MoleculeExperiment

The R package MoleculeExperiment contains functions to create and work with the MoleculeExperiment class. We introduce this class for analysing molecule-based spatial transcriptomics data (xenium by 10x, cosmx SMI by nanostring, and merscope by vizgen).

Contents

TODO Have hyperlinks here 1. Why the MoleculeExperiment class 2. The ME object ...

Why the MoleculeExperiment class?

1) Enable easy analysis of spatial transcriptomics data at the molecule level, rather than the cell level.
2) Standardisation of molecule-based ST data across vendors, to hopefully facilitate comparison of different data sources.

The ME object

Constructing an ME object

```
# load necessary libraries
library(devtools)
```

```
## Loading required package: usethis
```

```
load_all()
```

```
## i Loading SpatialUtils
```

```
#library(MoleculeExperiment)
library(ggplot2)
```

usecase 1

We will demonstrate how to generate an ME object with toy data representing a scenario where both the detected transcripts information and the boundary information is read into R.

First we generate a toy transcripts data.frame:

```
# molecules data.frame toy example
molecules_df <- data.frame(
  sample_id = rep(c("sample1", "sample2"), times = c(30, 20)),
  feature_name = rep(c("gene1", "gene2"), times = c(20, 30)),
  x_location = runif(50),
  y_location = runif(50)
  )
head(molecules_df)
```

```
##   sample_id feature_name x_location y_location
## 1   sample1        gene1 0.39662798  0.5046235
## 2   sample1        gene1 0.44778193  0.9488975
## 3   sample1        gene1 0.99068979  0.2600446
## 4   sample1        gene1 0.20336426  0.1255218
## 5   sample1        gene1 0.03030918  0.4976240
## 6   sample1        gene1 0.55542501  0.1471956
```

Then we generate a toy boundaries data.frame:

```
# boundaries data.frame toy example
boundaries_df <- data.frame(
  sample_id = rep(c("sample1", "sample2"), times = c(16, 6)),
  cell_id = rep(c("cell1", "cell2", "cell3", "cell4",
                  "cell1", "cell2"),
                times = c(4, 4, 4, 4, 3, 3)),
  x_location = c(0, 0.5, 0.5, 0,
                 0.5, 1, 1, 0.5,
                 0, 0.5, 0.5, 0,
                 0.5, 1, 1, 0.5,
                 0, 1, 0,
                 0, 1, 1),
  y_location = c(0, 0, 0.5, 0.5,
                 0, 0, 0.5, 0.5,
                 0.5, 0.5, 1, 1,
                 0.5, 0.5, 1, 1,
                 0, 1, 1,
                 0, 0, 1)
)
head(boundaries_df)
```

```
##   sample_id cell_id x_location y_location
## 1   sample1   cell1        0.0        0.0
## 2   sample1   cell1        0.5        0.0
## 3   sample1   cell1        0.5        0.5
## 4   sample1   cell1        0.0        0.5
## 5   sample1   cell2        0.5        0.0
## 6   sample1   cell2        1.0        0.0
```

To generate an ME object, the next step is to standardise these lists to the MoleculeExperiment list. This consists of a list of lists, ultimately ending in a tibble. This data structure was used to store disk space, as data in a list enables us to avoid redundantly storing gene names or sample IDs for the millions of transcripts. In the case of the molecules information, we would like to store the information in a list of lists with the following structure: "sample ID" > "feature name" > tibble with X and Y locations (and other additional columns of interest). Additionally, one might like to filter the transcripts and store them in the same object, in which case one can also have this structure: "filtered" > "sample ID" > "feature name" > tibble.

```
molecules_ls <- dataframeToMEList(molecules_df,
                                  df_type = "transcripts",
                                  assay_name = "raw",
                                  sample_col = "sample_id",
                                  factor_col = "feature_name",
                                  x_col = "x_location",
                                  y_col = "y_location")
# to avoid printing large nested list in the terminal, use str() and the
# max.level argument
str(molecules_ls, max.level = 3)
```

```
## List of 1
##  $ raw:List of 2
##   ..$ sample1:List of 2
##   .. ..$ gene1: tibble [20 x 2] (S3: tbl_df/tbl/data.frame)
##   .. ..$ gene2: tibble [10 x 2] (S3: tbl_df/tbl/data.frame)
##   ..$ sample2:List of 1
```

```
##   .. ..$ gene2: tibble [20 x 2] (S3: tbl_df/tbl/data.frame)
```

