATAC-seq data. Its main function is to use single-cell chromatin accessibility data to predict cis-regulatory interactions (such as those between enhancers and promoters) in the genome by examining co-accessibility. In addition, we will touch upon constructing trajectories with accessibility data and differential accessibility analyses.

References

This vignette is a modified version of this vignette and this vignette.

Load libraries

As before, we first load all necessary R packages that are needed for this vignette.

```
suppressPackageStartupMessages({
library(Signac)
library(Seurat)
library(SeuratWrappers)
library(monocle3)
library(tidyverse)
library(patchwork)
library(cicero)
})
set.seed(1990)
```

Load data

We begin by loading our pre-processed `Seurat` object from the introductory QC vignette into R. This code is shared among all subsequent vignettes.

```
# Specify the path to the "outs" folder from cellranger-atac here
# Make sure to have a trailing slash here
outFolder="/mnt/data/cellranger/outs/"
outFolder="/g/scb2/zaugg/zaugg_shared/Courses_and_Teaching/ATAC-Seq_Course/2022/data/testRun_March/coun
seu.s = readRDS(file = paste0(outFolder,"obj.filt.rds"))

# Let's make sure we are working with correct clusters
Idents(seu.s)  = seu.s$ATAC_snn_res.0.5
DefaultAssay(seu.s) ="ATAC"
```

Predict cis-regulatory interactions

`Cicero` is an R package that provides tools for analyzing single-cell chromatin accessibility experiments such as scATAC-seq data. Its main function is to use single-cell chromatin accessibility data to predict cis-regulatory interactions (such as those between enhancers and promoters) in the genome by examining co-accessibility.

Let's now try to find cis-co-accessible networks (CCANs)! As always, we need some pre-processing first.

Convert objects

The `Cicero` developers have developed a separate branch of the package that works with a Monocle 3 `cell_data_set` object.

```
cds.obj <- as.cell_data_set(seu.s)
```

```
## Warning: Monocle 3 trajectories require cluster partitions, which Seurat does
## not calculate. Please run 'cluster_cells' on your cell_data_set object
```

```
# takes a few minutes
cicero.obj <- make_cicero_cds(cds.obj, reduced_coordinates = reducedDims(cds.obj)$UMAP)
```

```
## Overlap QC metrics:
## Cells per bin: 50
## Maximum shared cells bin-bin: 44
## Mean shared cells bin-bin: 0.75993846652093
## Median shared cells bin-bin: 0
```

```
rm(cds.obj)
```

The warning we get we can safely ignore here, as we do not run `Monocle` trajectories here in this vignette.

Parameters

The default parameters were designed for use in human and mouse. Importantly, there are a few parameters that we expect will need to be changed for use in different model organisms. For more details, see here. Since we have mouse data, there is no immediate need to change any of the parameters at first, but as always, upon inspection of the results, we may want to tweak some parameters to obtain better results! First, let's run it with the default parameters, however.

Find Cicero connections

Here we demonstrate the most basic workflow for running `Cicero`. Its main function is to estimate the co-accessiblity of sites in the genome in order to predict cis-regulatory interactions. There are two ways to get this information:

- `run_cicero`: get Cicero outputs with all defaults The function `run_cicero` will call each of the relevant pieces of `Cicero` code using default values, and calculating best-estimate parameters as it goes. For most users, this will be the best place to start, and this is also our first approach.
- Call functions separately for users wanting more flexibility in the parameters that are called, and those that want access to intermediate information.

As you can see, the workflow can be broken down into several steps, each with parameters that can be changed from their defaults to fine-tune the `Cicero` algorithm depending on your data (if needed). As with all tools, we highly recommend that you explore the `Cicero` website, paper, and documentation for more information (see the last section for more references).

