# Time-course workflow of flowSpy in use case 3 and 4

Yuting Dai

2019-09-09



## Introduction

To illustrate the usage of flowSpy on differential trajectory reconstruction of time-course FCS data, we used a flow cytometry dataset of ten-day hematopoietic differentiation from the hESC line HUES9 on the basis of some modification of the previous work [1]. By adding different cytokine combinations on different days, HUES9 cells (CD90+CD49f+ on Day 0, D0) were directionally differentiated into mesodermal cells (FLK1+, D4), hemogenic endothelium (CD34+CD31+CD43-, D6) and hematopoietic stem/progenitor cells (HSPCs, CD34+CD43+CD38-CD45RA-CD90+, D8) in succession (Fig. 4a and Additional file 1: Figure S4). Ten cell surface markers (CD90, CD49f, FLK1, CD34, CD31, CD73, CD43, CD45, CD45RA, and CD38) were used for the flow cytometry analysis to monitor the generation of these cells. In particular, the initial expression of CD31 and CD43 at D6 and D8, respectively, reflected the emergence of endothelial cells and the endothelial-to-hematopoietic transition (EHT) (Fig. 4a and Additional file 1: Figure S4). The aim of this use case was to reconstruct the cellular differentiation trajectory of HUES9 cells and identify the cell-of-origin of HSPCs using flowSpy.

This tutorial contains key steps of `flowSpy` time-course workflow, including how to calculate the pseudotime and how to define cell subsets and rebuild an FSPY object using flowSpy. This use case also provided a framework for time-course cytometric data analysis and might provide support for research on stem cell reprogramming.

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

#########################
# Read Flow Cytometry Data
# It can be downloaded via `git clone https://github.com/ytdai/flowSpy-dataset.git`
# fcs.path musted be modified based on the download directory from GitHub
fcs.path <- "../../flowSpy-dataset/FCS/usecase3_4/"
fcs.files <- paste0(fcs.path, "D", c(0,2,4,6,8,10), ".fcs")

########################################
#   Get the expression matrix from FCS file
########################################

set.seed(1)
fcs.data <- runExprsMerge(fcs.files, comp = F, transformMethod = "none", fixedNum = 2000)
```

```
# Refine colnames of fcs data
# for usecase 2
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]

# Build an FSPY object
# If you don't want to see the running log information, set verbose FALSE
day.list <- c("D0", "D2", "D4", "D6", "D8", "D10")
meta.data <- data.frame(cell = rownames(fcs.data),
                        stage = str_replace(rownames(fcs.data), regex("_.+"), "") )
meta.data$stage <- factor(as.character(meta.data$stage), levels = day.list)

markers <- c("CD43", "CD34", "CD90", "CD45RA", "CD31", "CD49f", "CD73", "CD45", "FLK1", "CD3

fspy <- createFSPY(raw.data = fcs.data, markers = markers,
                   meta.data = meta.data,
                   normalization.method = "log",
                   verbose = T)
```

```
## 2019-09-09 23:12:02 [INFO] Number of cells in processing: 12000
```

```
## 2019-09-09 23:12:02 [INFO] rownames of meta.data and raw.data will be named using column
```

```
## 2019-09-09 23:12:02 [INFO] Index of markers in processing
```

```
## 2019-09-09 23:12:02 [INFO] Creating FSPY object.
```

```
## 2019-09-09 23:12:02 [INFO] Determining normalization factors
```

```
## 2019-09-09 23:12:02 [INFO] Normalization and log-transformation.
```

```
## 2019-09-09 23:12:02 [INFO] Build FSPY object succeed
```

```
# Cluster cells by SOM algorithm
# Set random seed to make results reproducible
set.seed(80)
fspy <- runCluster(fspy, cluster.method = "som", xdim = 6, ydim = 6)
```

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

```
# Do not perform downsampling
set.seed(2)
fspy <- processingCluster(fspy, downsampling.size = 1)

# run Principal Component Analysis (PCA)
fspy <- runFastPCA(fspy, verbose = T)
```

```
## 2019-09-09 23:12:03 [INFO] Calculating PCA.
```

```
## 2019-09-09 23:12:03 [INFO] Calculating PCA completed.
```

```
# run t-Distributed Stochastic Neighbor Embedding (tSNE)
set.seed(1)
fspy <- runTSNE(fspy, verbose = T)
```

```
## 2019-09-09 23:12:03 [INFO] Calculating tSNE.
```

```
## 2019-09-09 23:13:09 [INFO] Calculating tSNE completed.
```

```
# run Diffusion map
fspy <- runDiffusionMap(fspy, verbose = T)
```

```
## 2019-09-09 23:13:09 [INFO] Calculating Diffusion Map.
```

```
## 2019-09-09 23:13:09 [INFO] Destiny determined an optimal global sigma: 0.899
```

```
## 2019-09-09 23:13:40 [INFO] Calculating Diffusion Map completed
```

```
# run Uniform Manifold Approximation and Projection (UMAP)
fspy <- runUMAP(fspy, verbose = T)
```

```
## 2019-09-09 23:13:40 [INFO] Calculating Umap.
```

```
## 2019-09-09 23:15:24 [INFO] Calculating Umap.
```

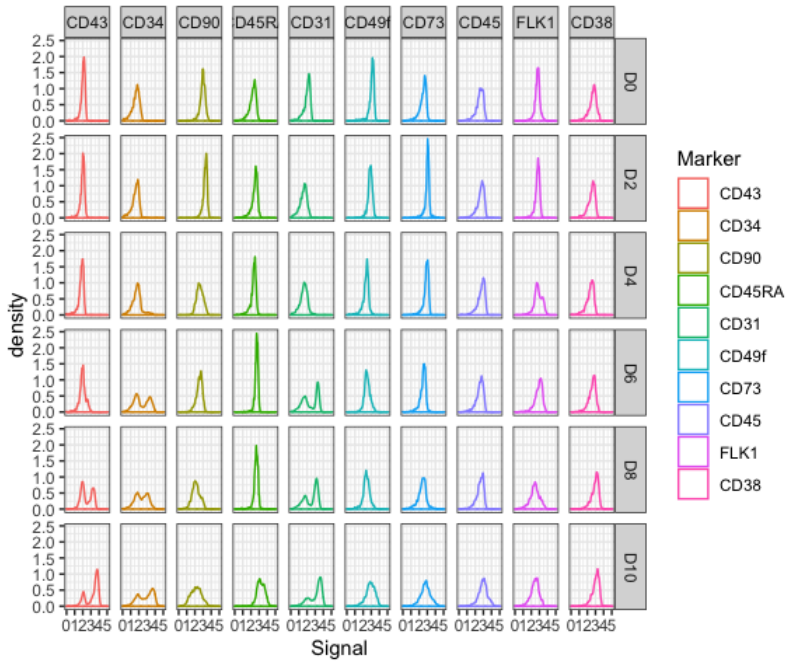
```
# build minimum spanning tree based on UMAP
fspy <- buildTree(fspy, dim.type = "umap", dim.use = 1:2, verbose = T)
```

