

# Quick start of flowSpy

Yuting Dai

2020-02-19

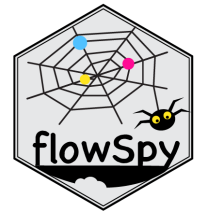
- [Introduction](#)
- [Overview of flowSpy workflow](#)
- [References](#)

## Introduction

Although multidimensional single-cell-based flow and mass cytometry have been increasingly applied to microenvironmental composition and stem-cell research, integrated analysis workflows to facilitate the interpretation of experimental cytometry data remain underdeveloped. We present flowSpy, a comprehensive R package designed for the analysis and interpretation of flow and mass cytometry data. We applied flowSpy to mass cytometry and time-course flow cytometry data to demonstrate the usage and practical utility of its computational modules. flowSpy is a reliable tool for multidimensional cytometry data workflows and produces compelling results for trajectory construction and pseudotime estimation.

## Overview of flowSpy workflow

The flowSpy package is developed to complete the majority of standard analysis and visualization workflow for FCS data. In flowSpy workflow, an S4 object in R is built to implement the statistical and computational approach, and all computational modules are integrated into one single channel which only requires a specified input data format. Computational modules of flowSpy can be divided into four main parts (Fig. 1): preprocessing, trajectory, analysis and visualization.



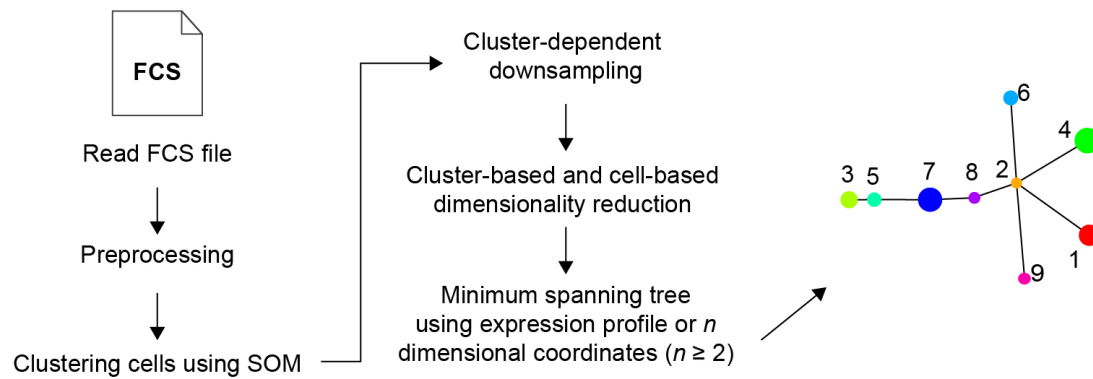
- **Preprocessing.** Data import, compensation, quality control, filtration, normalization and merge cells from different samples can be implemented in the preprocessing module. After preprocessing, a matrix contains clean cytometric signaling data, a data.frame containing meta-information of the experiment and a vector contains all markers enrolled in the computational process are required to build an FPSY object.
- **Trajectory.** Cells built in the FPSY object are classified into different clusters based on the expression level of input markers. You can choose different clustering methods by inputting different parameters. After clustering, cells are downsampled in a cluster-dependent fashion to reduce the total cell size and avoid small cluster deletion. Dimensionality reduction for both cells and clusters are also implemented in the clustering procedure. After dimensionality reduction, we use Minimus Spanning Tree (MST) to construct cell trajectory.
- **Analysis.** This module is design for time couse FCS data. Before running pseudotime, root cells must be defined first based on users' priori knowledge. Root cells in flowSpy workflow are the initial cells of the trajectory tree. So it can be set using one vertex node of the tree or a cluster of cells with specific antibodies combination. Intermediate state evaluation is also involved in the pseudotime module. Leaf cells are defined by the end node of trajectory tree or the end stage of the experiment. Intermediate state cells are cells with higher betweenness in the graph built on cell-cell connection, which play an important role between the connection of root cells and leaf cells.
- **Visualization.** The visualization module can provide clear and concise visualization of FCS data in an effective and easy-to-comprehend manner. flowSpy package offers various plotting functions to

generate customizable and publication-quality plots. Two-dimensional or three-dimensional plot can fit most requirement from dimensionality reduction results. And tree-based plot can visualize cell trajectory as force-directed layout tree. Other special plots such as heatmap and violin plot are also provided in flowSpy.

## flowSpy workflow



## Trajectory



## Analysis

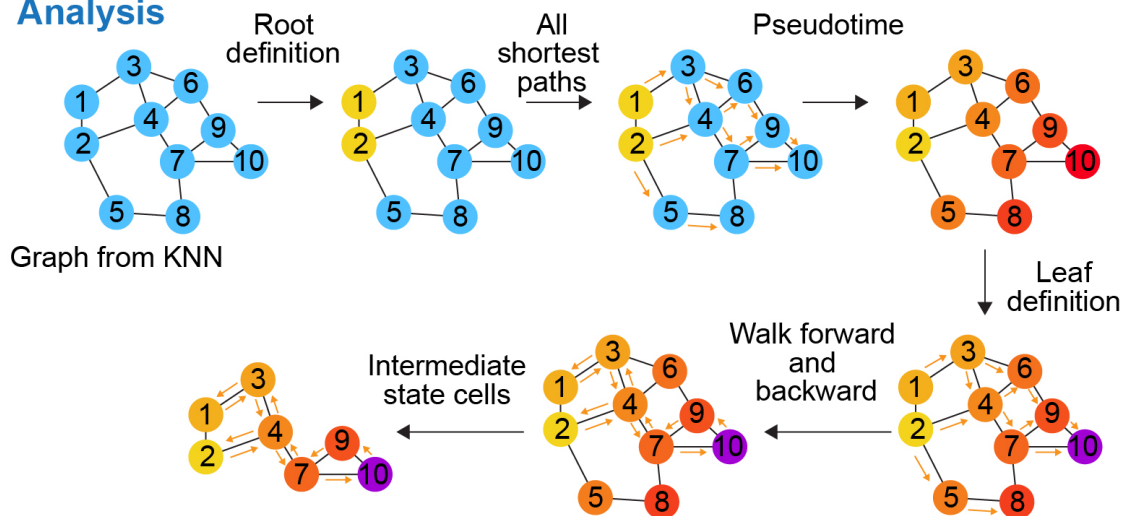


Fig. 1. Workflow of flowSpy

```

# Loading packages
suppressMessages({
  library(ggplot2)
  library(flowSpy)
  library(flowCore)
  library(stringr)
})

# Read fcs files
fcs.path <- system.file("extdata", package = "flowSpy")
fcs.files <- list.files(fcs.path, pattern = '.FCS$', full = TRUE)

fcs.data <- runExprsMerge(fcs.files, comp = FALSE, transformMethod = "none")

# Refine colnames of fcs data
recol <- c(`FITC-A<CD43>` = "CD43", `APC-A<CD34>` = "CD34",

```

```

    `BV421-A<CD90>` = "CD90", `BV510-A<CD45RA>` = "CD45RA",
    `BV605-A<CD31>` = "CD31", `BV650-A<CD49f>` = "CD49f",
    `BV 735-A<CD73>` = "CD73", `BV786-A<CD45>` = "CD45",
    `PE-A<FLK1>` = "FLK1", `PE-Cy7-A<CD38>` = "CD38")
colnames(fcs.data)[match(names(recol), colnames(fcs.data))] = recol
fcs.data <- fcs.data[, recol]

day.list <- c("D0", "D2", "D4", "D6", "D8", "D10")
meta.data <- data.frame(cell = rownames(fcs.data),
                        stage = str_replace(rownames(fcs.data), regex(".FCS.+"), "") )
meta.data$stage <- factor(as.character(meta.data$stage), levels = day.list)

markers <- c("CD43", "CD34", "CD90", "CD45RA", "CD31", "CD49f", "CD73", "CD45", "FLK1", "CD38")

# Build the FSPY object
fspy <- createFSPY(raw.data = fcs.data, markers = markers,
                  meta.data = meta.data,
                  normalization.method = "log",
                  verbose = TRUE)