Here, for the purpose of this vignette, in order to achieve shorter running times, we here employ 2 tricks:

- Running this not for the full genome but only one (part of a) chromosome (*chr19*, one of the smallest mouse chromosomes, and only 10 million bases)
- Reduce the number of samples from 100 to a smaller value

```
# How many sample genomic windows to use to generate distance_parameter parameter. Default: 100.
# Normally set to 100, here smaller so it runs faster - set to a higher value if possible
sampleNum = 20

# get the chromosome sizes from the Seurat object
genome<- seqlengths(seu.s)

# use smallest mouse chromosome 19 to save some time
# omit this step to run on the whole genome
genome <-genome["chr19"]

# convert chromosome sizes to a data frame.
# Here we set the chromosome size to a small value and not even the full chromosome
# Set length to genome to run it for the full chromosome
genome.df <- data.frame("chr" = names(genome), "length" = 10000000)

# Run cicero. This may take a lot of time
conns <- run_cicero(cicero.obj, genomic_coords = genome.df, sample_num = sampleNum)
```

```
## [1] "Starting Cicero"
## [1] "Calculating distance_parameter value"
## [1] "Running models"
## [1] "Assembling connections"
## [1] "Successful cicero models:  29"
## [1] "Other models: "
##
## Zero or one element in range
##                           11
## [1] "Models with errors:  0"
## [1] "Done"
```

```
rm(cicero.obj)

# Let's look into the results, sort by co-accesibility first
arrange(conns, desc(coaccess)) %>% head(10)
```

```
##                    Peak1                 Peak2  coaccess
## 1  chr19-8982920-8983671 chr19-9106697-9107655 0.7450070
## 2  chr19-9106697-9107655 chr19-8982920-8983671 0.7450070
```

```
## 3  chr19-8982920-8983671 chr19-8984249-8985101 0.6991165
## 4  chr19-8984249-8985101 chr19-8982920-8983671 0.6991165
## 5  chr19-7261207-7262197 chr19-7583128-7583920 0.6489091
## 6  chr19-7583128-7583920 chr19-7261207-7262197 0.6489091
## 7  chr19-7369237-7370115 chr19-7583128-7583920 0.6406417
## 8  chr19-7583128-7583920 chr19-7369237-7370115 0.6406417
## 9  chr19-8982920-8983671 chr19-9073544-9074433 0.6313957
## 10 chr19-9073544-9074433 chr19-8982920-8983671 0.6313957
```

If the time allows, the same workflow should be applied to find *CCANs* for the whole genome by not restricting the chromosome as we did above.

Find cis-co-accessible networks (CCANs)

In addition to pairwise co-accessibility scores, Cicero also has a function to find *Cis-Co-accessibility Networks* (CCANs), which are modules of sites that are highly co-accessible with one another. We use the *Louvain community detection algorithm* (Blondel et al., 2008) to find clusters of sites that tended to be co-accessible. The function `generate_ccans` takes as input a connection data frame and outputs a data frame with CCAN assignments for each input peak. Sites not included in the output data frame were not assigned a CCAN.

Now that we have found pairwise co-accessibility scores for each peak, let's group these pairwise connections into larger co-accessible networks!

```
# generate co-accessibility networks
ccans <- generate_ccans(conns)
```

```
## [1] "Coaccessibility cutoff used: 0.09"
```

```
# Let's look into the results
ccans[1:20,]
```

```
##                                          Peak CCAN
## chr19-10012680-10013593 chr19-10012680-10013593    1
## chr19-10017735-10018619 chr19-10017735-10018619    1
## chr19-10019213-10019868 chr19-10019213-10019868    1
## chr19-10041117-10041977 chr19-10041117-10041977    1
## chr19-10059593-10060617 chr19-10059593-10060617    1
## chr19-10129317-10130181 chr19-10129317-10130181    1
## chr19-10137231-10138254 chr19-10137231-10138254    1
## chr19-10157469-10158389 chr19-10157469-10158389    1
## chr19-10169429-10170320 chr19-10169429-10170320    1
## chr19-10203575-10204482 chr19-10203575-10204482    1
## chr19-10222356-10223230 chr19-10222356-10223230    1
## chr19-10223399-10224167 chr19-10223399-10224167    1
## chr19-10224564-10225670 chr19-10224564-10225670    1
## chr19-10230023-10230709 chr19-10230023-10230709    1
## chr19-10231594-10232505 chr19-10231594-10232505    1
## chr19-10242723-10243534 chr19-10242723-10243534    1
## chr19-3282549-3283455     chr19-3282549-3283455    3
## chr19-3322909-3323753     chr19-3322909-3323753    3
## chr19-3325058-3325875     chr19-3325058-3325875    4
## chr19-3388456-3389256     chr19-3388456-3389256    3
```