```
## 2019-09-09 23:15:24 [INFO] Calculating buildTree.
```

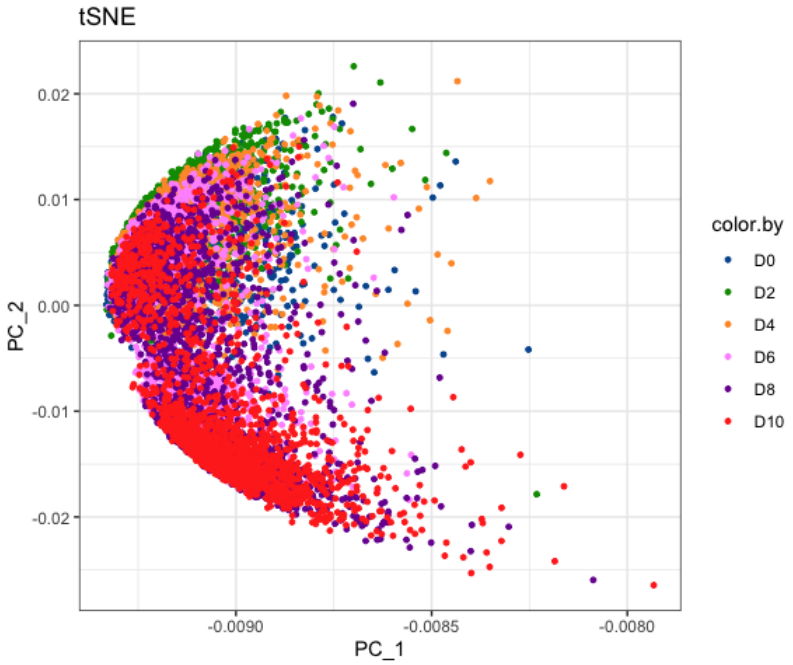
```
## 2019-09-09 23:15:24 [INFO] Initialization for root.cells and leaf cells
```

```
## 2019-09-09 23:15:24 [INFO] Calculating buildTree completed.
```

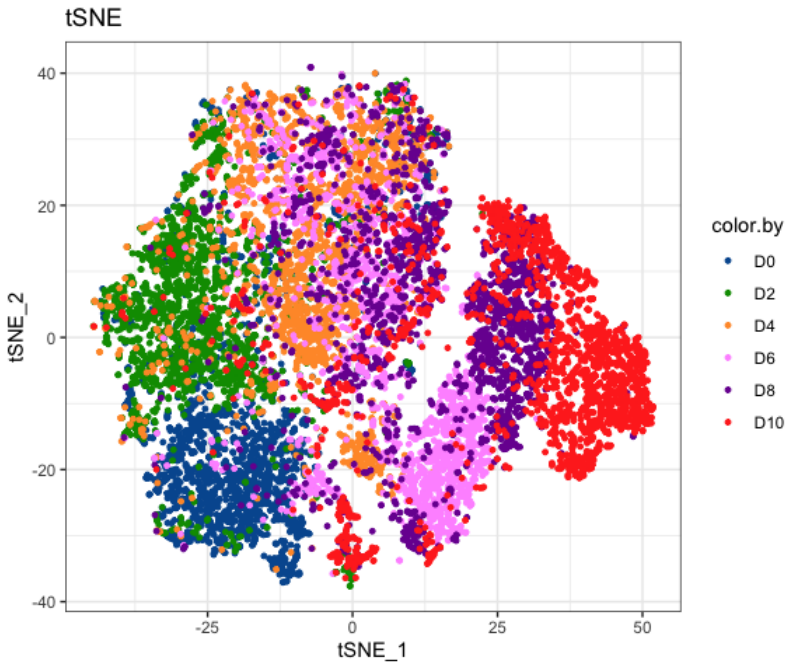
```
##########################################
# This is visualization module
##########################################

