Decontamination of ambient RNA in single-cell genomic data with DecontX

Shiyi (Iris) Yang [1], ***Zhe Wang*** [1], ***and Joshua Campbell*** *[1]

[1]Boston University School of Medicine

*camp@bu.edu

2020-10-27

Contents

1 Introduction

Droplet-based microfluidic devices have become widely used to perform single-cell RNA sequencing (scRNA-seq). However, ambient RNA present in the cell suspension can be aberrantly counted along with a cell's native mRNA and result in cross-contamination of transcripts between different cell populations. DecontX is a Bayesian method to estimate and remove contamination in individual cells. DecontX assumes the observed expression of a cell is a mixture of counts from two multinomial distributions: (1) a distribution of native transcript counts from the cell's actual population and (2) a distribution of contaminating transcript counts from all other cell populations captured in the assay. Overall, computational decontamination of single cell counts can aid in downstream clustering and visualization. **Only the expression profile of *"real"* cells after cell calling are required to run DecontX. Empty cell droplet information (low expression cell barcodes before cell calling) are not needed.**

2 Installation

celda can be installed from Bioconductor:

```
if (!requireNamespace("BiocManager", quietly = TRUE)) {
    install.packages("BiocManager")
}
BiocManager::install("celda")
```

The package can be loaded using the `library` command.

```
library(celda)
```

DecontX can take either `SingleCellExperiment` object from package SingleCellExperiment package or a single counts matrix as input. `decontX` will attempt to convert any input matrix to class `dgCMatrix` from package Matrix before beginning any analyses.

3 Load PBMC4k data from 10X

We will utlize the 10X PBMC 4K dataset as an example. This can be easily retrieved from the package TENxPBMCData. Make sure the the column names are set before running decontX.

```
# Install TENxPBMCData if is it not already
if (!requireNamespace("TENxPBMCData", quietly = TRUE)) {
  if (!requireNamespace("BiocManager", quietly = TRUE)) {
    install.packages("BiocManager")
  }
  BiocManager::install("TENxPBMCData")
}

# Load PBMC data
library(TENxPBMCData)
pbmc4k <- TENxPBMCData("pbmc4k")
colnames(pbmc4k) <- paste(pbmc4k$Sample, pbmc4k$Barcode, sep = "_")
rownames(pbmc4k) <- rowData(pbmc4k)$Symbol_TENx
```

4 Running decontX

To run decontX with a SingleCellExperiment object, simply use the following command.

```
pbmc4k <- decontX(x = pbmc4k)
```

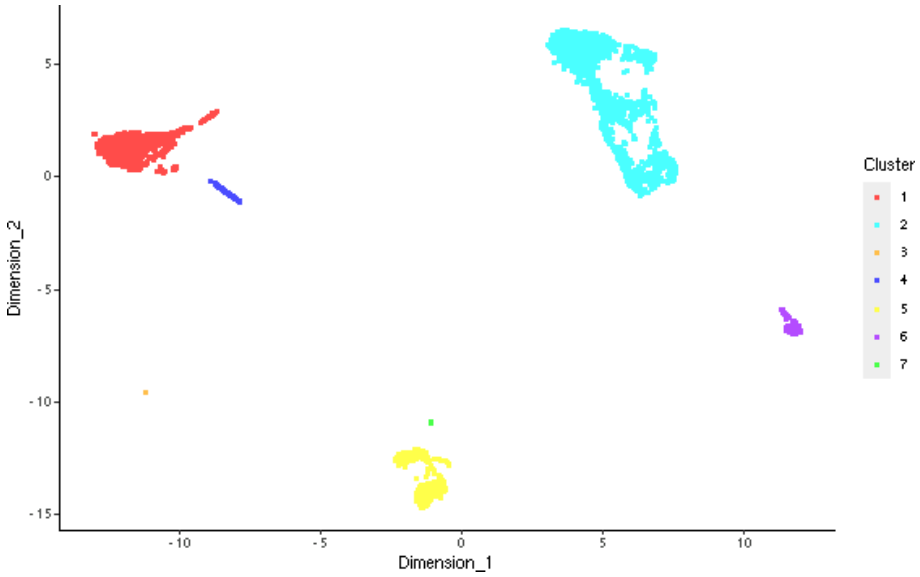
The contamination can be found in the `colData(pbmc4k)$decontX_contamination` and the decontaminated counts can be accessed with `decontXcounts(pbmc4k)`. If the input object was a matrix, make sure to save the output into a variable with a different name (e.g. result). The result object will be a list with contamination in `result$contamination` and the decontaminated counts in `result$decontXcounts`.