```

```
## 2020-02-19 15:23:28 [INFO] Number of cells in processing: 600
```

```
## 2020-02-19 15:23:28 [INFO] rownames of meta.data and raw.data will be named using column
```

```
## 2020-02-19 15:23:28 [INFO] Index of markers in processing
```

```
## 2020-02-19 15:23:28 [INFO] Creating FSPY object.
```

```
## 2020-02-19 15:23:28 [INFO] Determining normalization factors
```

```
## 2020-02-19 15:23:28 [INFO] Normalization and log-transformation.
```

```
## 2020-02-19 15:23:28 [INFO] Build FSPY object succeed
```

```
# See information
fspy
```

```
## FSPY Information:
## Input cell number: 600 cells
## Enroll marker number: 10 markers
## Cells after downsampling: 600 markers
```

```
# Cluster cells by SOM algorithm
# Set random seed to make results reproducible
set.seed(1)
fspy <- runCluster(fspy, cluster.method = "som")
```

```
## Mapping data to SOM
```

```
# Do not perform downsampling
set.seed(1)
fspy <- processingCluster(fspy)
```

```
# run Principal Component Analysis (PCA)
fspy <- runFastPCA(fspy)

# run t-Distributed Stochastic Neighbor Embedding (tSNE)
fspy <- runTSNE(fspy)

# run Diffusion map
fspy <- runDiffusionMap(fspy)

# run Uniform Manifold Approximation and Projection (UMAP)
fspy <- runUMAP(fspy)

# build minimum spanning tree based on tsne
fspy <- buildTree(fspy, dim.type = "tsne", dim.use = 1:2)

# DEGs of different branch
diff.list <- runDiff(fspy)

# define root cells
fspy <- defRootCells(fspy, root.cells = c(28,26))

# run pseudotime
fspy <- runPseudotime(fspy, verbose = TRUE, dim.type = "raw")
```

```
## 2020-02-19 15:23:34 [INFO] Calculating Pseudotime.
```

```
## 2020-02-19 15:23:34 [INFO] Pseudotime exists in meta.data, it will be replaced.
```

```
## 2020-02-19 15:23:34 [INFO] The log data will be used to calculate trajectory
```

```
## 2020-02-19 15:23:34 [INFO] Calculating Pseudotime completed.
```

```
# define leaf cells
fspy <- defLeafCells(fspy, leaf.cells = c(27, 13), verbose = TRUE)
```

```
## 2020-02-19 15:23:34 [INFO] 37 cells will be added to leaf.cells .
```

```
# run walk between root cells and leaf cells
fspy <- runWalk(fspy, verbose = TRUE)
```

```
## 2020-02-19 15:23:34 [INFO] Calculating walk between root.cells and leaf.cells .
```

```
## 2020-02-19 15:23:34 [INFO] Generating an adjacency matrix.
```

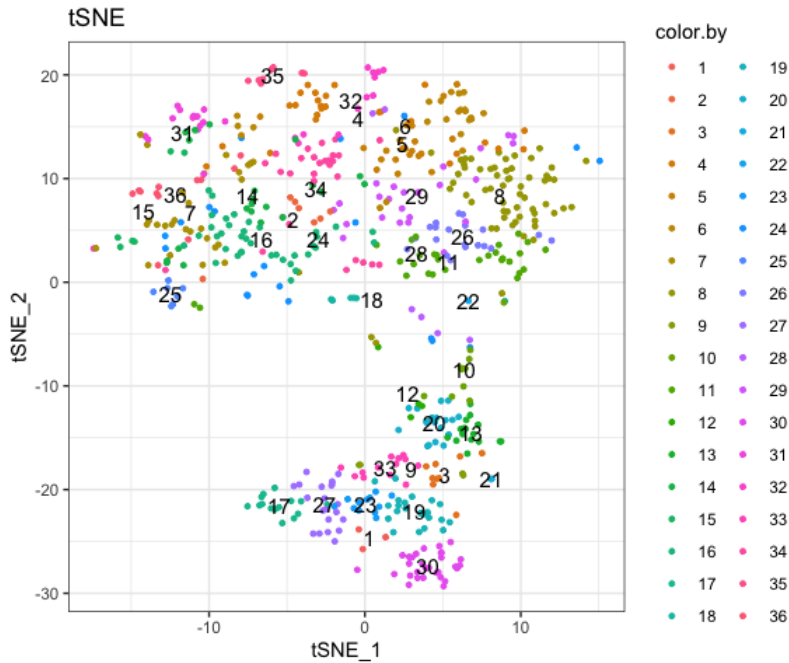
```
## 2020-02-19 15:23:34 [INFO] Walk forward.
```

```
## 2020-02-19 15:23:34 [INFO] Calculating walk completed.
```

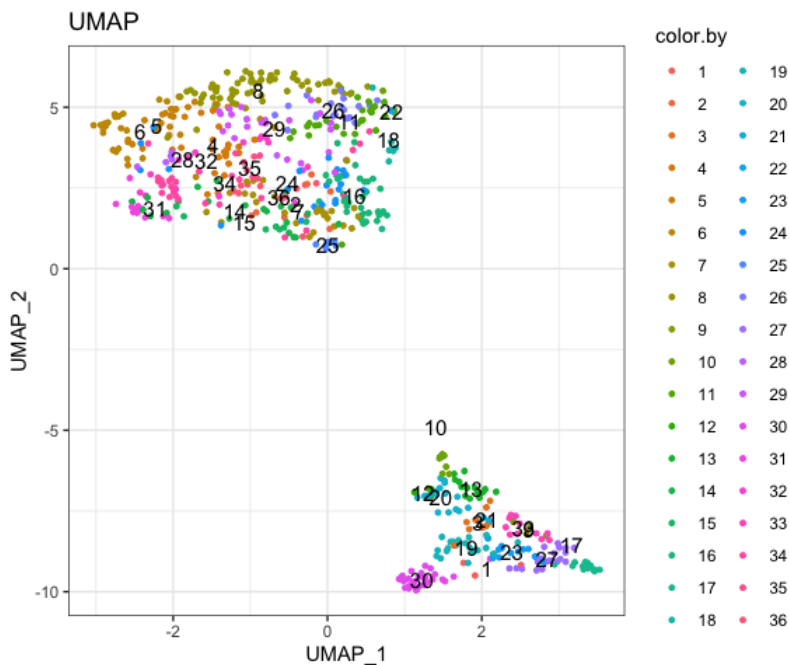
```
# Save object
if (FALSE) {
  save(fspy, file = "Path to you output directory")
}
```

```
##### Visualization
```

```
# Plot 2D tSNE. And cells are colored by cluster id  
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"), color.by = "cluster.id",  
alpha = 1, main = "tSNE", category = "categorical", show.cluser.id = TRUE)
```