# Plot marker density
plotMarkerDensity(fspy)
```



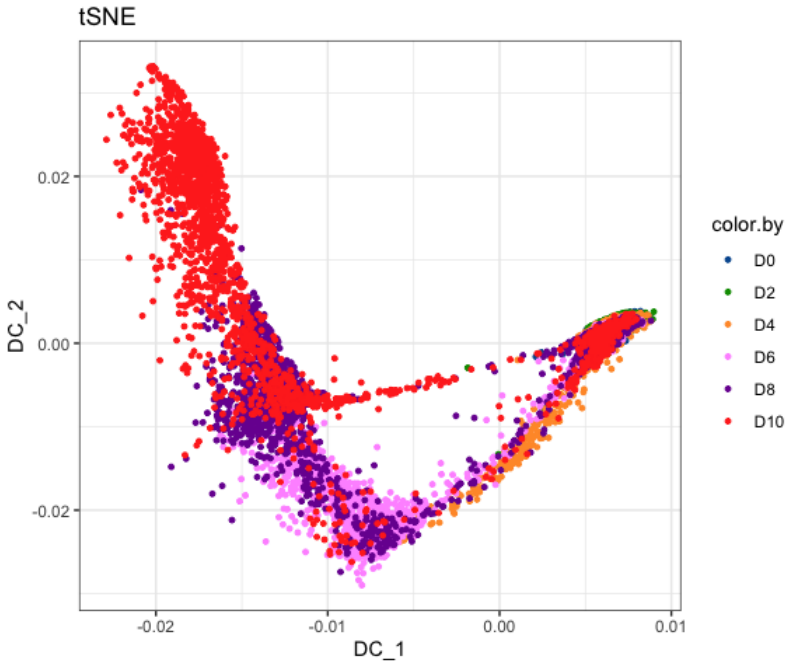

```
# Plot 2D PCA. And cells are colored by stage
plot2D(fspy, item.use = c("PC_1", "PC_2"), color.by = "stage",
       alpha = 1, main = "tSNE", category = "categorical") +
  scale_color_manual(values = c("#00599F","#009900","#FF9933",
                                "#FF99FF","#7A06A0","#FF3222"))
```



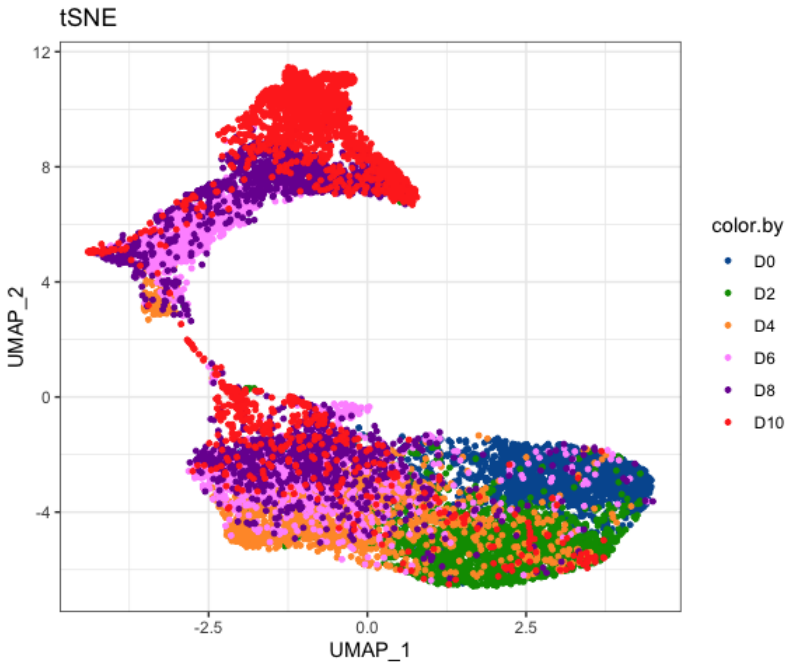

```
# Plot 2D tSNE. And cells are colored by stage
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"), color.by = "stage",
       alpha = 1, main = "tSNE", category = "categorical") +
  scale_color_manual(values = c("#00599F","#009900","#FF9933",
                                "#FF99FF","#7A06A0","#FF3222"))
```



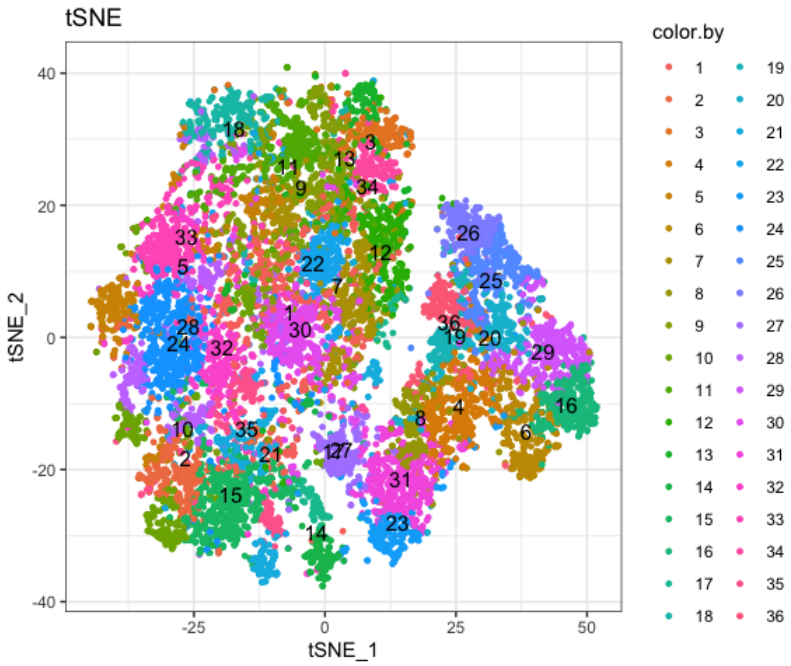

```
# Plot 2D diffusion maps. And cells are colored by stage
plot2D(fspy, item.use = c("DC_1", "DC_2"), color.by = "stage",
       alpha = 1, main = "tSNE", category = "categorical") +
  scale_color_manual(values = c("#00599F","#009900","#FF9933",
                                "#FF99FF","#7A06A0","#FF3222"))
```



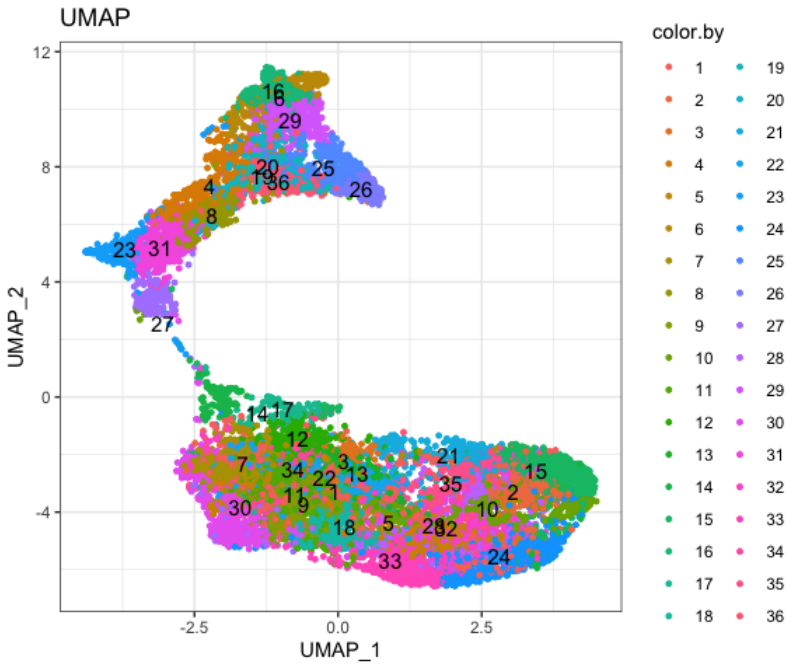

```
# Plot 2D UMAP. And cells are colored by stage
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), color.by = "stage",
       alpha = 1, main = "tSNE", category = "categorical") +
  scale_color_manual(values = c("#00599F","#009900","#FF9933",
                                "#FF99FF","#7A06A0","#FF3222"))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



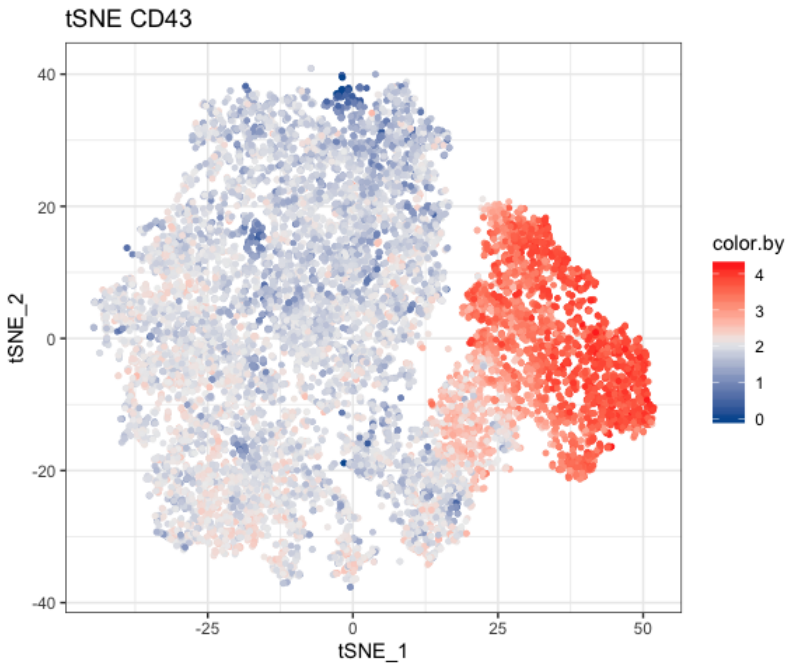

```
# Plot 2D tSNE. And cells are colored by cluster id
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"), color.by = "cluster.id",
       alpha = 1, main = "tSNE", category = "categorical", show.cluser.id = T)
```