5 Plotting DecontX results

5.1 Cluster labels on UMAP

DecontX creates a UMAP which we can use to plot the cluster labels automatically identified in the analysis. Note that the clustering approach used here is designed to find "broad" cell types rather than individual cell subpopulations within a cell type.

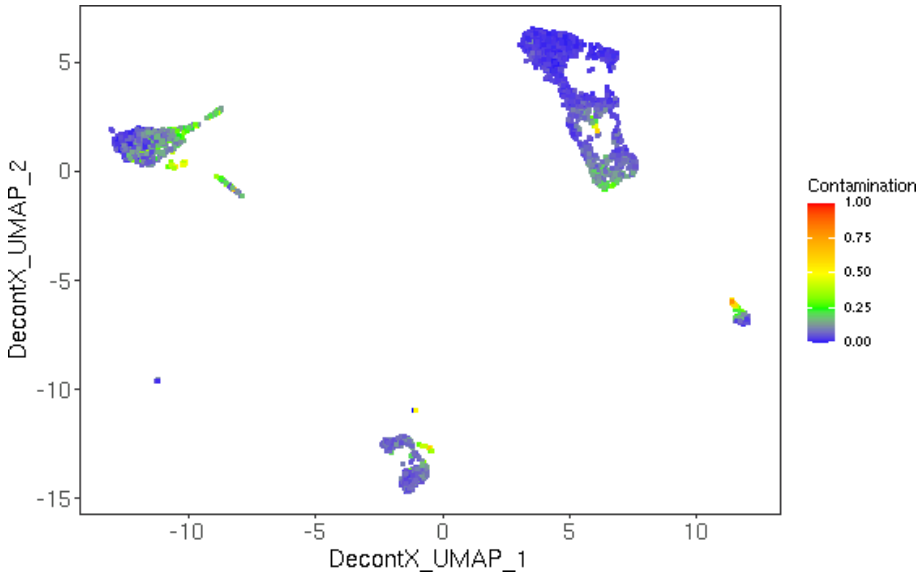
```
umap <- reducedDim(pbmc4k, "decontX_UMAP")
plotDimReduceCluster(x = pbmc4k$decontX_clusters,
    dim1 = umap[, 1], dim2 = umap[, 2])
```



5.2 Contamination on UMAP

The percentage of contamination in each cell can be plotting on the UMAP to visualize what what clusters may have higher levels of ambient RNA.

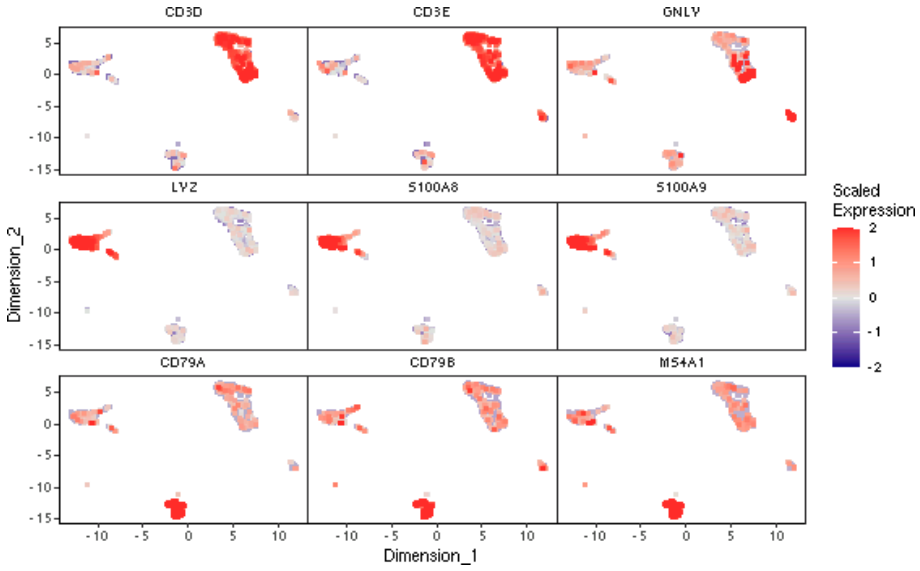
```
plotDecontXContamination(pbmc4k)
```