Add links to a Seurat object

We can add the co-accessible links found by `Cicero` to the `ChromatinAssay` object in `Seurat`. Using the `ConnectionsToLinks()` function in `Signac` we can convert the outputs of `Cicero` to the format needed to

store in the links slot in the `ChromatinAssay`, and add this to the object using the `Links<-` assignment function.

```
# transform to the links
links <- ConnectionsToLinks(conns = conns, ccans = ccans)
Links(seu.s) <- links
```

Visualize links

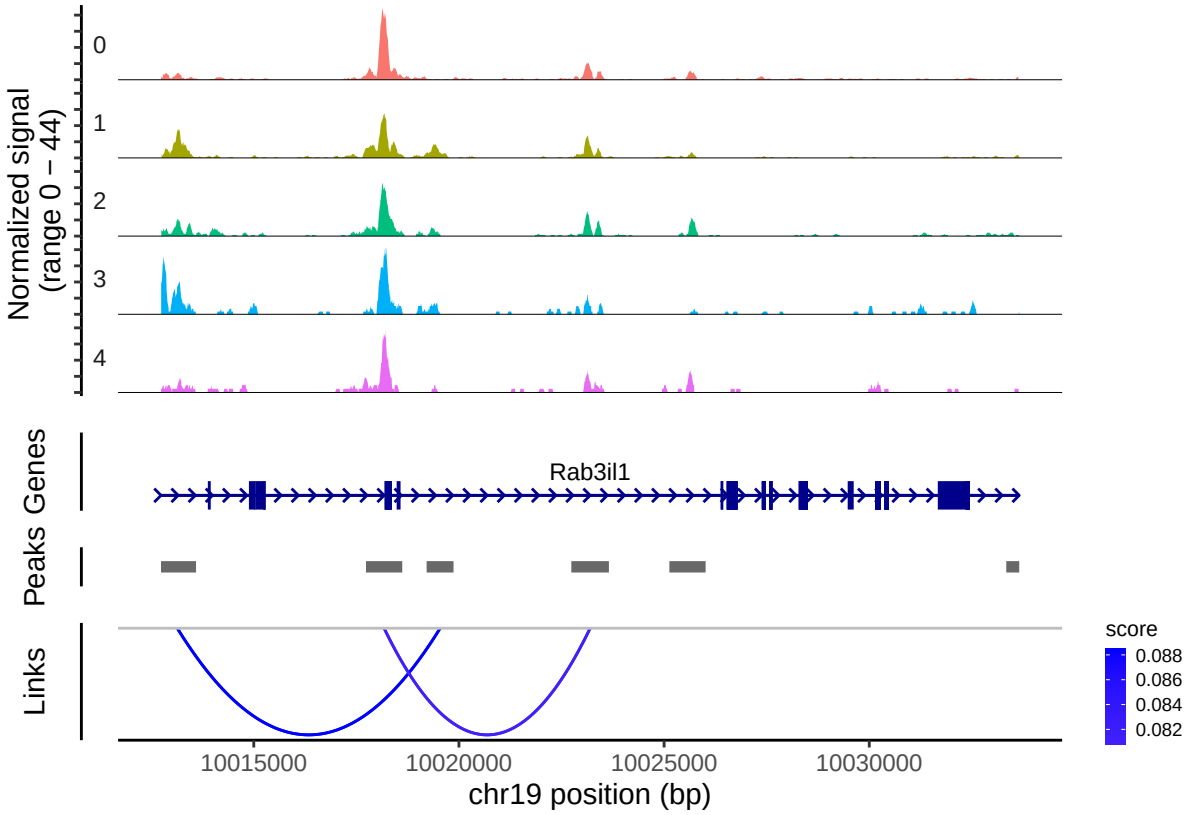
We can now visualize these links along with DNA accessibility information by running `CoveragePlot()` for one particular, customizable region:

```
# coverage plot for the specific region
region = "chr19-10022739-10023655"

# in principle here we can also show links network close to some interesting gene on chr19
CoveragePlot(seu.s, region = region, extend.upstream = 10000, extend.downstream = 10000)
```

```
## Warning: Removed 15 rows containing missing values (geom_segment).
```

```
## Warning: Removed 1 rows containing missing values (geom_segment).
```