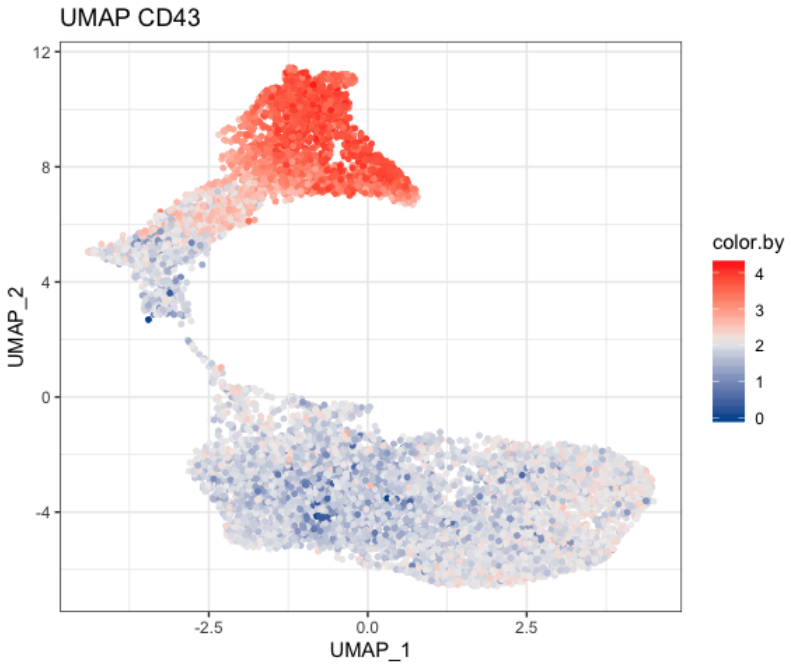

```
# Plot 2D UMAP. And cells are colored by cluster id
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), color.by = "cluster.id",
       alpha = 1, main = "UMAP", category = "categorical", show.cluser.id = T)
```



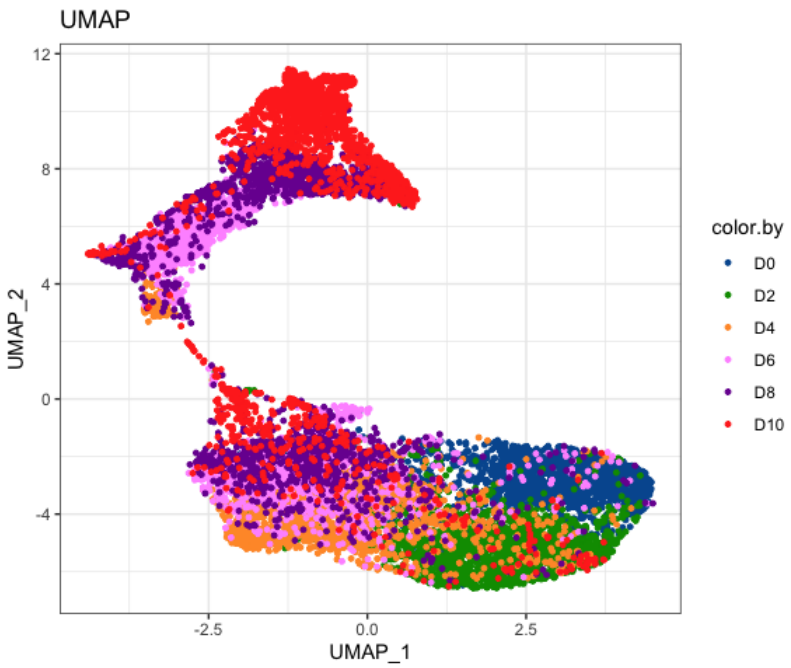

```
# Plot 2D tSNE. And cells are colored by CD43 markers expression
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"), color.by = "CD43",
               main = "tSNE CD43", category = "numeric")  +
 scale_colour_gradientn(colors = c("#00599F","#EEEEEE","#FF3222"))
```




```
# Plot 2D UMAP. And cells are colored by CD43 markers expression
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), color.by = "CD43",
               main = "UMAP CD43", category = "numeric")  +
 scale_colour_gradientn(colors = c("#00599F","#EEEEEE","#FF3222"))
```