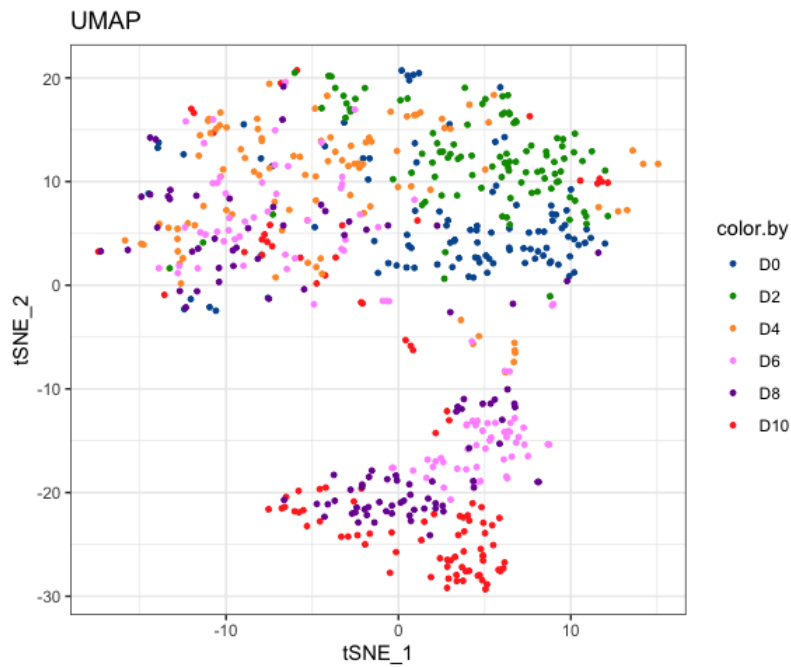


```
# Plot 2D UMAP. And cells are colored by cluster id  
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), color.by = "cluster.id",  
alpha = 1, main = "UMAP", category = "categorical", show.cluser.id = TRUE)
```

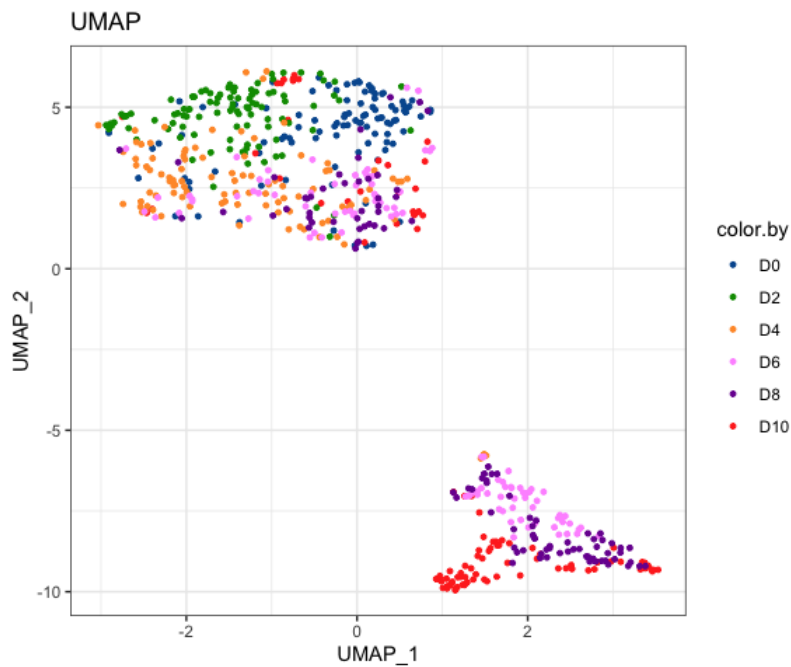


```
# Plot 2D tSNE. And cells are colored by branch id  
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"), color.by = "branch.id",  
alpha = 1, main = "tSNE", category = "categorical", show.cluser.id = TRUE)
```



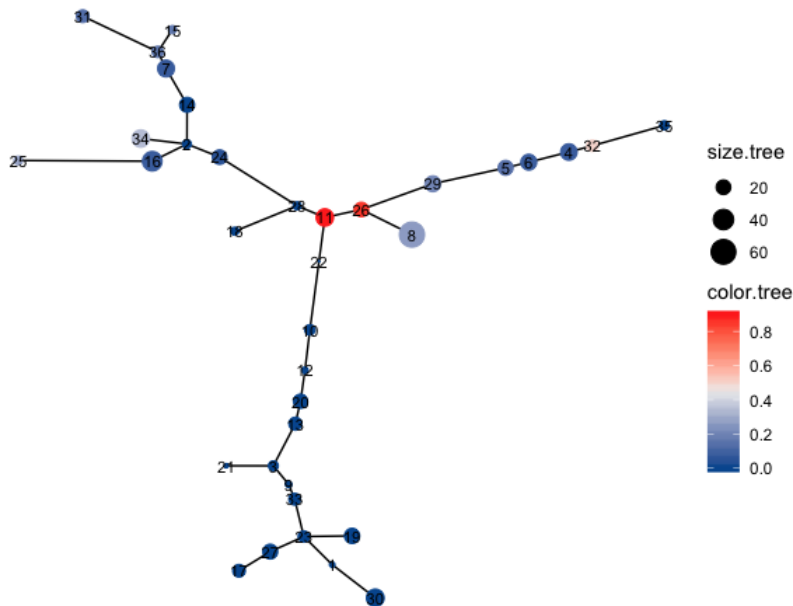


```
# Plot 2D UMAP. And cells are colored by stage
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), color.by = "stage",
       alpha = 1, main = "UMAP", category = "categorical") +
  scale_color_manual(values = c("#00599F", "#009900", "#FF9933",
                                "#FF99FF", "#7A06A0", "#FF3222"))
```



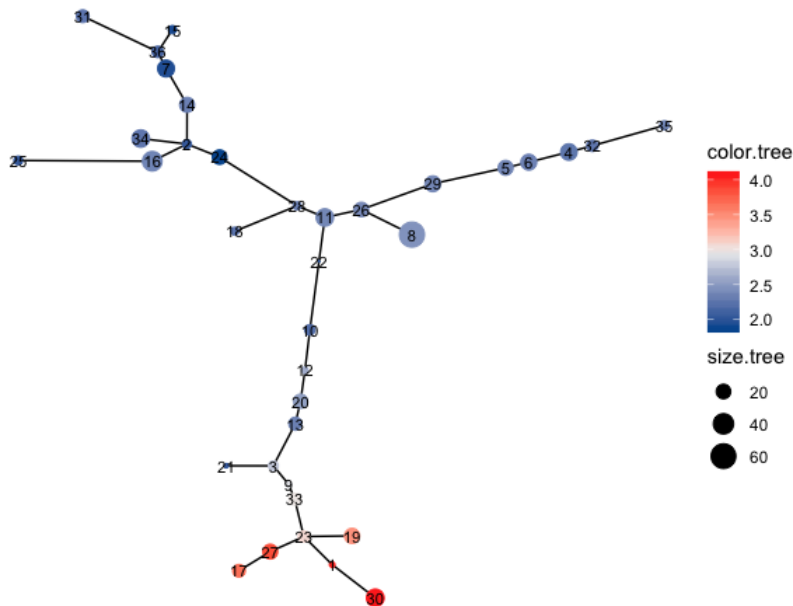
```
# Tree plot
plotTree(fspy, color.by = "D0.percent", show.node.name = TRUE, cex.size = 1) +
  scale_colour_gradientn(colors = c("#00599F", "#EEEEEE", "#FF3222"))
```

Tree plot, color.by: D0.percent, size.by: cell.number



```
plotTree(fspy, color.by = "CD43", show.node.name = TRUE, cex.size = 1) +  
  scale_colour_gradientn(colors = c("#00599F", "#EEEEEE", "#FF3222"))
```