5.3 Expression of markers on UMAP

Known marker genes can also be plotted on the UMAP to identify the cell types for each cluster. We will use CD3D and CD3E for T-cells, LYZ, S100A8, and S100A9 for monocytes, CD79A, CD79B, and MS4A1 for B-cells, GNLY for NK-cells, and PPBP for megakaryocytes.

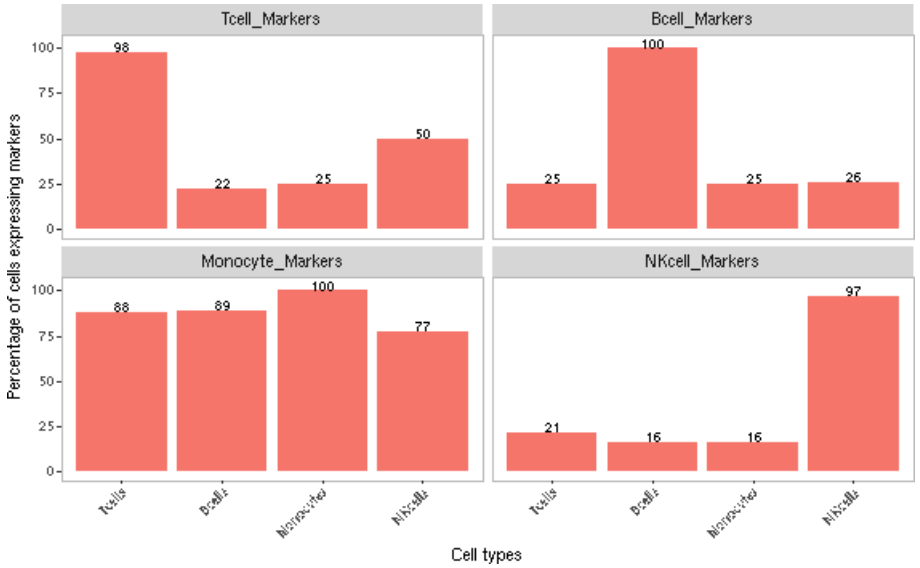
```
library(scater)
pbmc4k <- logNormCounts(pbmc4k)
plotDimReduceFeature(as.matrix(logcounts(pbmc4k)),
    dim1 = umap[, 1],
    dim2 = umap[, 2],
    features = c("CD3D", "CD3E", "GNLY",
        "LYZ", "S100A8", "S100A9",
        "CD79A", "CD79B", "MS4A1"),
    exactMatch = TRUE)
```



5.4 Barplot of markers detected in cell clusters

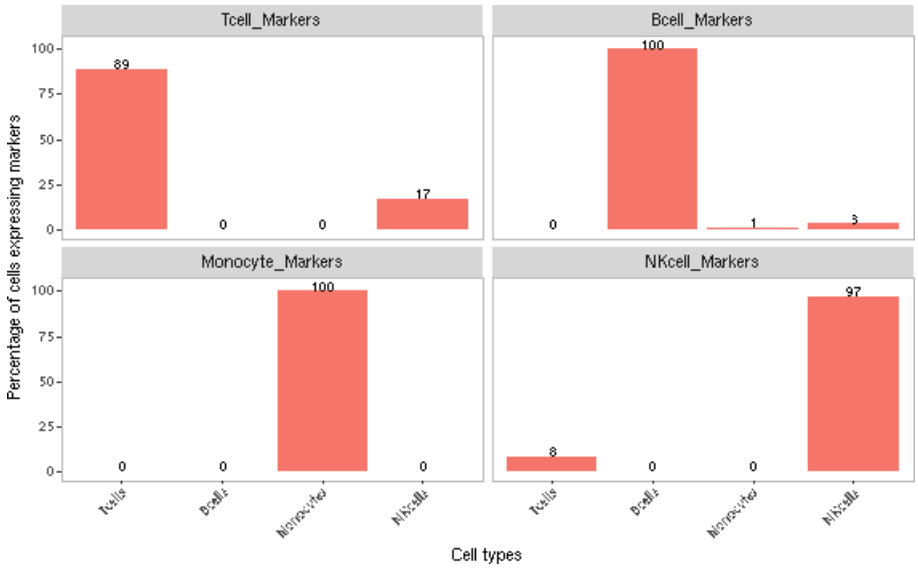
The percetage of cells within a cluster that have detectable expression of marker genes can be displayed in a barplot. Markers for cell types need to be supplied in a named list. First, the detection of marker genes in the original `counts` assay is shown:

```
markers <- list(Tcell_Markers = c("CD3E", "CD3D"),
    Bcell_Markers = c("CD79A", "CD79B", "MS4A1"),
    Monocyte_Markers = c("S100A8", "S100A9", "LYZ"),
    NKcell_Markers = "GNLY")
cellTypeMappings <- list(Tcells = 2, Bcells = 5, Monocytes = 1, NKcells = 6)
plotDecontXMarkerPercentage(pbmc4k,
    markers = markers,
    groupClusters = cellTypeMappings,
    assayName = "counts")
```