For the boundaries slot, if the boundary information is for cells, the structure would look like this: "cells" > "sample ID" > "cell IDs" > tibble with the vertex coordinates defining the boundaries for each cell. If the boundary information is for nuclei, the structure would be: "nuclei" > "sample ID" > "cell ID" > tibble with the vertex coordinates defining the boundary information for the nuclei.

```
boundaries_ls <- dataframeToMEList(boundaries_df,
                                   df_type = "boundaries",
                                   assay_name = "cells",
                                   sample_col = "sample_id",
                                   factor_col = "cell_id",
                                   x_col = "x_location",
                                   y_col = "y_location")
str(boundaries_ls, 3)
```

```
## List of 1
##  $ cells:List of 2
##   ..$ sample1:List of 4
##   .. ..$ cell1: tibble [4 x 2] (S3: tbl_df/tbl/data.frame)
##   .. ..$ cell2: tibble [4 x 2] (S3: tbl_df/tbl/data.frame)
##   .. ..$ cell3: tibble [4 x 2] (S3: tbl_df/tbl/data.frame)
##   .. ..$ cell4: tibble [4 x 2] (S3: tbl_df/tbl/data.frame)
##   ..$ sample2:List of 2
##   .. ..$ cell1: tibble [3 x 2] (S3: tbl_df/tbl/data.frame)
##   .. ..$ cell2: tibble [3 x 2] (S3: tbl_df/tbl/data.frame)
```

Now that the transcript and boundary information is in a standardised format, it can be used as input to generate an ME object.

```
# use MoleculeExperiment object constructor
toy_me <- MoleculeExperiment(molecules = molecules_ls,
                             boundaries = boundaries_ls)

# visualise contents of me object
toy_me
```

```
## class:  MoleculeExperiment
## 2 samples: sample1 sample2
##
## @molecules contents:
## -raw assay:
## 2 unique features across all samples in assay raw
## 25 molecules on average across all samples in assay raw
## Location range across all samples in assay raw: [0,0.99] x [0.04,0.99]
##
## @boundaries contents:
## -cells:
## 3 unique compartments: cell1 cell2 cell3 cell4 ...
## Location range across all samples: [0,1] x [0,1]
```

Usecase 2: using directory with Xenium data as input for an ME.

The MoleculeExperiment package also contains convenience functions to read in data from a Xenium's output directory. This is especially useful when wanting to create an ME object with data from multiple samples.

```
# create ME obj with example mouse brain xenium dataset
repo_dir <- "/dski/nobackup/bpeters/SpatialUtils/inst/extdata/mouse_brain_mini_xenium"
me <- readXenium(repo_dir,
                 n_samples = 2,
                 keep_cols = "essential",
                 add_boundaries = TRUE)
```

```
##
## Detected transcript information can be accessed with molecules(me) or
## molecules(me, "raw")
## Boundary information can be accessed with boundaries(me)
```

```
# visualise me contents
me
```

```
## class:  MoleculeExperiment
## 2 samples: Xenium_V1_FF_Mouse_Brain_MultiSection_1_outs Xenium_V1_FF_Mouse_Brain_MultiSection_2_outs
##
## @molecules contents:
## -raw assay:
## 16 unique features across all samples in assay raw
## 99 molecules on average across all samples in assay raw
## Location range across all samples in assay raw: [4817.3,4880.05] x [286.19,6632.11]
##
## @boundaries contents:
## -cells:
## 8 unique compartments: 1 2 3 4 5 6 ...
## Location range across all samples: [1076.31,1621.59] x [2519.19,3411.69]
```

readXenium calls readMolecules and readBoundaries under the hood. These convenience function can also be used by themselves, which might be useful in this context where new molecule-level spatial transcriptomics technologies are being developed.

- highlight benefits of how readXenium() works e.g., readMolecules enables the user to decide if they want to keep all the data that is vendor-specific (e.g., qv in xenium).