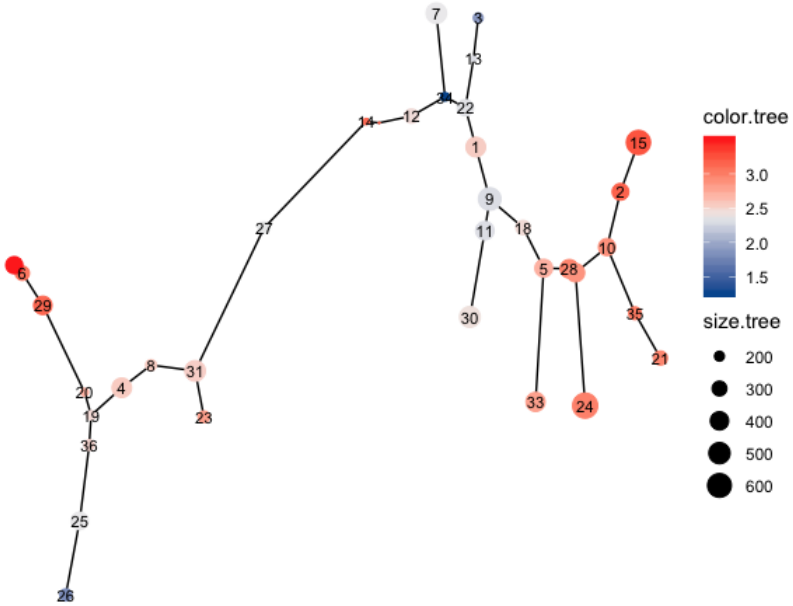



```
# Plot 2D UMAP. And cells are colored by stage
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), color.by = "stage",
       alpha = 1, main = "UMAP", category = "categorical") +
 scale_color_manual(values = c("#00599F","#009900","#FF9933",
                               "#FF99FF","#7A06A0","#FF3222"))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



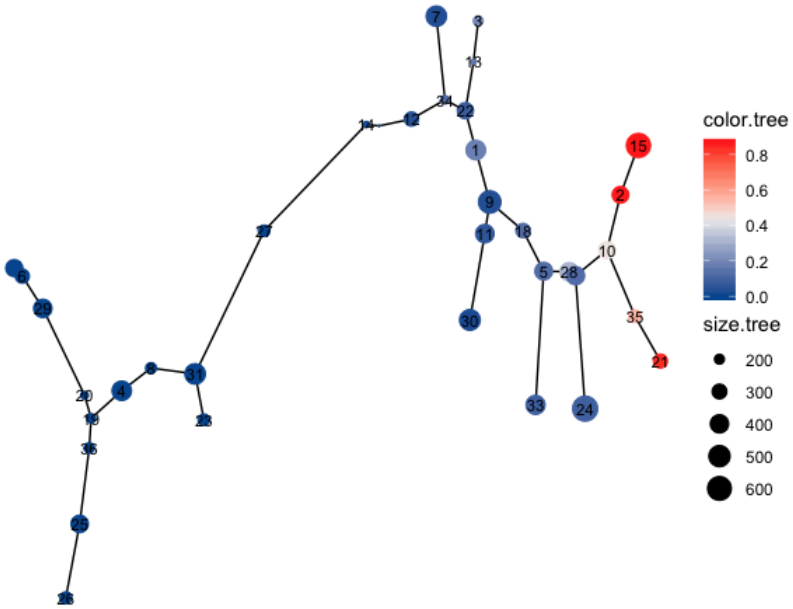

```
# Tree plot
plotTree(fspy, color.by = "CD49f", show.node.name = T, cex.size = 1) +
  scale_colour_gradientn(colors = c("#00599F", "#EEEEEE", "#FF3222"))
```

Tree plot, color.by: CD49f, size.by: cell.number

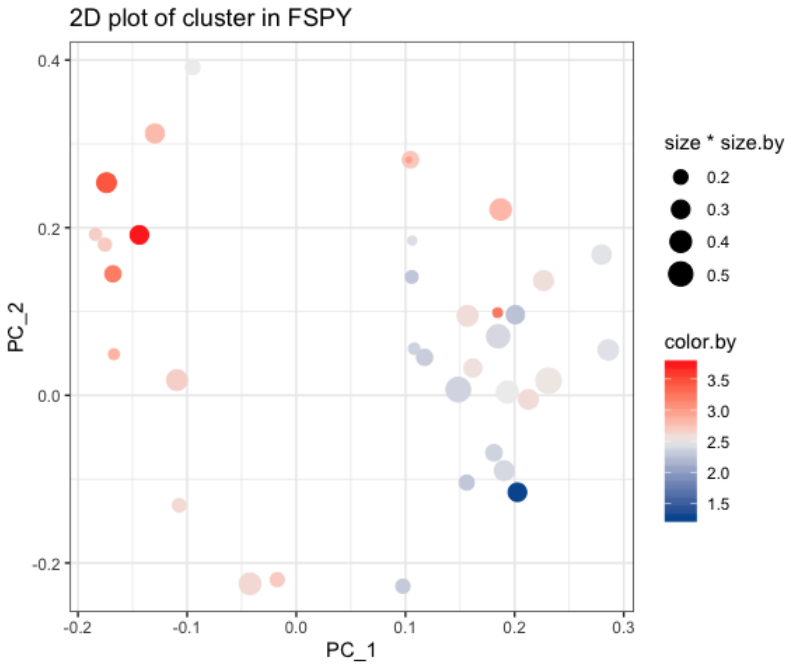



```
plotTree(fspy, color.by = "D0.percent", show.node.name = T, cex.size = 1) +
  scale_colour_gradientn(colors = c("#00599F", "#EEEEEE", "#FF3222"))
```

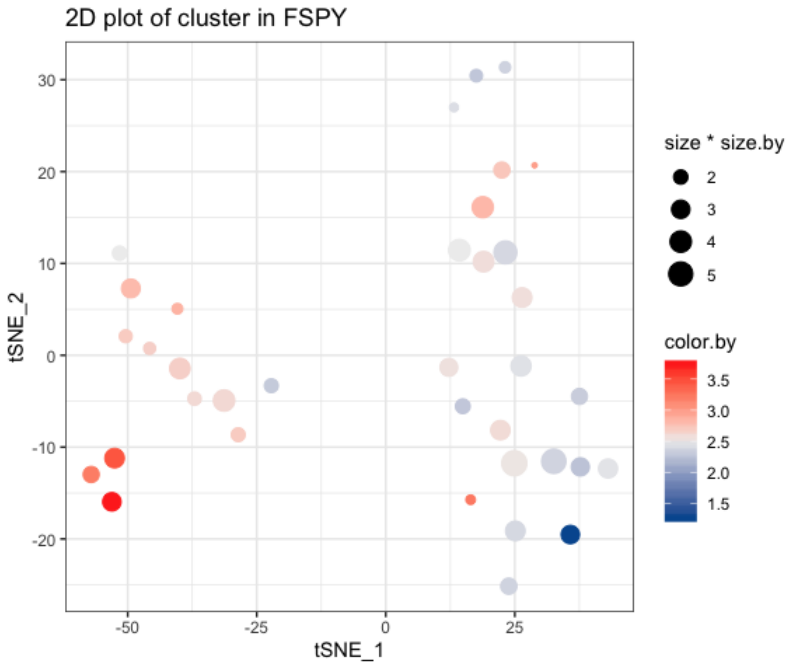
Tree plot, color.by: D0.percent, size.by: cell.number



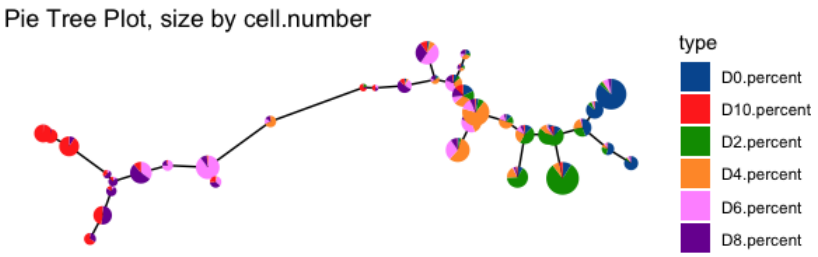