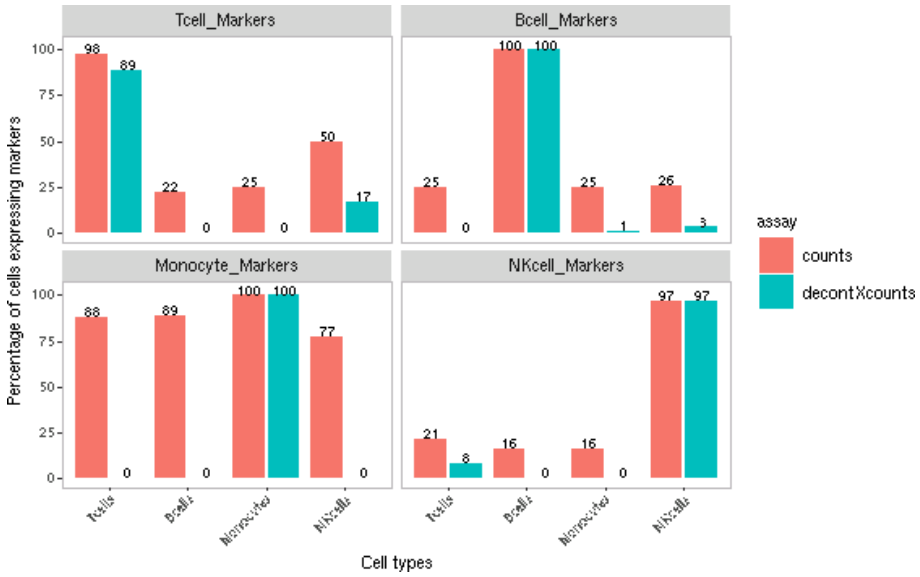
We can then look to see how much DecontX removed aberrant expression of marker genes in each cell type by changing the `assayName` to `decontXcounts`:

```
plotDecontXMarkerPercentage(pbmc4k,
    markers = markers,
    groupClusters = cellTypeMappings,
    assayName = "decontXcounts")
```



Percentages of marker genes detected in other cell types were reduced or completely removed. For example, the percentage of cells that expressed Monocyte marker genes was greatly reduced in T-cells, B-cells, and NK-cells. The original counts and decontamined counts can be plotted side-by-side by listing multiple assays in the `assayName` parameter. This option is only available if the data is stored in `SingleCellExperiment` object.

```
plotDecontXMarkerPercentage(pbmc4k,
    markers = markers,
    groupClusters = cellTypeMappings,
    assayName = c("counts", "decontXcounts"))
```