ME object structure

- what are the slots? For now, the MoleculeExperiment contains a @molecules slot and an @boundaries slot.

molecules slot

The "@molecules" slot contains molecule-level information. The essential data it contains is the gene name and x and y locations of the detected transcripts, in each sample. Nevertheless, the user can also decide to keep all transcript metadata (e.g., subcellular location: nucleus/cytoplasm).

highlight how this enables standardisation of ST data across different vendors.

```
# list contents can be very large. We recommend visualising contents with the
# summariseMolecules method
strMolecules(me)
```

```
## List of 1
##  $ raw:List of 2
##   ..$ Xenium_V1_FF_Mouse_Brain_MultiSection_1_outs:List of 13
##   .. ..$ Bhlhe40               : tibble [22 x 2] (S3: tbl_df/tbl/data.frame)
##   .. ..$ Car4                  : tibble [8 x 2] (S3: tbl_df/tbl/data.frame)
```

```
##   .. .. [list output truncated]
##   ..$ Xenium_V1_FF_Mouse_Brain_MultiSection_2_outs:List of 13
##   .. ..$ Bhlhe40                : tibble [27 x 2] (S3: tbl_df/tbl/data.frame)
##   .. ..$ Cpne8                  : tibble [1 x 2] (S3: tbl_df/tbl/data.frame)
##   .. .. [list output truncated]
```

- what is the format in which the information is stored? for now the list of lists with dfs format.
- Why? Explain

boundaries slot

The "@boundaries" slot contains a list of lists, also ultimately ending in tibbles. The boundaries can come from e.g., cells, or nuclei. To store these different boundaries in the same object, one can specify the title/header of each list to easily access specific information later on.

```
strBoundaries(me)
```

```
## List of 1
##  $ cells:List of 2
##   ..$ Xenium_V1_FF_Mouse_Brain_MultiSection_1_outs:List of 8
##   .. ..$ 1: tibble [13 x 2] (S3: tbl_df/tbl/data.frame)
##   .. ..$ 2: tibble [13 x 2] (S3: tbl_df/tbl/data.frame)
##   .. .. [list output truncated]
##   ..$ Xenium_V1_FF_Mouse_Brain_MultiSection_2_outs:List of 8
##   .. ..$ 1: tibble [13 x 2] (S3: tbl_df/tbl/data.frame)
##   .. ..$ 2: tibble [13 x 2] (S3: tbl_df/tbl/data.frame)
##   .. .. [list output truncated]
```

Basic methods to work with an ME object

- Briefly introduce all methods that can be used to access and manipulate data in the @molecules slot.
  - getters: molecules() and boundaries()

```
# note that output from the following methods can be very large.
# These getters should be used when the data from the slots needs to be used as
# input for other functions.

# molecules(me) or molecules(me, "raw")
identical(molecules(me), molecules(me, "raw"))
```

```
## Warning in .local(object, ...): The transcripts from the raw assay were retrieved.
## Other assay transcripts can be retrieved by specifying the assay_name argument.

## Warning in .local(object, ...): The transcripts from the raw assay were retrieved.
## Other assay transcripts can be retrieved by specifying the assay_name argument.
```

```
## [1] TRUE
```

```
molecules(me)[[1]][[1]][[1]]
```

```
## Warning in .local(object, ...): The transcripts from the raw assay were retrieved.
## Other assay transcripts can be retrieved by specifying the assay_name argument.
```

```
## # A tibble: 22 x 2
##    x_location y_location
##         <dbl>      <dbl>
##  1      4843.      6428.
```

```
##  2      4843.      6478.
##  3      4844.      6525.
##  4      4845.      6460.
##  5      4848.      6331.
##  6      4847.      6478.
##  7      4848.      6520.
##  8      4849.      6262.
##  9      4848.      6476.
## 10      4850.      6290.
## # i 12 more rows
```

```
# it is recommended to use strMolecules or strBoundaries when trying to
# get a quick visualisation of the data.
```

```
features(me)
```

```
## $Xenium_V1_FF_Mouse_Brain_MultiSection_1_outs
##  [1] "Bhlhe40"                 "Car4"
##  [3] "Cpne8"                   "Dkk3"
##  [5] "Gjb2"                    "Lyz2"
##  [7] "NegControlCodeword_0505" "Neurod6"
##  [9] "Parm1"                   "Sema5b"
## [11] "Sox10"                   "Sst"
## [13] "Wfs1"
##
## $Xenium_V1_FF_Mouse_Brain_MultiSection_2_outs
##  [1] "Bhlhe40"                 "Cpne8"
##  [3] "Dkk3"                    "Fign"
##  [5] "Gjb2"                    "Lyz2"
##  [7] "NegControlCodeword_0529" "Neurod6"
##  [9] "Nxph3"                   "Parm1"
## [11] "Sema5b"                  "Sox10"
## [13] "Wfs1"
```

```
# boundaries(me, "cells")
boundaries(me, "cells")[[1]][[1]][[1]]
```

```
## # A tibble: 13 x 2
##    x_location y_location
##         <dbl>      <dbl>
##  1      1555.      2519.
##  2      1551.      2522.
##  3      1550.      2523.
##  4      1550.      2527.
##  5      1551.      2538.
##  6      1556.      2537.
##  7      1561.      2536.
##  8      1563.      2535.
##  9      1564.      2531.
## 10      1567.      2524.
## 11      1567.      2522.
## 12      1556.      2519.
## 13      1555.      2519.
```

```
compartmentIDs(me, "cells")
```

```
## $Xenium_V1_FF_Mouse_Brain_MultiSection_1_outs
```

```
## [1] "1" "2" "3" "4" "5" "6" "7" "8"
##
## $Xenium_V1_FF_Mouse_Brain_MultiSection_2_outs
## [1] "1" "2" "3" "4" "5" "6" "7" "8"
```