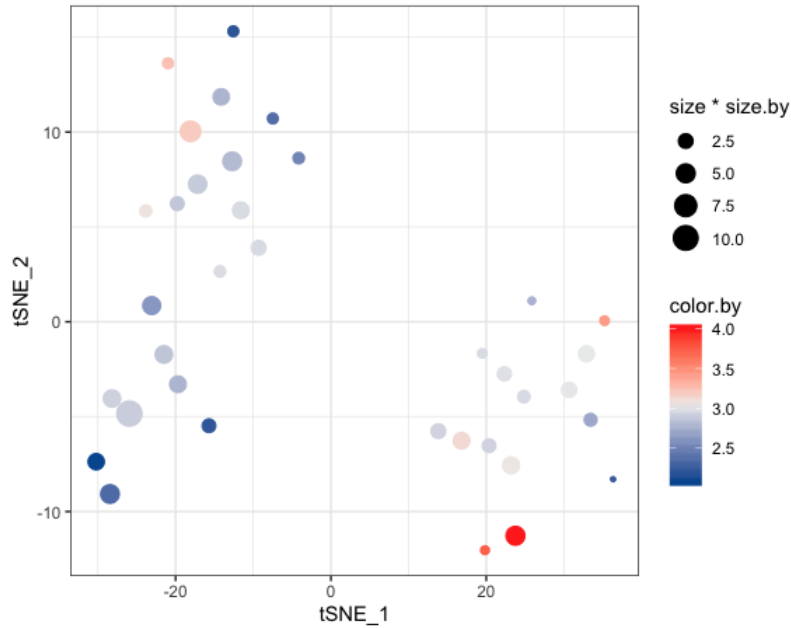
Tree plot, color.by: CD43, size.by: cell.number



```
# plot clusters  
plotCluster(fspy, item.use = c("tSNE_1", "tSNE_2"), category = "numeric",  
  size = 100, color.by = "CD45RA") +  
  scale_colour_gradientn(colors = c("#00599F", "#EEEEEE", "#FF3222"))
```

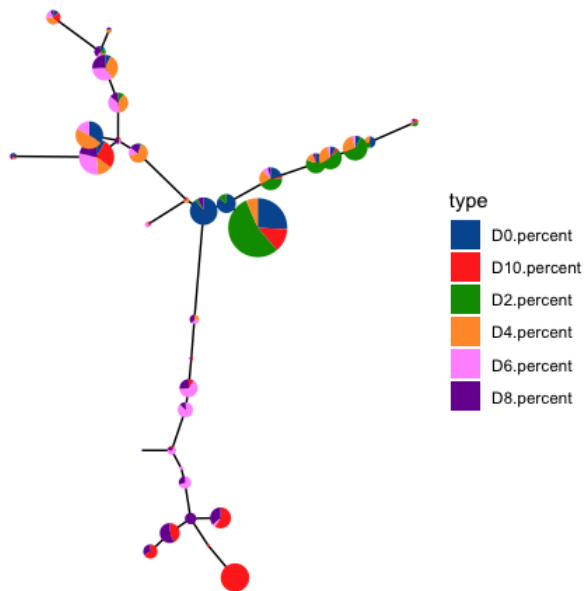


2D plot of cluster in FSPY



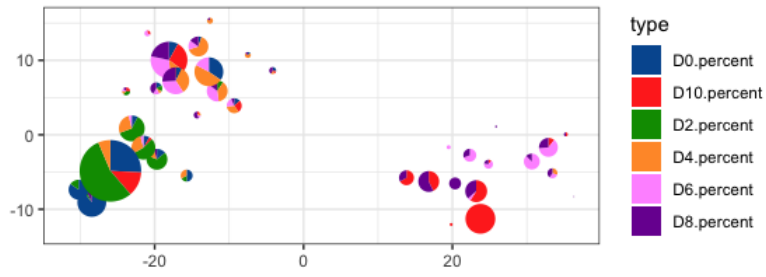
```
# plot pie tree
plotPieTree(fspy, cex.size = 3, size.by.cell.number = TRUE) +
  scale_fill_manual(values = c("#00599F", "#FF3222", "#009900",
    "#FF9933", "#FF99FF", "#7A06A0"))
```

Pie Tree Plot, size by cell.number

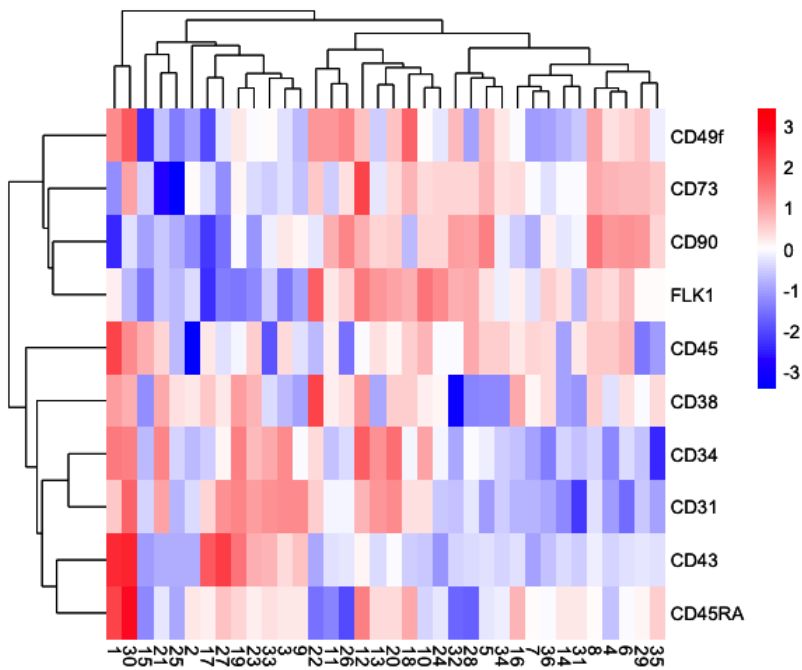


```
# plot pie cluster
plotPieCluster(fspy, item.use = c("tSNE_1", "tSNE_2"), cex.size = 40) +
  scale_fill_manual(values = c("#00599F", "#FF3222", "#009900",
    "#FF9933", "#FF99FF", "#7A06A0"))
```