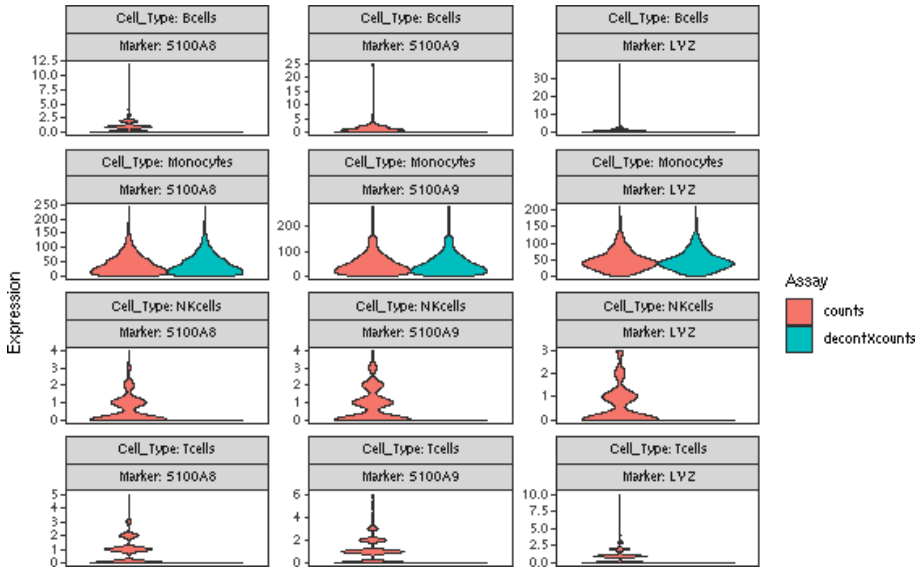
Some helpful hints when using `plotDecontXMarkerPercentage`:

1. Cell clusters can be renamed and re-grouped using the `groupCluster` parameter, which also needs to be a named list. If `groupCluster` is used, cell clusters not included in the list will be excluded in the barplot. For example, if we wanted to group T-cells and NK-cells together, we could set `cellTypeMappings <- list(NK_Tcells = c(2,6), Bcells = 5, Monocytes = 1)`
2. The level a gene needs to be expressed to be considered detected in a cell can be adjusted using the `threshold` parameter.
3. If you are not using a `SingleCellExperiment`, then you will need to supply the original counts matrix or the decontaminated counts matrix as the first argument to generate the barplots.

5.5 Violin plot to compare the distributions of original and decontaminated counts

Another useful way to assess the amount of decontamination is to view the expression of marker genes before and after `decontX` across cell types. Here we view the monocyte markers in each cell type. The violin plot shows that the markers have been removed from T-cells, B-cells, and NK-cells, but are largely unaffected in monocytes.

```
plotDecontXMarkerExpression(pbmc4k,
    markers = markers[["Monocyte_Markers"]],
    groupClusters = cellTypeMappings,
    ncol = 3)
```



Some helfpul hints when using `plotDecontXMarkerExpression`:

1. `groupClusters` works the same way as in `plotDecontXMarkerPercentage`.
2. This function will plot each pair of markers and clusters (or cell type specified by `groupClusters`). Therefore, you may want to keep the number of markers small in each plot and call the function multiple times for different sets of marker genes.
3. You can also plot the individual points by setting `plotDots = TRUE` and/or log tranform the points on the fly by setting `log1p = TRUE`.
4. This function can plot any assay in a `SingleCellExperiment`. Therefore you could also examine normalized expression of the original and decontaminated counts. For example:

```
pbmc4k <- scater::logNormCounts(pbmc4k,
    exprs_values = "decontXcounts",
    name = "dlogcounts")

plotDecontXMarkerExpression(pbmc4k,
    markers = markers[["Monocyte_Markers"]],
    groupClusters = cellTypeMappings,
    ncol = 3,
    assayName = c("logcounts", "dlogcounts"))
```

6 Other important notes

6.1 Choosing appropriate cell clusters

The ability of DecontX to accurately identify contamination is dependent on the cell cluster labels. DecontX assumes that contamination for a cell cluster comes from combination of counts from all other clusters. The default clustering approach used by DecontX tends to select fewer clusters that represent broader cell types. For example, all T-cells tend to be clustered together rather than splitting naive and cytotoxic T-cells into separate clusters. Custom cell type labels can be suppled via the `z` parameter if some cells are not being clustered appropriately by the default method.