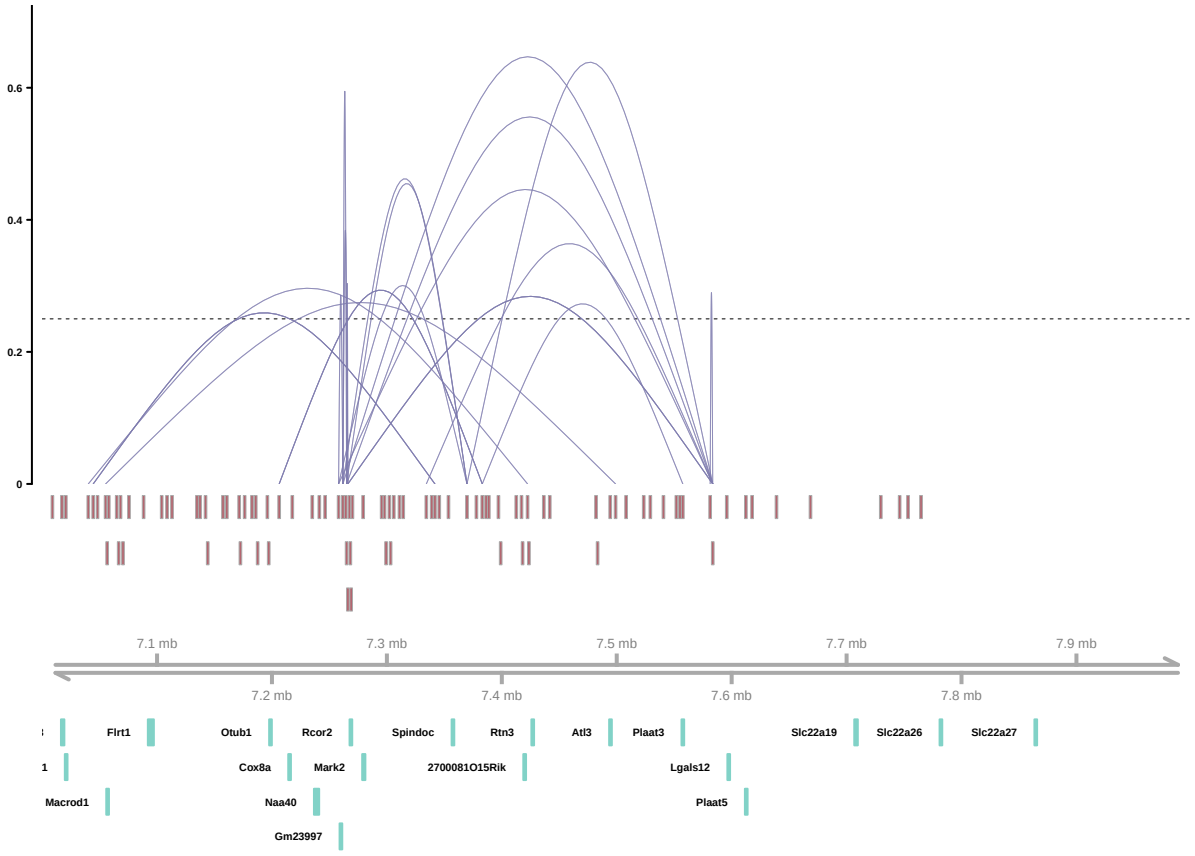
The `Cicero` package also includes a general plotting function for visualizing co-accessibility called `plot_connections`. This function uses the `Gviz` framework for plotting genome browser-style plots. The authors adapted a function from the `Sushi` R package for mapping connections. `plot_connections` has many options, but to get a basic plot from your co-accessibility table is quite simple.