```
# plot clusters
plotCluster(fspy, item.use = c("PC_1", "PC_2"), category = "numeric",
            size = 10, color.by = "CD45RA") +
  scale_colour_gradientn(colors = c("#00599F", "#EEEEEE", "#FF3222"))
```



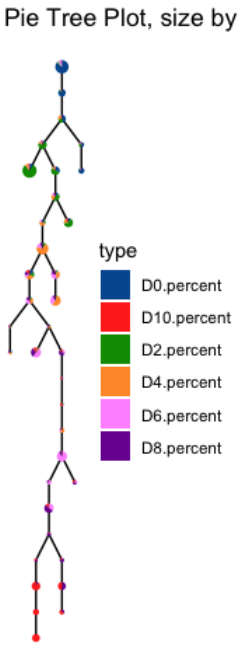

```
plotCluster(fspy, item.use = c("tSNE_1", "tSNE_2"), category = "numeric",
            size = 100, color.by = "CD45RA") +
  scale_colour_gradientn(colors = c("#00599F", "#EEEEEE", "#FF3222"))
```



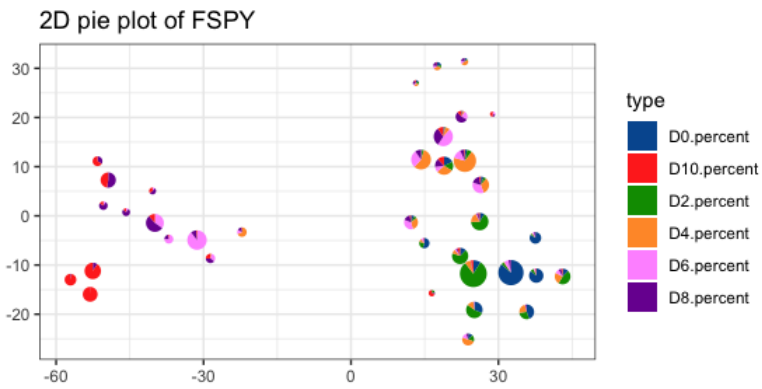

```
# plot pie tree
plotPieTree(fspy, cex.size = 3, size.by.cell.number = T) +
 scale_fill_manual(values = c("#00599F","#FF3222","#009900",
                              "#FF9933","#FF99FF","#7A06A0"))
```



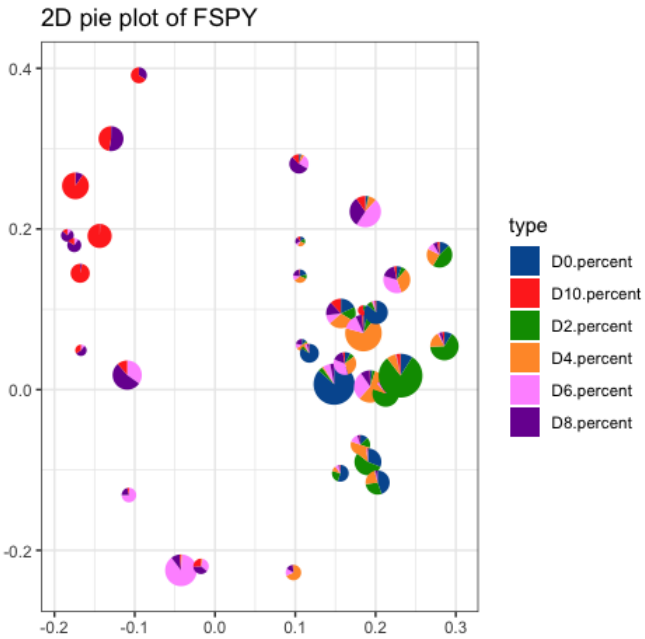

```
plotPieTree(fspy, cex.size = 5, size.by.cell.number = T, as.tree = T, root.id = 15) +
 scale_fill_manual(values = c("#00599F","#FF3222","#009900",
                              "#FF9933","#FF99FF","#7A06A0"))
```



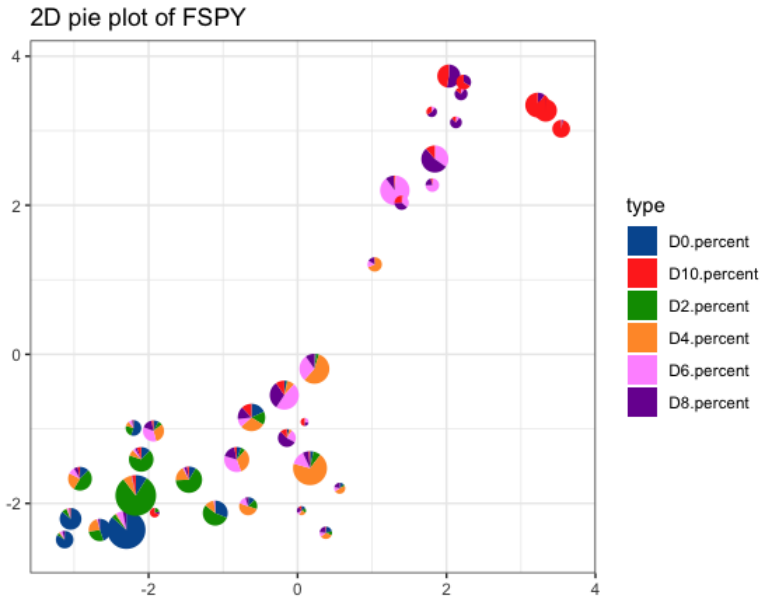

```
# plot pie cluster
plotPieCluster(fspy, item.use = c("tSNE_1", "tSNE_2"), cex.size = 50) +
 scale_fill_manual(values = c("#00599F","#FF3222","#009900",
                              "#FF9933","#FF99FF","#7A06A0"))
```




```
plotPieCluster(fspy, item.use = c("PC_1", "PC_2"), cex.size = 0.5) +
 scale_fill_manual(values = c("#00599F","#FF3222","#009900",
                              "#FF9933","#FF99FF","#7A06A0"))
```

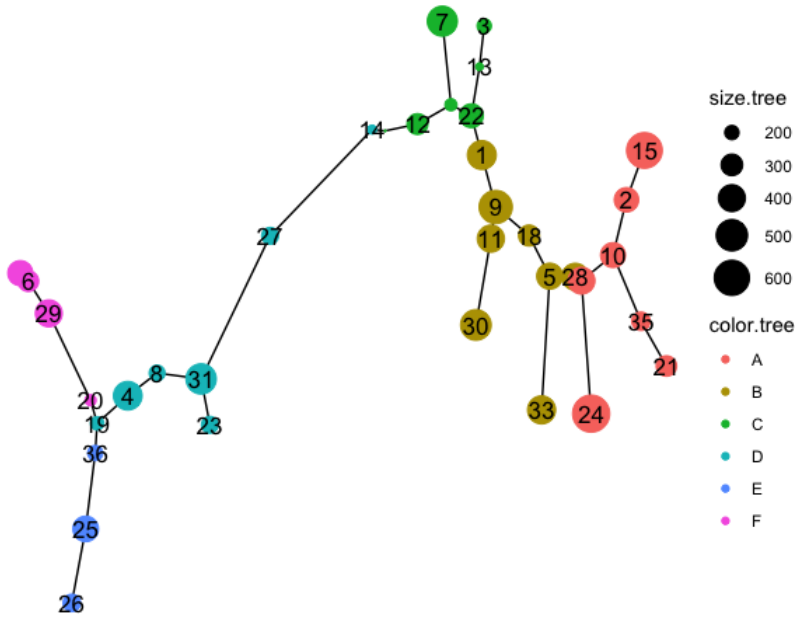