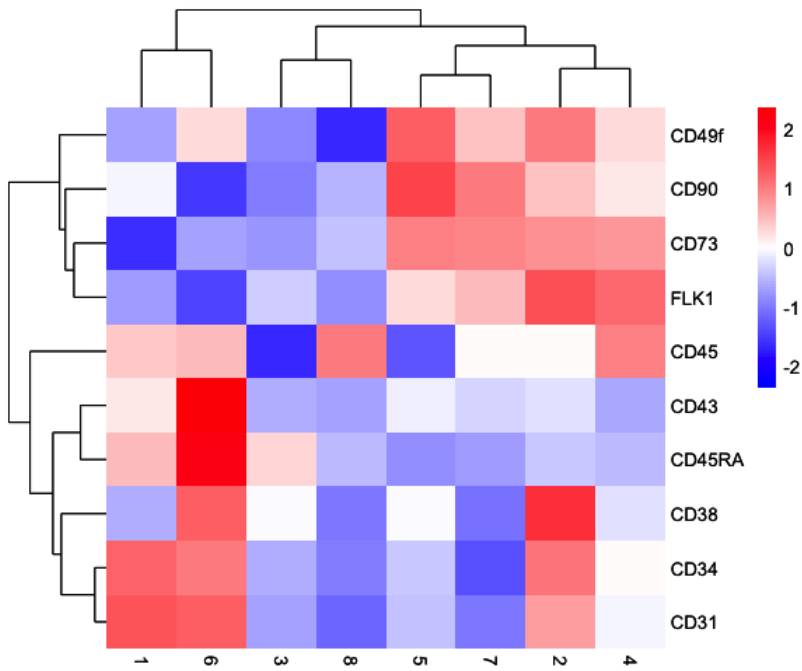
2D pie plot of FSPY



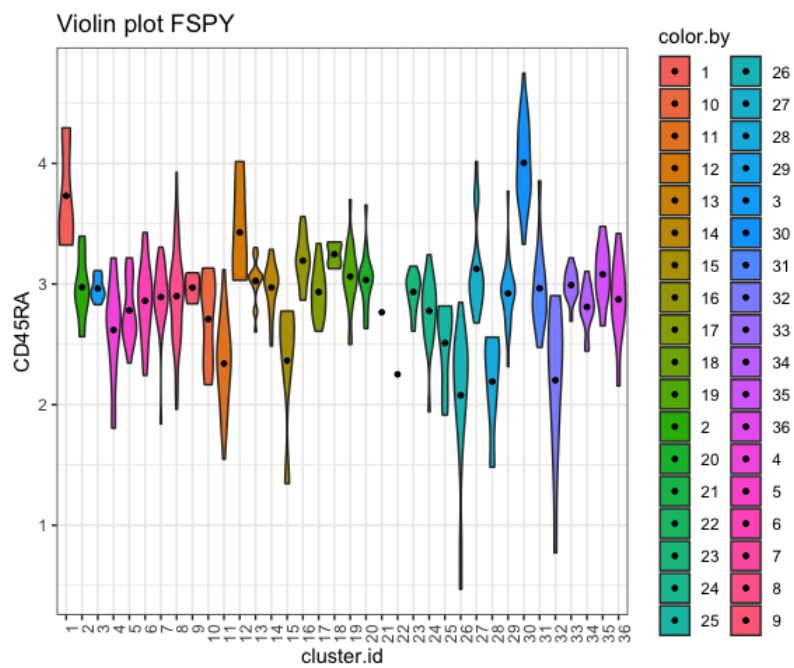
```
# plot heatmap of cluster  
plotClusterHeatmap(fspy)
```



```
plotBranchHeatmap(fspy)
```

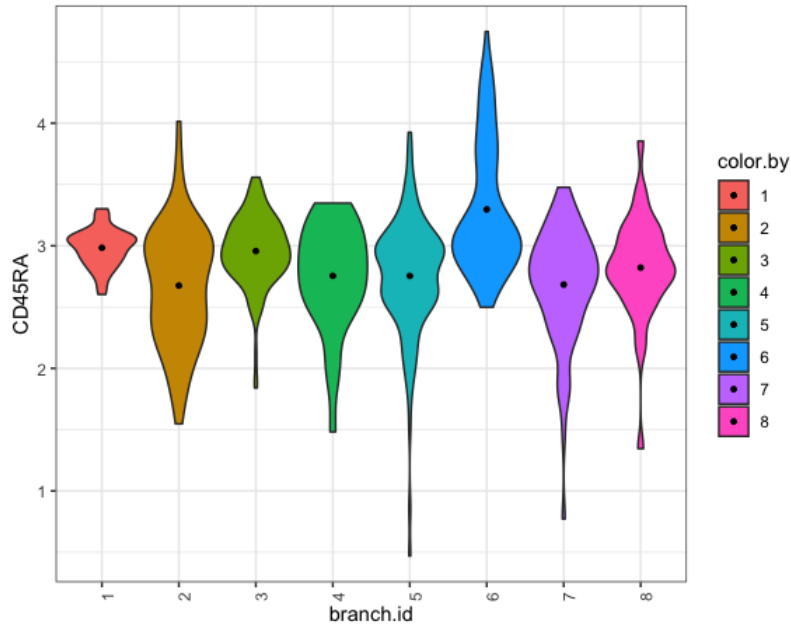


```
# Violin plot
plotViolin(fspy, color.by = "cluster.id", marker = "CD45RA", text.angle = 90)
```



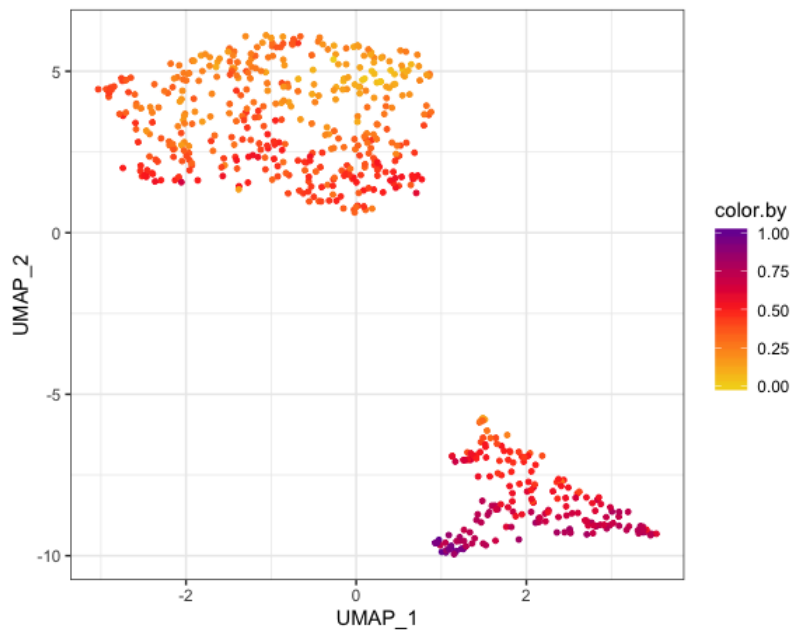
```
plotViolin(fspy, color.by = "branch.id", marker = "CD45RA", text.angle = 90)
```

Violin plot FSPY

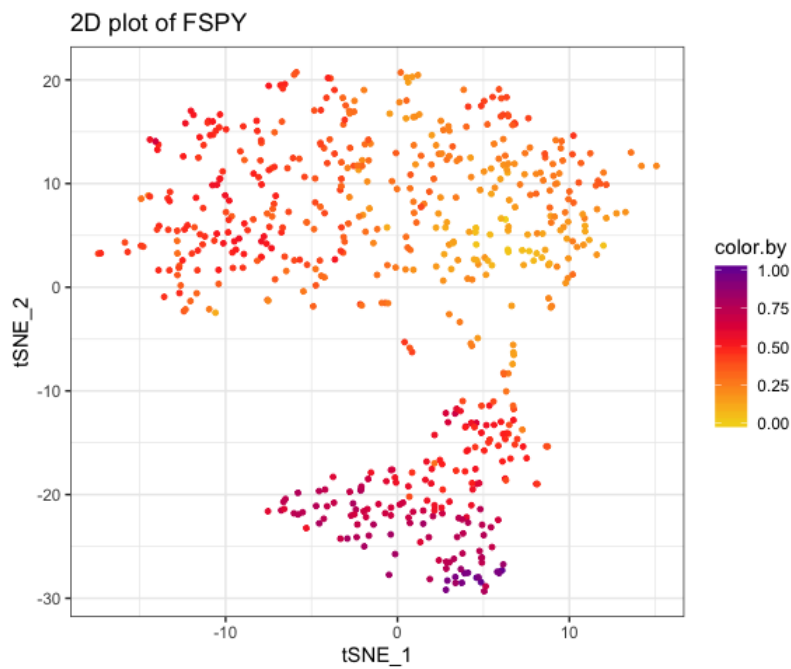


```
# UMAP plot colored by pseudotime
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), category = "numeric",
       size = 1, color.by = "pseudotime") +
scale_colour_gradientn(colors = c("#F4D31D", "#FF3222", "#7A06A0"))
```