Here, we will include optional `gene_model` data so that genes can be plotted as well.

```
# Construct a data frame to provide a gene annotation
```

```
geneAnno = data.frame(chromosome = seqnames(Annotation(seu.s)) %>% as.factor(),
                      start = start(Annotation(seu.s)), end = end(Annotation(seu.s)),
                      gene = mcols(Annotation(seu.s))$gene_id,
                      strand = strand(Annotation(seu.s)),
                      symbol = mcols(Annotation(seu.s))$gene_name,
                      transcript = mcols(Annotation(seu.s))$tx_id
                      ) %>%
    group_by(transcript) %>% # we have to group transcript IDs here because they may appear multiple ti
    slice_head(n = 1) %>%
    ungroup()

plot_connections(conns, "chr19", 7000000, 8000000,
                 gene_model = geneAnno,
                 coaccess_cutoff = .25,
                 connection_width = .5,
                 collapseTranscripts = "longest" )
```



Single-cell accessibility trajectories

The second major function of the `Cicero` package is to extend `Monocle 3` for use with single-cell accessibility data for clustering, ordering, and differential accessibility analysis of single cells. The main obstacle to overcome with chromatin accessibility data is the sparsity, so most of the extensions and methods are designed to address that.

Constructing trajectories with accessibility data

For constructing the trajectories, as you have learned in the `Monocle 3` vignette, the workflow is generally as follows:

1. Preprocess the data
2. Reduce the dimensionality of the data
3. Cluster the cells
4. Learn the trajectory graph
5. Order the cells in pseudotime

The main `Monocle 3` workflow as described in the `Monocle 3` vignette can be used here, and is therefore not covered in this `Cicero` specific vignette. However, scATAC-seq specific functions that `Cicero` provides will now be mentioned. **We leave it as an exercise to the curious to perform them!**

Differential Accessibility Analysis

The primary way that the `Cicero` package deals with the sparsity of single-cell chromatin accessibility data is through aggregation. Aggregating the counts of either single cells or single peaks allows us to produce a "consensus" count matrix, reducing noise and allowing us to move out of the binary regime. Under this grouping, the number of cells in which a particular site is accessible can be modeled with a binomial distribution or, for sufficiently large groups, the corresponding Gaussian approximation. Modeling grouped accessibility counts as normally distributed allows `Cicero` to easily adjust them for arbitrary technical covariates by simply fitting a linear model and taking the residuals with respect to it as the adjusted accessibility score for each group of cells. We demonstrate how to apply this grouping practically below.

Once you have your cells ordered in pseudotime, you can ask where in the genome chromatin accessibility is changing across time. If you know of specific sites that are important to your system, you may want to visualize the accessibility at those sites across pseudotime using the function `plot_accessibility_in_pseudotime`.

The package also provides the function `aggregate_by_cell_bin` for site-level statistic (whether a site is changing in pseudotime), which works by aggregating similar cells. For more information and examples, see here.