6.2 Adjusting the priors to influence contamination estimates

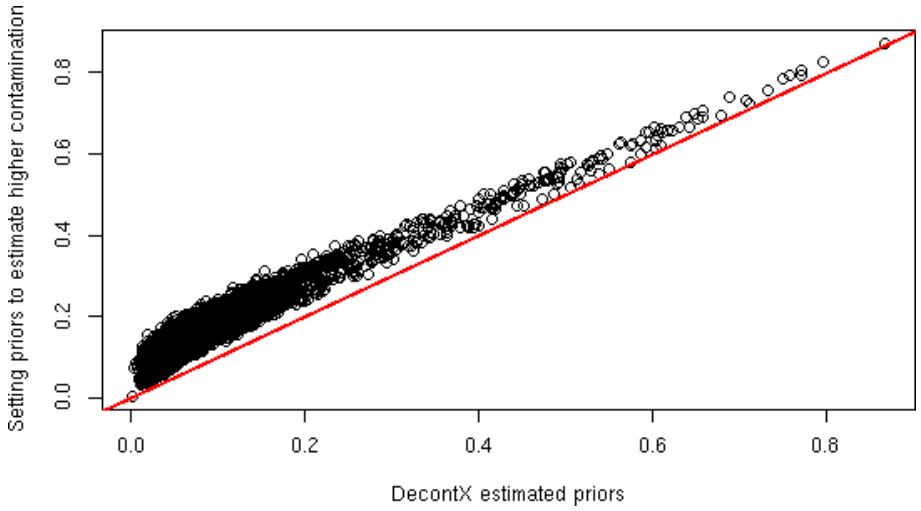
There are ways to force `decontX` to estimate more or less contamination across a dataset by manipulating the priors. The `delta` parameter is a numeric vector of length two. It is the concentration parameter for the Dirichlet distribution which serves as the prior for the proportions of native and contamination counts in each cell. The first element is the prior for the proportion of native counts while the second element is the prior for the proportion of contamination counts. These essentially act as pseudocounts for the native and contamination in each cell. If `estimateDelta = TRUE`, `delta` is only used to produce a random sample of proportions for an initial value of contamination in each cell. Then `delta` is updated in each iteration. If `estimateDelta = FALSE`, then `delta` is fixed with these values for the entire inference procedure. Fixing `delta` and setting a high number in the second element will force `decontX` to be more aggressive and estimate higher levels of contamination in each cell at the expense of potentially removing native expression. For example, in the previous PBMC example, we can see what the estimated `delta` was by looking in the estimates:

```
metadata(pbmc4k)$decontX$estimates$all_cells$delta
## [1] 8.751503 1.010116
```

Setting a higher value in the second element of delta and `estimateDelta = FALSE` will force `decontX` to estimate higher levels of contamination per cell:

```
pbmc4k.delta <- decontX(pbmc4k, delta = c(9, 20), estimateDelta = FALSE)

plot(pbmc4k$decontX_contamination, pbmc4k.delta$decontX_contamination,
     xlab = "DecontX estimated priors",
     ylab = "Setting priors to estimate higher contamination")
abline(0, 1, col = "red", lwd = 2)
```