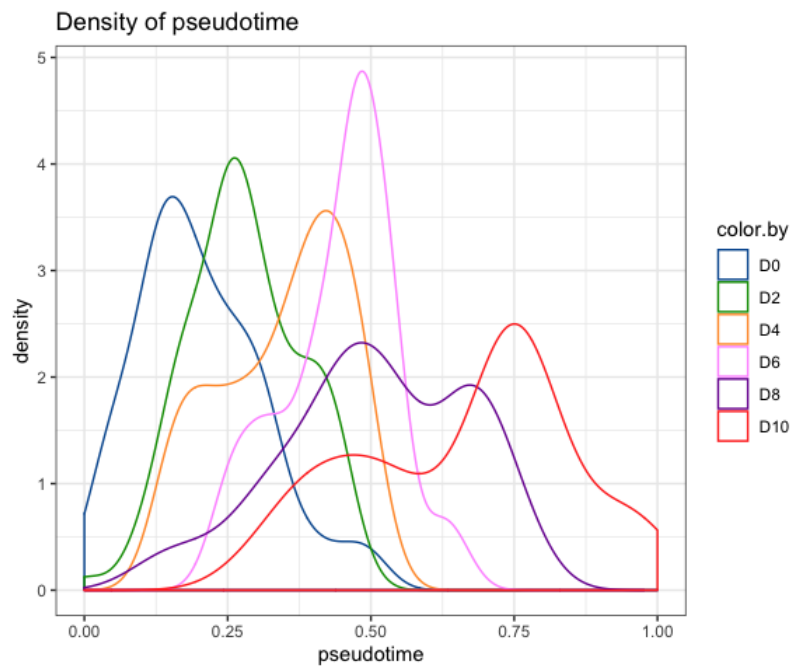
2D plot of FSPY



```
# tSNE plot colored by pseudotime
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"), category = "numeric",
       size = 1, color.by = "pseudotime") +
scale_colour_gradientn(colors = c("#F4D31D", "#FF3222", "#7A06A0"))
```

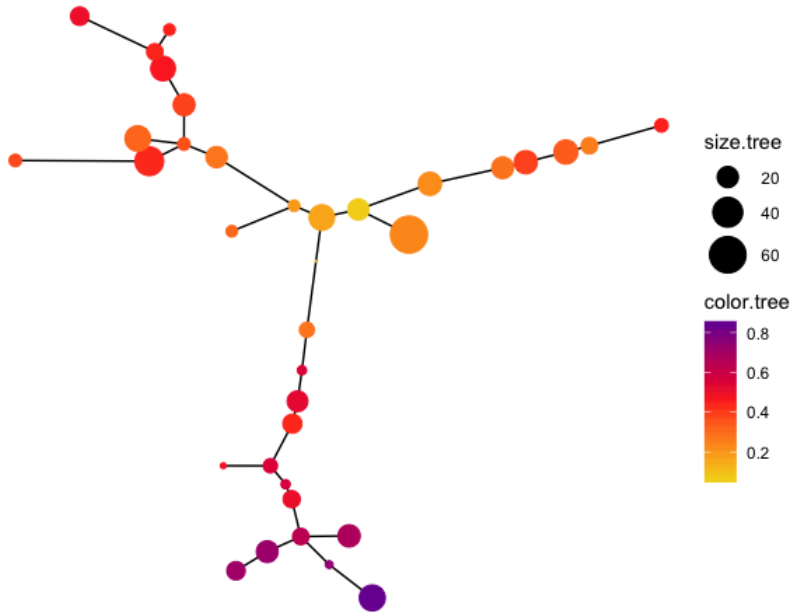


```
# density plot by different stage
plotPseudotimeDensity(fspy, adjust = 1) +
  scale_color_manual(values = c("#00599F", "#009900", "#FF9933",
                                "#FF99FF", "#7A06A0", "#FF3222"))
```

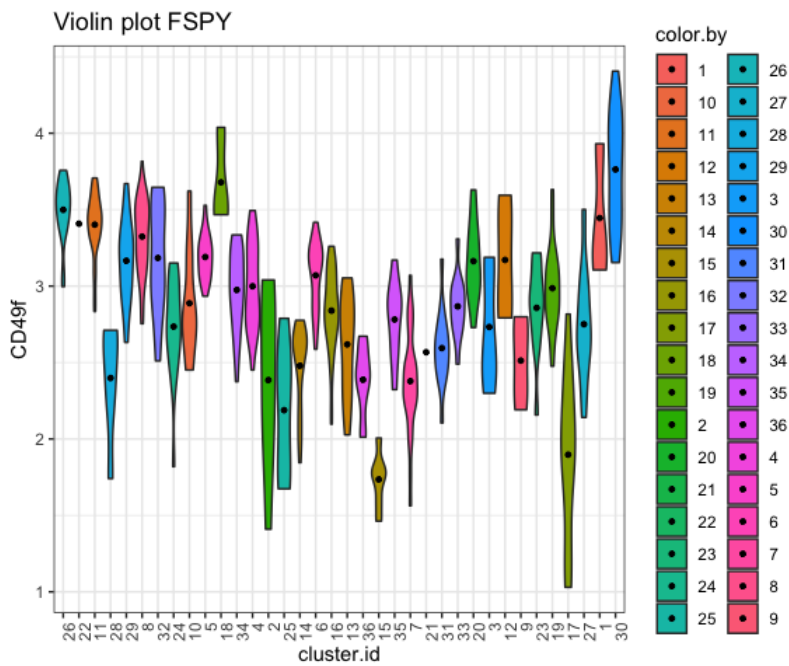


```
# Tree plot
plotTree(fspy, color.by = "pseudotime", cex.size = 1.5) +
  scale_colour_gradientn(colors = c("#F4D31D", "#FF3222", "#7A06A0"))
```

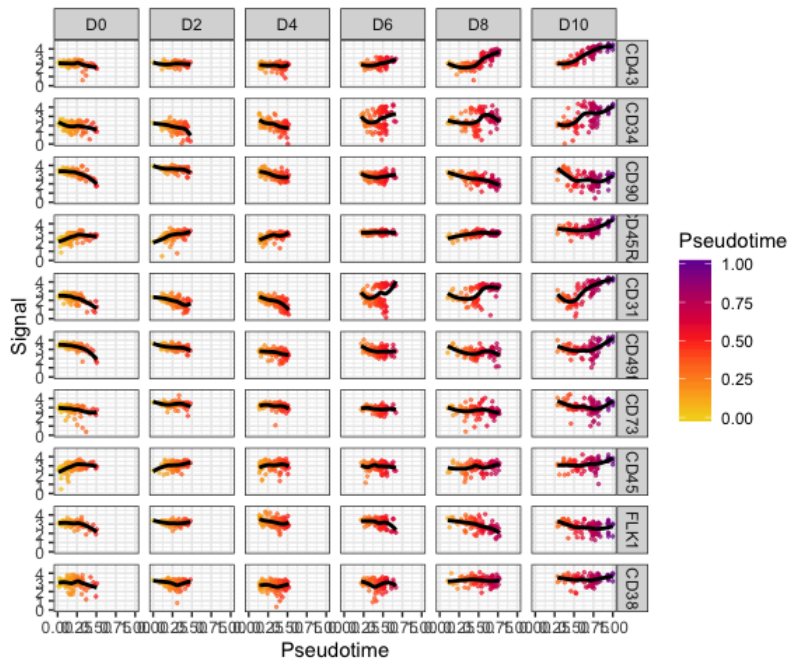
Tree plot, color.by: pseudotime, size.by: cell.number