```
plotPieCluster(fspy, item.use = c("UMAP_1", "UMAP_2"), cex.size = 5) +
 scale_fill_manual(values = c("#00599F","#FF3222","#009900",
                              "#FF9933","#FF99FF","#7A06A0"))
```

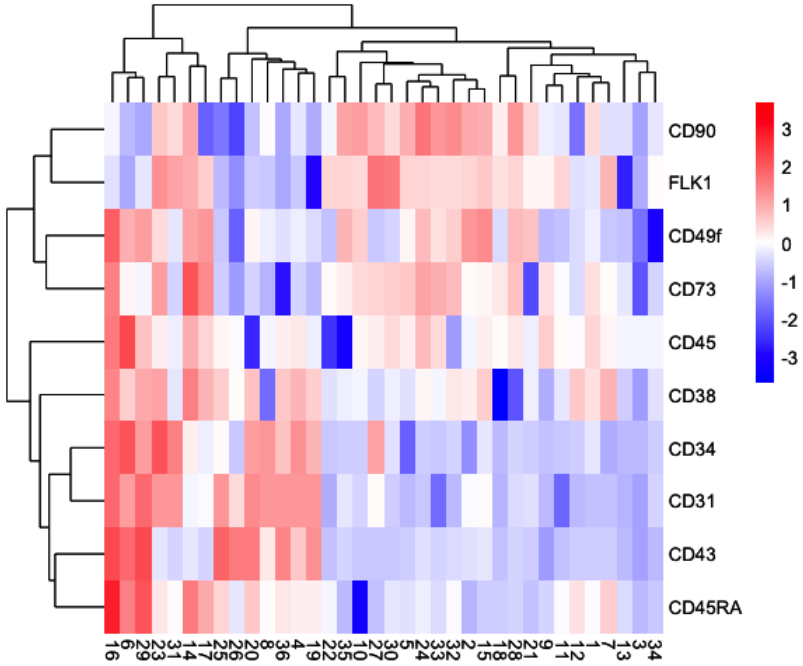



```
##############  Modify branch id
fspy@meta.data$branch.id[fspy@meta.data$branch.id %in% c(1)] = "C"
fspy@meta.data$branch.id[fspy@meta.data$branch.id %in% c(2)] = "E"
fspy@meta.data$branch.id[fspy@meta.data$branch.id %in% c(3)] = "D"
fspy@meta.data$branch.id[fspy@meta.data$branch.id %in% c(4)] = "F"
fspy@meta.data$branch.id[fspy@meta.data$branch.id %in% c(5)] = "B"
fspy@meta.data$branch.id[fspy@meta.data$branch.id %in% c(6)] = "A"

plotTree(fspy, color.by = "branch.id", show.node.name = T, cex.size = 1.5)
```

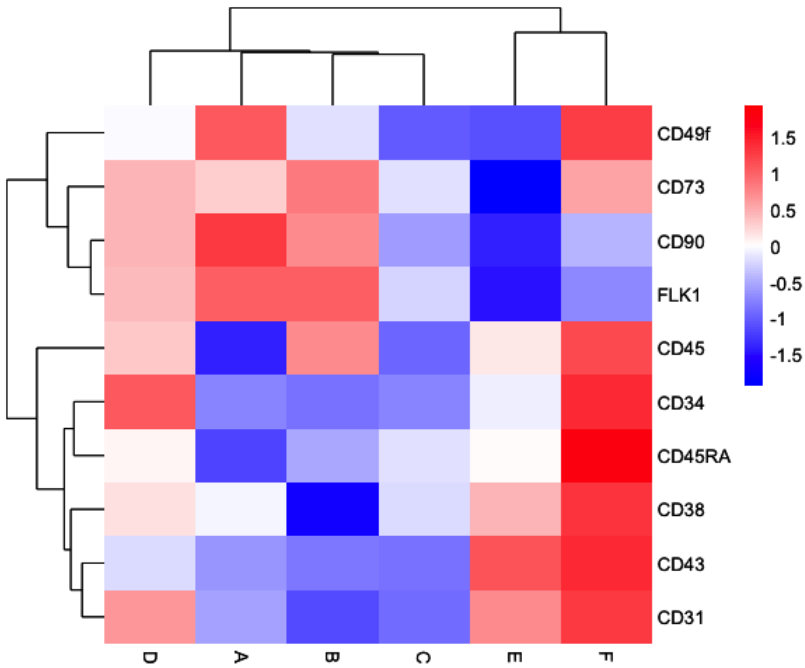
Tree plot, color.by: branch.id, size.by: cell.number



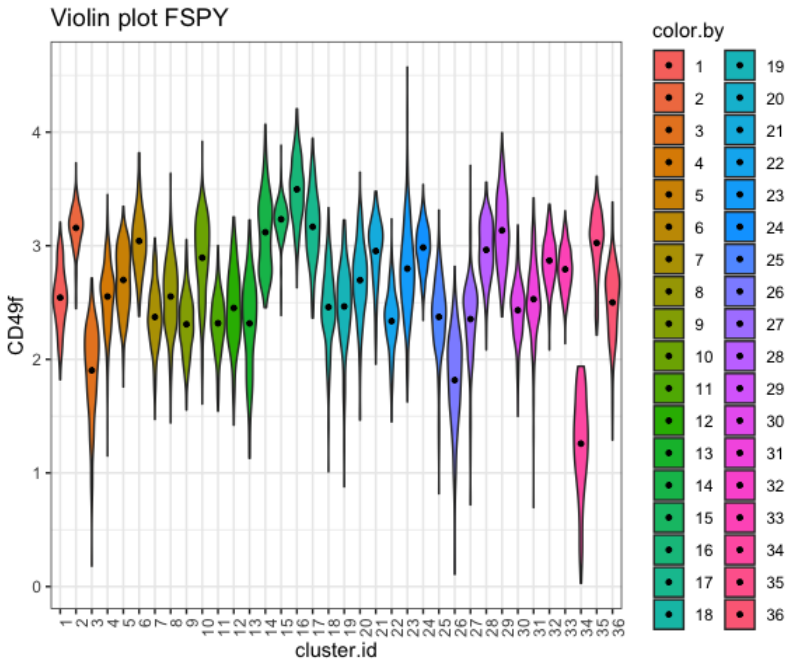