7 Session Information

```
sessionInfo()
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-apple-darwin17.0 (64-bit)
```

```
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] scater_1.18.0               ggplot2_3.3.2
##  [3] TENxPBMCData_1.7.0          HDF5Array_1.18.0
##  [5] rhdf5_2.34.0                DelayedArray_0.16.0
##  [7] Matrix_1.2-18               SingleCellExperiment_1.12.0
##  [9] SummarizedExperiment_1.20.0 Biobase_2.50.0
## [11] GenomicRanges_1.42.0        GenomeInfoDb_1.26.0
## [13] IRanges_2.24.0              S4Vectors_0.28.0
## [15] BiocGenerics_0.36.0         MatrixGenerics_1.2.0
## [17] matrixStats_0.57.0          celda_1.6.1
## [19] BiocStyle_2.18.0
##
## loaded via a namespace (and not attached):
##   [1] circlize_0.4.10               AnnotationHub_2.22.0
##   [3] BiocFileCache_1.14.0          RcppEigen_0.3.3.7.0
##   [5] igraph_1.2.6                  plyr_1.8.6
##   [7] assertive.files_0.0-2         enrichR_2.1
##   [9] multipanelfigure_2.1.2        BiocParallel_1.24.0
##  [11] digest_0.6.27                 foreach_1.5.1
##  [13] htmltools_0.5.0               viridis_0.5.1
##  [15] magick_2.5.0                  magrittr_1.5
##  [17] memoise_1.1.0                 assertive.numbers_0.0-2
##  [19] cluster_2.1.0                 doParallel_1.0.16
##  [21] limma_3.46.0                  ComplexHeatmap_2.6.0
##  [23] prettyunits_1.1.1             colorspace_1.4-1
##  [25] blob_1.2.1                    rappdirs_0.3.1
##  [27] ggrepel_0.8.2                 xfun_0.18
##  [29] dplyr_1.0.2                   crayon_1.3.4
##  [31] RCurl_1.98-1.2                iterators_1.0.13
##  [33] glue_1.4.2                    gtable_0.3.0
##  [35] zlibbioc_1.36.0               XVector_0.30.0
##  [37] GetoptLong_1.0.4              BiocSingular_1.6.0
##  [39] Rhdf5lib_1.12.0               shape_1.4.5
##  [41] abind_1.4-5                   scales_1.1.1
##  [43] edgeR_3.32.0                  DBI_1.1.0
##  [45] Rcpp_1.0.5                    viridisLite_0.3.0
##  [47] xtable_1.8-4                  progress_1.2.2
##  [49] clue_0.3-57                   dqrng_0.2.1
```

```
##  [51] gridGraphics_0.5-0           bit_4.0.4
##  [53] rsvd_1.0.3                   httr_1.4.2
##  [55] RColorBrewer_1.1-2           ellipsis_0.3.1
##  [57] pkgconfig_2.0.3              farver_2.0.3
##  [59] scuttle_1.0.0                uwot_0.1.8
##  [61] dbplyr_1.4.4                 locfit_1.5-9.4
##  [63] tidyselect_1.1.0             labeling_0.4.2
##  [65] rlang_0.4.8                  reshape2_1.4.4
##  [67] later_1.1.0.1                AnnotationDbi_1.52.0
##  [69] munsell_0.5.0                BiocVersion_3.12.0
##  [71] tools_4.0.3                  dbscan_1.1-5
##  [73] generics_0.0.2               RSQLite_2.2.1
##  [75] ExperimentHub_1.16.0         evaluate_0.14
##  [77] stringr_1.4.0                fastmap_1.0.1
##  [79] yaml_2.2.1                   knitr_1.30
##  [81] bit64_4.0.5                  purrr_0.3.4
##  [83] sparseMatrixStats_1.2.0      mime_0.9
##  [85] scran_1.18.0                 compiler_4.0.3
##  [87] beeswarm_0.2.3               curl_4.3
##  [89] png_0.1-7                    interactiveDisplayBase_1.28.0
##  [91] statmod_1.4.35               tibble_3.0.4
##  [93] stringi_1.5.3                RSpectra_0.16-0
##  [95] bluster_1.0.0                lattice_0.20-41
##  [97] assertive.base_0.0-7         vctrs_0.3.4
##  [99] pillar_1.4.6                 lifecycle_0.2.0
## [101] rhdf5filters_1.2.0           BiocManager_1.30.10
## [103] combinat_0.0-8               GlobalOptions_0.1.2
## [105] RcppAnnoy_0.0.16             BiocNeighbors_1.8.0
## [107] data.table_1.13.2            bitops_1.0-6
## [109] irlba_2.3.3                  httpuv_1.5.4
## [111] assertive.types_0.0-3        R6_2.4.1
## [113] bookdown_0.21                assertive.properties_0.0-4
## [115] promises_1.1.1               gridExtra_2.3
## [117] vipor_0.4.5                  codetools_0.2-16
## [119] MCMCprecision_0.4.0          assertthat_0.2.1
## [121] MAST_1.16.0                  rjson_0.2.20
## [123] withr_2.3.0                  GenomeInfoDbData_1.2.4
## [125] hms_0.5.3                    grid_4.0.3
## [127] beachmat_2.6.0               rmarkdown_2.5
## [129] DelayedMatrixStats_1.12.0    Cairo_1.5-12.2
## [131] Rtsne_0.15                   shiny_1.5.0
## [133] ggbeeswarm_0.6.0             tinytex_0.26
```