Save object to disk

We reached the end of the vignette. We again save our updated `Seurat` object to disk with a new name, in analogy to what we did in the other vignettes.

```
saveRDS(seu.s, file = paste0(outFolder,"obj.filt.cicero.rds"))
```

Further reading

Pliner, H.A., Packer, J.S., McFaline-Figueroa, J.L., Cusanovich, D.A., Daza, R.M., Aghamirzaie, D., Srivatsan, S., Qiu, X., Jackson, D., Minkina, A. and Adey, A.C., 2018. Cicero predicts cis-regulatory DNA interactions from single-cell chromatin accessibility data. Molecular cell, 71(5), pp.858-871.

Session info

It is good practice to print the so-called session info at the end of an R script, which prints all loaded libraries, their versions etc. This can be helpful for reproducibility and recapitulating which package versions have been used to produce the results obtained above.

```
sessionInfo()
```

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.4 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_GB.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_GB.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_GB.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] grid      stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] cicero_1.3.6                Gviz_1.38.4
##  [3] patchwork_1.1.1             forcats_0.5.1
##  [5] stringr_1.4.0               dplyr_1.0.7
##  [7] purrr_0.3.4                 readr_2.1.1
##  [9] tidyr_1.1.4                 tibble_3.1.6
## [11] ggplot2_3.3.5               tidyverse_1.3.1
## [13] monocle3_1.0.0              SingleCellExperiment_1.16.0
## [15] SummarizedExperiment_1.24.0 GenomicRanges_1.46.1
## [17] GenomeInfoDb_1.30.0         IRanges_2.28.0
## [19] S4Vectors_0.32.3            MatrixGenerics_1.6.0
## [21] matrixStats_0.61.0          Biobase_2.54.0
## [23] BiocGenerics_0.40.0         SeuratWrappers_0.3.0
## [25] sp_1.4-7                    SeuratObject_4.1.0
## [27] Seurat_4.1.1                Signac_1.6.0
##
## loaded via a namespace (and not attached):
##   [1] rappdirs_0.3.3           SnowballC_0.7.0          rtracklayer_1.54.0
##   [4] scattermore_0.8          R.methodsS3_1.8.1        bit64_4.0.5
##   [7] knitr_1.37               irlba_2.3.5              DelayedArray_0.20.0
##  [10] R.utils_2.11.0           data.table_1.14.2        rpart_4.1-15
##  [13] AnnotationFilter_1.18.0  KEGGREST_1.34.0          RCurl_1.98-1.5
##  [16] generics_0.1.1           GenomicFeatures_1.46.3   cowplot_1.1.1
##  [19] RSQLite_2.2.9            VGAM_1.1-6               RANN_2.6.1
##  [22] future_1.25.0            bit_4.0.4                tzdb_0.2.0
##  [25] spatstat.data_2.2-0      xml2_1.3.3               lubridate_1.8.0
##  [28] httpuv_1.6.5             assertthat_0.2.1         viridis_0.6.2
##  [31] xfun_0.29                hms_1.1.1                evaluate_0.14
##  [34] promises_1.2.0.1         fansi_1.0.0              restfulr_0.0.13
##  [37] progress_1.2.2           dbplyr_2.1.1             readxl_1.3.1
##  [40] igraph_1.2.11            DBI_1.1.2                htmlwidgets_1.5.4
##  [43] sparsesvd_0.2            spatstat.geom_2.4-0      ellipsis_0.3.2
##  [46] backports_1.4.1          biomaRt_2.50.2           deldir_1.0-6
##  [49] vctrs_0.4.1              ensembldb_2.18.4         remotes_2.4.2
```