- setters: readBoundaries()

```
nuclei_ls <- readBoundaries(data_dir = repo_dir,
                            pattern = "nucleus_boundaries.csv",
                            n_samples = 2,
                            compartment_id_col = "cell_id",
                            x_col = "vertex_x",
                            y_col = "vertex_y",
                            keep_cols = "essential",
                            boundaries_mode = "nucleus")

# add nucleus boundaries to already existing me object
boundaries(me, "nuclei") <- nuclei_ls
me # note the addition of the nuclei boundaries to the nuclei slot
```

```
## class:  MoleculeExperiment
## 2 samples: Xenium_V1_FF_Mouse_Brain_MultiSection_1_outs Xenium_V1_FF_Mouse_Brain_MultiSection_2_outs
##
## @molecules contents:
## -raw assay:
## 16 unique features across all samples in assay raw
## 99 molecules on average across all samples in assay raw
## Location range across all samples in assay raw: [4817.3,4880.05] x [286.19,6632.11]
##
## @boundaries contents:
## -cells:
## 8 unique compartments: 1 2 3 4 5 6 ...
## Location range across all samples: [1076.31,1621.59] x [2519.19,3411.69]
## -nuclei:
## 8 unique compartments: 1 2 3 4 5 6 ...
## Location range across all samples: [1089.91,1622.01] x [2526.41,3404.68]
```

```
# note that one can also assign new lists to the molecules slot with molecules<-
```

From MoleculeExperiment to SpatialExperiment

The idea behind the MoleculeExperiment class is to store data form molecule-based spatial transcriptomics technologies in a way that enables a molecule-level analysis, and a standardisation of data across the many different vendors. Nevertheless, if one is interested in continuing downstream analysis at the cell-level, the MoleculeExperiment package also provides a convenience function, called countMolecules(), that enables the transition from a MoleculeExperiment object to a SpatialExperiment object. With this functionality, it is possible to use already existing methods developed to analyse data stored as a SpatialExperiment object.

Supose we are working with the toy example of boundary information described before. Recall it below:

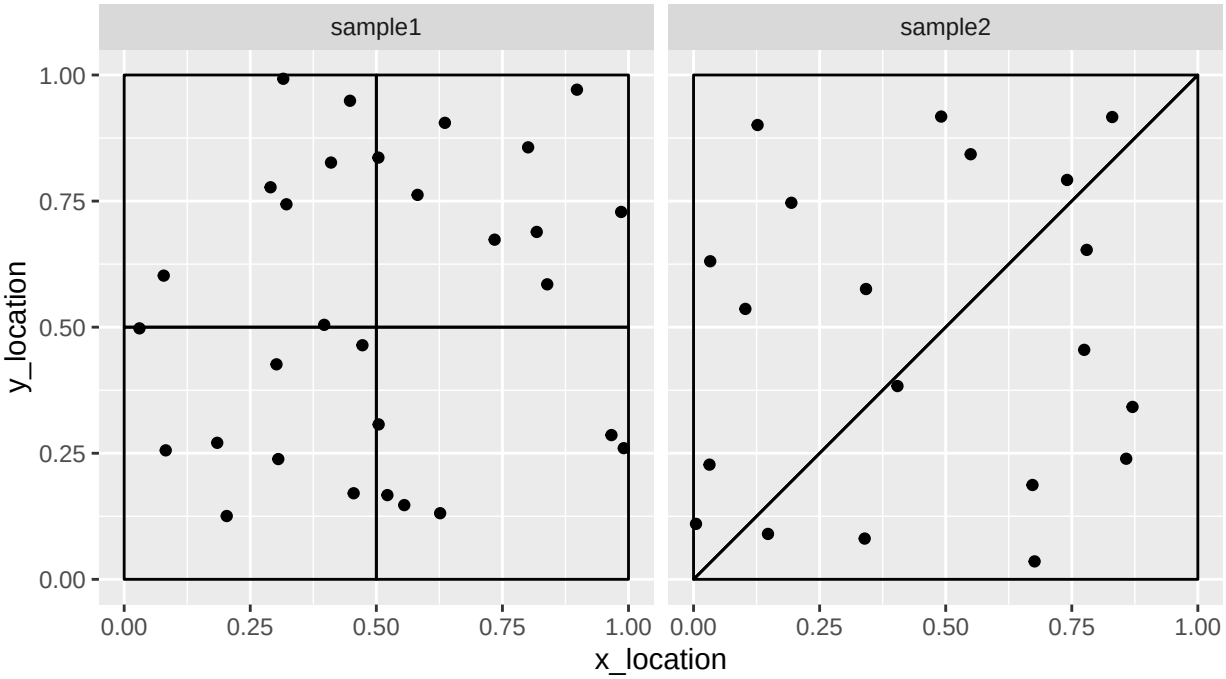
```
head(boundaries_df)
```

```
##   sample_id cell_id x_location y_location
## 1   sample1   cell1        0.0        0.0
## 2   sample1   cell1        0.5        0.0
## 3   sample1   cell1        0.5        0.5
```

```
## 4   sample1   cell1        0.0        0.5
## 5   sample1   cell2        0.5        0.0
## 6   sample1   cell2        1.0        0.0
```

This would be generating a scenario that graphically looks like this:

```
ggplot(molecules_df, aes(x = x_location, y = y_location)) +
  geom_point() +
  geom_polygon(aes(group = cell_id),
                   fill = NA,
                   colour = "black",
                   data = boundaries_df) +
  facet_wrap(~sample_id) +
  coord_fixed()
```



The idea now is to go from a MoleculeExperiment object to a SpatialExperiment object.

```
#countMolecules()
```

Future developments

- SpatialUtils
- one plot to show usefulness of @molecules slot
- instructions on how to use readMolecules() for up-and-coming molecule-based ST technologies.

sessionInfo()

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Debian GNU/Linux 11 (bullseye)
##
```

```
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.13.so
##
## locale:
##  [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
##  [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
##  [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
## [10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ggplot2_3.4.1           SpatialUtils_0.0.0.9000 devtools_2.4.5
## [4] usethis_2.1.6
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.2.0  xfun_0.36         remotes_2.4.2     purrr_1.0.0
##  [5] colorspace_2.1-0  vctrs_0.5.2       generics_0.1.3    miniUI_0.1.1.1
##  [9] htmltools_0.5.4   yaml_2.3.7        utf8_1.2.3        rlang_1.0.6
## [13] pkgbuild_1.4.0    later_1.3.0       urlchecker_1.0.1  pillar_1.9.0
## [17] glue_1.6.2        withr_2.5.0       bit64_4.0.5       sessioninfo_1.2.2
## [21] lifecycle_1.0.3   stringr_1.5.0     munsell_0.5.0     gtable_0.3.1
## [25] htmlwidgets_1.6.2 memoise_2.0.1     evaluate_0.20     labeling_0.4.2
## [29] knitr_1.42        callr_3.7.3       fastmap_1.1.1     httpuv_1.6.8
## [33] ps_1.7.3          fansi_1.0.4       highr_0.10        Rcpp_1.0.10
## [37] xtable_1.8-4      scales_1.2.1      promises_1.2.0.1  cachem_1.0.7
## [41] desc_1.4.2        pkgload_1.3.2     farver_2.1.1      bit_4.0.5
## [45] mime_0.12         fs_1.6.1          digest_0.6.31     stringi_1.7.8
## [49] processx_3.8.0    dplyr_1.1.0       shiny_1.7.4       rprojroot_2.0.3
## [53] grid_4.2.1        cli_3.6.1         tools_4.2.1       magrittr_2.0.3
## [57] tibble_3.2.0      profvis_0.3.7     crayon_1.5.2      pkgconfig_2.0.3
## [61] ellipsis_0.3.2    data.table_1.14.4 prettyunits_1.1.1 rmarkdown_2.21
## [65] rstudioapi_0.14   R6_2.5.1          compiler_4.2.1
```