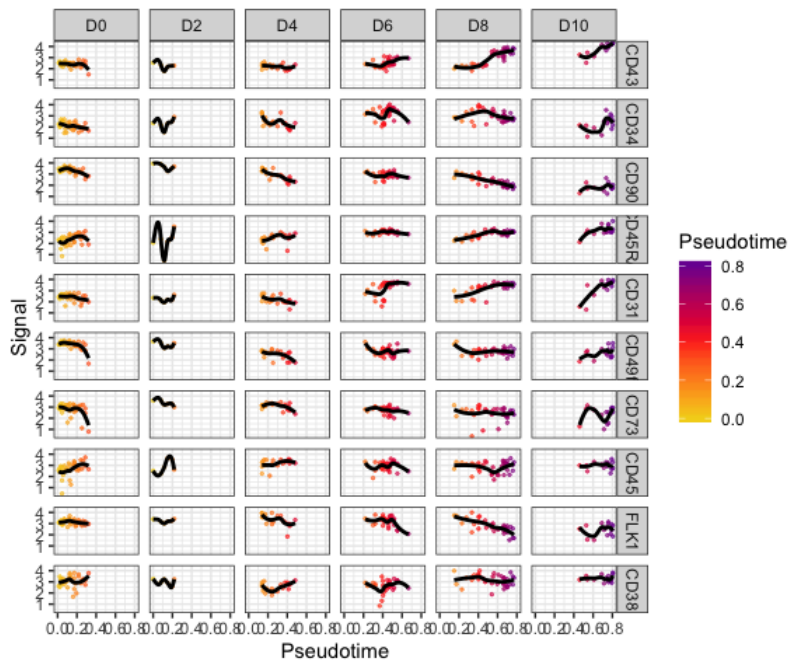
```
plotViolin(fspy, color.by = "cluster.id", order.by = "pseudotime",
            marker = "CD49f", text.angle = 90)
```



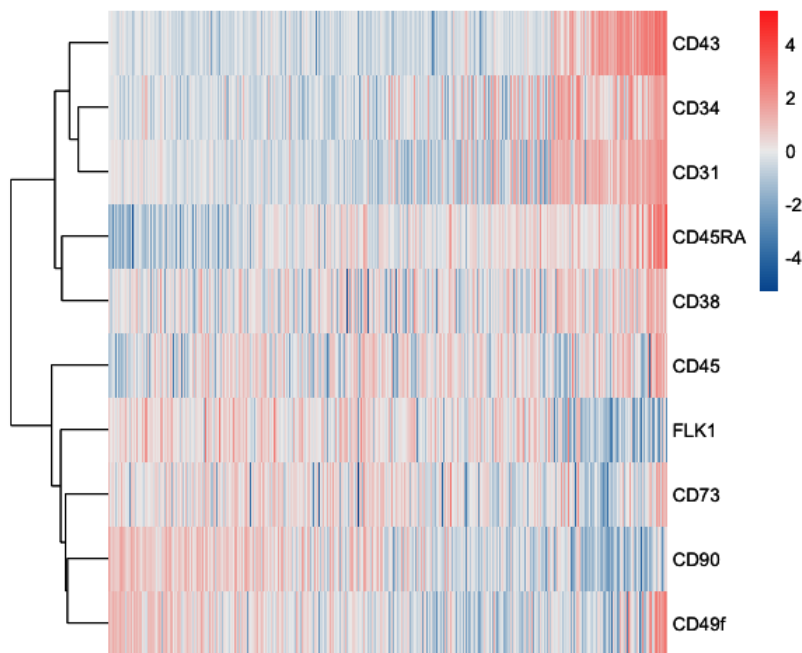
```
# trajectory value
plotPseudotimeTraj(fspy, var.cols = TRUE) +
  scale_colour_gradientn(colors = c("#F4D31D", "#FF3222", "#7A06A0"))
```



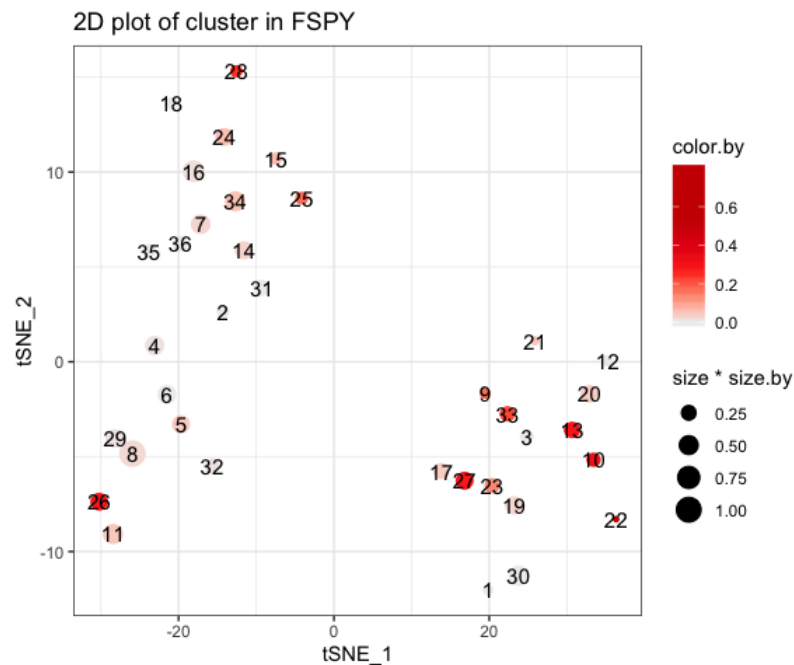
```
plotPseudotimeTraj(fspy, cutoff = 0.05, var.cols = TRUE) +
  scale_colour_gradientn(colors = c("#F4D31D", "#FF3222", "#7A06A0"))
```



```
plotHeatmap(fspy, downsize = 1000, cluster_rows = TRUE, clustering_method = "ward.D",
  color = colorRampPalette(c("#00599F", "#EEEEEE", "#FF3222"))(100))
```



```
# plot cluster
plotCluster(fspy, item.use = c("tSNE_1", "tSNE_2"), color.by = "traj.value.log",
            size = 10, show.cluser.id = TRUE, category = "numeric") +
scale_colour_gradientn(colors = c("#EEEEEE", "#FF3222", "#CC0000", "#CC0000"))
```



```
# Show session information
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Catalina 10.15.3
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
```



```

##
## attached base packages:
## [1] stats      graphics  grDevices utils      datasets  methods   base
##
## other attached packages:
## [1] stringr_1.4.0  flowCore_1.52.1 flowSpy_1.0.2  igraph_1.2.4.2
## [5] ggplot2_3.2.1
##
## loaded via a namespace (and not attached):
## [1] reticulate_1.13          R.utils_2.9.2
## [3] ks_1.11.6                RUnit_0.4.32
## [5] tidymodels_0.2.5        RSQlite_2.1.4
## [7] AnnotationDbi_1.48.0    grid_3.6.1
## [9] ranger_0.11.2           BiocParallel_1.20.0
## [11] Rtsne_0.15              scatterpie_0.1.4
## [13] munsell_0.5.0           destiny_2.15.0
## [15] umap_0.2.4.0            withr_2.1.2
## [17] colorspace_1.4-1       flowViz_1.50.0
## [19] Biobase_2.46.0         knitr_1.26
## [21] stats4_3.6.1           flowClust_3.24.0
## [23] robustbase_0.93-5      vcd_1.4-4
## [25] openCyto_1.24.0        VIM_4.8.0
## [27] TTR_0.23-6             labeling_0.3
## [29] GenomeInfoDbData_1.2.2 mnormt_1.5-5
## [31] polyclip_1.10-0        bit64_0.9-7
## [33] farver_2.0.1           pheatmap_1.0.12
## [35] flowWorkspace_3.34.0  vctr_0.2.0
## [37] xfun_0.11              ggthemes_4.2.0
## [39] R6_2.4.1               GenomeInfoDb_1.22.0
## [41] clue_0.3-57           RcppEigen_0.3.3.7.0
## [43] bitops_1.0-6          DelayedArray_0.12.0
## [45] assertthat_0.2.1      scales_1.1.0
## [47] nnet_7.3-12           gtable_0.3.0
## [49] sva_3.34.0            rlang_0.4.2
## [51] zeallot_0.1.0         genefilter_1.68.0
## [53] scatterplot3d_0.3-41  flowUtils_1.50.0
## [55] splines_3.6.1         lazyeval_0.2.2
## [57] hexbin_1.28.0         reshape2_1.4.3
## [59] BiocManager_1.30.10  yaml_2.2.0
## [61] abind_1.4-5           backports_1.1.5
## [63] IDPmisc_1.1.19       RBGL_1.62.1
## [65] tools_3.6.1          ellipsis_0.3.0
## [67] RColorBrewer_1.1-2   proxy_0.4-23
## [69] BiocGenerics_0.32.0  Rcpp_1.0.3
## [71] plyr_1.8.5           base64enc_0.1-3
## [73] zlibbioc_1.32.0      purrr_0.3.3
## [75] RCurl_1.95-4.12     FlowSOM_1.18.0
## [77] openssl_1.4.1       S4Vectors_0.24.1
## [79] zoo_1.8-6            SummarizedExperiment_1.16.0
## [81] haven_2.2.0          cluster_2.1.0
## [83] fda_2.4.8           magrittr_1.5
## [85] RSpectra_0.16-0     ncdfFlow_2.32.0
## [87] data.table_1.12.8    openxlsx_4.1.4
## [89] gmodels_2.18.1      lmtest_0.9-37
## [91] RANN_2.6.1          mvtnorm_1.0-11
## [93] matrixStats_0.55.0  hms_0.5.2
## [95] evaluate_0.14        xtable_1.8-4
## [97] smoother_1.1        XML_3.98-1.20
## [99] rio_0.5.16          mclust_5.4.5
## [101] readxl_1.3.1        IRanges_2.20.1
## [103] gridExtra_2.3       ggcyto_1.14.0
## [105] compiler_3.6.1     ellipse_0.4.1
## [107] tibble_2.1.3       flowStats_3.44.0
## [109] KernSmooth_2.23-16  crayon_1.3.4
## [111] R.oo_1.23.0        htmltools_0.4.0