```
##  [52] ROCR_1.0-11              abind_1.4-5              cachem_1.0.6
##  [55] withr_2.4.3              ggforce_0.3.3            BSgenome_1.62.0
##  [58] progressr_0.10.0         checkmate_2.0.0          sctransform_0.3.3
##  [61] GenomicAlignments_1.30.0 prettyunits_1.1.1        goftest_1.2-3
##  [64] cluster_2.1.2            lazyeval_0.2.2           crayon_1.4.2
##  [67] labeling_0.4.2           pkgconfig_2.0.3          slam_0.1-50
##  [70] tweenr_1.0.2             ProtGenerics_1.26.0      nlme_3.1-153
##  [73] nnet_7.3-16              rlang_1.0.2              globals_0.14.0
##  [76] lifecycle_1.0.1          miniUI_0.1.1.1           filelock_1.0.2
##  [79] BiocFileCache_2.2.1      modelr_0.1.8             rsvd_1.0.5
##  [82] dichromat_2.0-0.1        cellranger_1.1.0         polyclip_1.10-0
##  [85] lmtest_0.9-40            Matrix_1.4-0             ggseqlogo_0.1
##  [88] zoo_1.8-10               reprex_2.0.1             base64enc_0.1-3
##  [91] ggridges_0.5.3           png_0.1-7                viridisLite_0.4.0
##  [94] rjson_0.2.21             bitops_1.0-7             R.oo_1.24.0
##  [97] KernSmooth_2.23-20       Biostrings_2.62.0        blob_1.2.2
## [100] parallelly_1.31.1        spatstat.random_2.2-0    jpeg_0.1-9
## [103] scales_1.1.1             memoise_2.0.1            magrittr_2.0.1
## [106] plyr_1.8.6               ica_1.0-2                zlibbioc_1.40.0
## [109] compiler_4.1.2           BiocIO_1.4.0             RColorBrewer_1.1-2
## [112] fitdistrplus_1.1-8       Rsamtools_2.10.0         cli_3.3.0
## [115] XVector_0.34.0           listenv_0.8.0            pbapply_1.5-0
## [118] htmlTable_2.4.0          Formula_1.2-4            MASS_7.3-54
## [121] mgcv_1.8-38              tidyselect_1.1.1         stringi_1.7.6
## [124] highr_0.9                yaml_2.2.1               latticeExtra_0.6-29
## [127] ggrepel_0.9.1            VariantAnnotation_1.40.0 fastmatch_1.1-3
## [130] tools_4.1.2              future.apply_1.9.0       parallel_4.1.2
## [133] rstudioapi_0.13          foreign_0.8-81           lsa_0.73.2
## [136] gridExtra_2.3            farver_2.1.0             Rtsne_0.16
## [139] digest_0.6.29            BiocManager_1.30.16      rgeos_0.5-9
## [142] FNN_1.1.3                shiny_1.7.1              qlcMatrix_0.9.7
## [145] Rcpp_1.0.8               broom_0.7.11             later_1.3.0
## [148] RcppAnnoy_0.0.19         httr_1.4.2               AnnotationDbi_1.56.2
## [151] biovizBase_1.42.0        colorspace_2.0-2         rvest_1.0.2
## [154] XML_3.99-0.8             fs_1.5.2                 tensor_1.5
## [157] reticulate_1.24          splines_4.1.2            uwot_0.1.11
## [160] RcppRoll_0.3.0           spatstat.utils_2.3-0     plotly_4.10.0
## [163] xtable_1.8-4             jsonlite_1.7.2           glasso_1.11
## [166] R6_2.5.1                 Hmisc_4.7-0              pillar_1.6.4
## [169] htmltools_0.5.2          mime_0.12                glue_1.6.0
## [172] fastmap_1.1.0            BiocParallel_1.28.3      codetools_0.2-18
## [175] utf8_1.2.2               lattice_0.20-45          spatstat.sparse_2.1-1
## [178] curl_4.3.2               leiden_0.3.10            survival_3.2-13
## [181] rmarkdown_2.11           docopt_0.7.1             munsell_0.5.0
## [184] GenomeInfoDbData_1.2.7   haven_2.4.3              reshape2_1.4.4
## [187] gtable_0.3.0             spatstat.core_2.4-2
```