```

```

## [113] mgcv_1.8-31                corpcor_1.6.9
## [115] pcaPP_1.9-73                tidyr_1.0.0
## [117] rrcov_1.4-9                 RcppParallel_4.4.4
## [119] DBI_1.1.0                   tweenr_1.0.1
## [121] MASS_7.3-51.4              boot_1.3-23
## [123] Matrix_1.2-18              car_3.0-5
## [125] R.methodsS3_1.7.1          gdata_2.18.0
## [127] parallel_3.6.1             GenomicRanges_1.38.0
## [129] forcats_0.4.0              pkgconfig_2.0.3
## [131] rvcheck_0.1.7              prettydoc_0.3.1
## [133] foreign_0.8-72             laeken_0.5.0
## [135] sp_1.3-2                    annotate_1.64.0
## [137] XVector_0.26.0             digest_0.6.23
## [139] tsne_0.1-3                  ConsensusClusterPlus_1.50.0
## [141] graph_1.64.0                rmarkdown_2.0
## [143] cellranger_1.1.0           curl_4.3
## [145] gtools_3.8.1                nlme_3.1-143
## [147] lifecycle_0.1.0            jsonlite_1.6
## [149] carData_3.0-3              BiocNeighbors_1.4.1
## [151] askpass_1.1                 limma_3.42.0
## [153] pillar_1.4.2                lattice_0.20-38
## [155] survival_3.1-8             DEoptimR_1.0-8
## [157] glue_1.3.1                  xts_0.11-2
## [159] zip_2.0.4                   bit_1.1-14
## [161] Rgraphviz_2.30.0           ggforce_0.3.1
## [163] class_7.3-15                stringi_1.4.3
## [165] blob_1.2.0                  CytoML_1.12.0
## [167] memoise_1.1.0              latticeExtra_0.6-28
## [169] dplyr_0.8.3                 e1071_1.7-3

```

## References

1. Hahne F, Arlt D, Saueremann M, Majety M, Poustka A, Wiemann S, Huber W: Statistical methods and software for the analysis of highthroughput reverse genetic assays using flow cytometry readouts. *Genome Biol* 2006, 7:R77.
2. Olsen LR, Leipold MD, Pedersen CB, Maecker HT: The anatomy of single cell mass cytometry data. *Cytometry A* 2019, 95:156-172.
3. Butler A, Hoffman P, Smibert P, Papalexi E, Satija R: Integrating single-cell transcriptomic data across different conditions, technologies, and species. *Nat Biotechnol* 2018, 36:411-420.
4. Trapnell C, Cacchiarelli D, Grimsby J, Pokharel P, Li S, Morse M, Lennon NJ, Livak KJ, Mikkelsen TS, Rinn JL: The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nat Biotechnol* 2014, 32:381-386.
5. Kiselev VY, Yiu A, Hemberg M: scmap: projection of single-cell RNA-seq data across data sets. *Nat Methods* 2018, 15:359-362.
6. Amir el AD, Davis KL, Tadmor MD, Simonds EF, Levine JH, Bendall SC, Shenfeld DK, Krishnaswamy S, Nolan GP, Pe'er D: viSNE enables visualization of high dimensional single-cell data and reveals phenotypic heterogeneity of leukemia. *Nat Biotechnol* 2013, 31:545-552.
7. Haghverdi L, Buettner F, Theis FJ: Diffusion maps for high-dimensional single-cell analysis of differentiation data. *Bioinformatics* 2015, 31:2989-2998.
8. Becht E, McInnes L, Healy J, Dutertre CA, Kwok IWH, Ng LG, Ginhoux F, Newell EW: Dimensionality reduction for visualizing single-cell data using UMAP. *Nat Biotechnol* 2018.
9. Wang L, Hoffman RA: Standardization, Calibration, and Control in Flow Cytometry. *Curr Protoc Cytom* 2017, 79:1 3 1-1 3 27.
10. Hahne F, LeMeur N, Brinkman RR, Ellis B, Haaland P, Sarkar D, Spidlen J, Strain E, Gentleman R: flowCore: a Bioconductor package for high throughput flow cytometry. *BMC Bioinformatics* 2009, 10:106.
11. Sarkar D, Le Meur N, Gentleman R: Using flowViz to visualize flow cytometry data. *Bioinformatics* 2008, 24:878-879.
12. Van Gassen S, Callebaut B, Van Helden MJ, Lambrecht BN, Demeester P, Dhaene T, Saeys Y: FlowSOM: Using self-organizing maps for visualization and interpretation of cytometry data. *Cytometry A* 2015, 87:636-645.

13. Qiu P, Simonds EF, Bendall SC, Gibbs KD, Jr., Bruggner RV, Linderman MD, Sachs K, Nolan GP, Plevritis SK: Extracting a cellular hierarchy from high-dimensional cytometry data with SPADE. *Nat Biotechnol* 2011, 29:886-891.
14. Chen H, Lau MC, Wong MT, Newell EW, Poidinger M, Chen J: Cytokit: A Bioconductor Package for an Integrated Mass Cytometry Data Analysis Pipeline. *PLoS Comput Biol* 2016, 12:e1005112.
15. Chattopadhyay PK, Winters AF, Lomas WE, 3rd, Laino AS, Woods DM: High-Parameter Single-Cell Analysis. *Annu Rev Anal Chem (Palo Alto Calif)* 2019, 12:411-430.
16. Bendall SC, Davis KL, Amir el AD, Tadmor MD, Simonds EF, Chen TJ, Shenfeld DK, Nolan GP, Pe'er D: Single-cell trajectory detection uncovers progression and regulatory coordination in human B cell development. *Cell* 2014, 157:714-725.
17. Nowicka M, Krieg C, Crowell HL, Weber LM, Hartmann FJ, Guglietta S, Becher B, Levesque MP, Robinson MD: CyTOF workflow: differential discovery in high-throughput high-dimensional cytometry datasets. *F1000Res* 2017, 6:748.