```
# plot heatmap of cluster
plotClusterHeatmap(fspy)
```



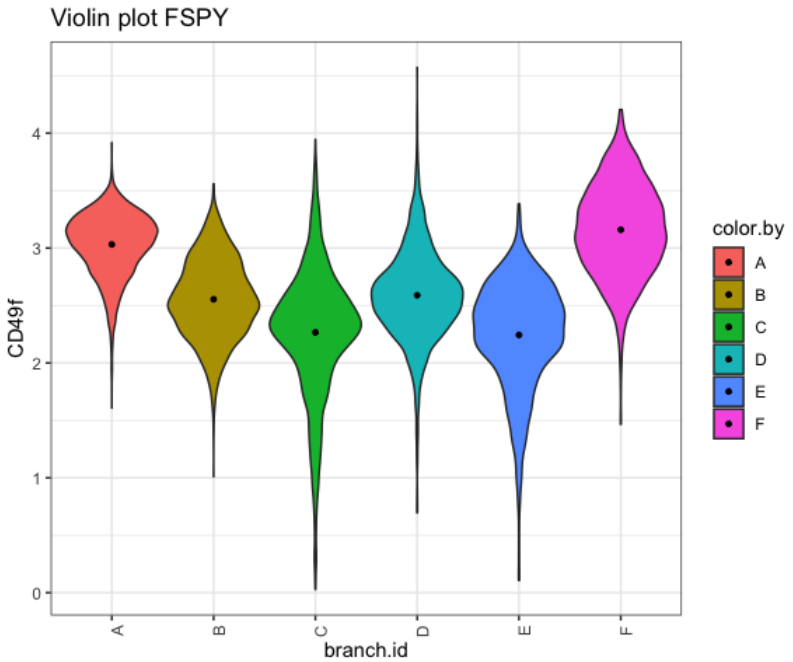

```
plotBranchHeatmap(fspy)
```



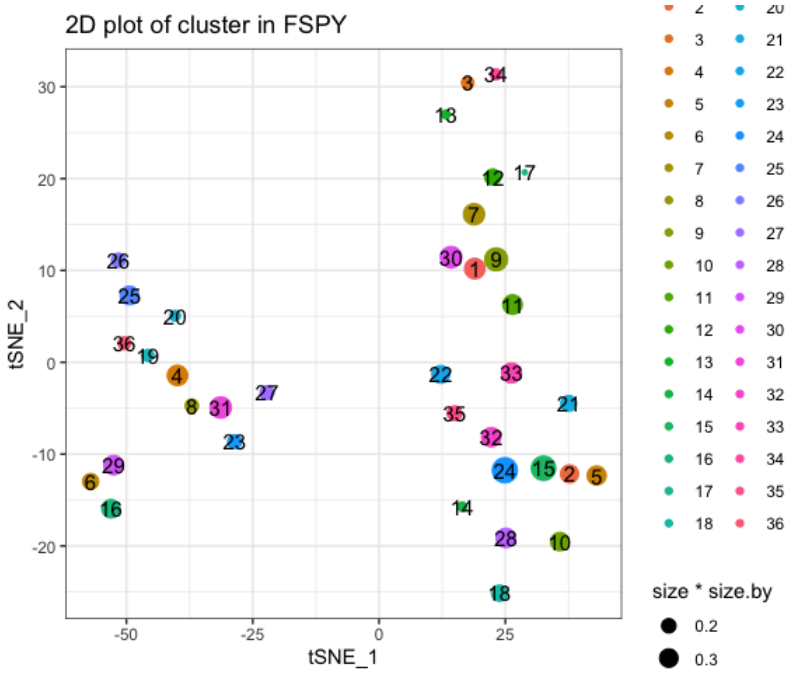

```
# Violin plot
plotViolin(fspy, color.by = "cluster.id", marker = "CD49f", text.angle = 90)
```



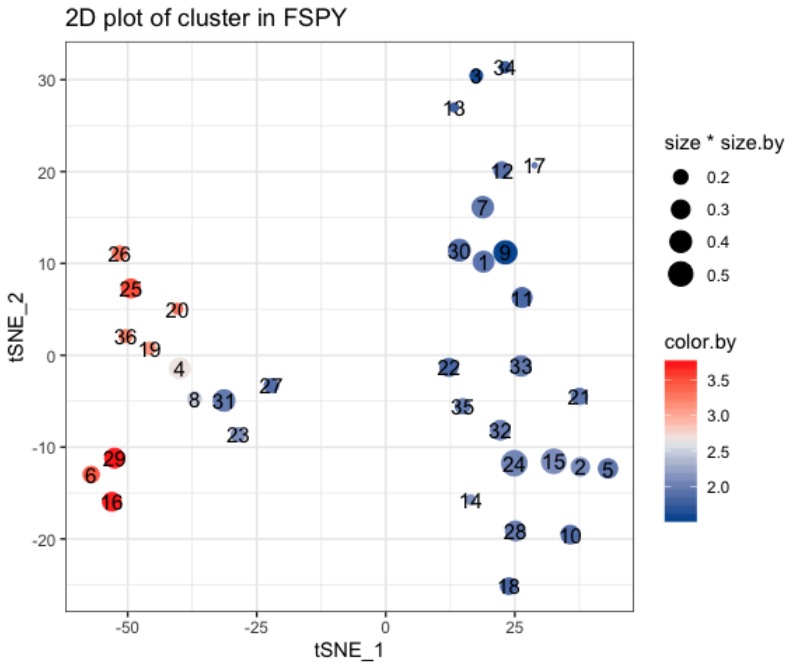

```
plotViolin(fspy, color.by = "branch.id", marker = "CD49f", text.angle = 90)
```




```
# plot cluster
plotCluster(fspy, item.use = c("tSNE_1", "tSNE_2"), size = 10, show.cluser.id = T)
```




```
plotCluster(fspy, item.use = c("tSNE_1", "tSNE_2"), color.by = "CD43",
            size = 10, show.cluser.id = T, category = "numeric") +
 scale_colour_gradientn(colors = c("#00599F", "#EEEEEE", "#FF3222"))
```




```
# run diff list
diff.list <- runDiff(fspy)

##############################################
# Pseudotime
##############################################

fspy <- defRootCells(fspy, root.cells = c(15), verbose = T)
```

```
## 2019-09-09 23:15:49 [INFO] 615 cells will be added to root.cells .
```

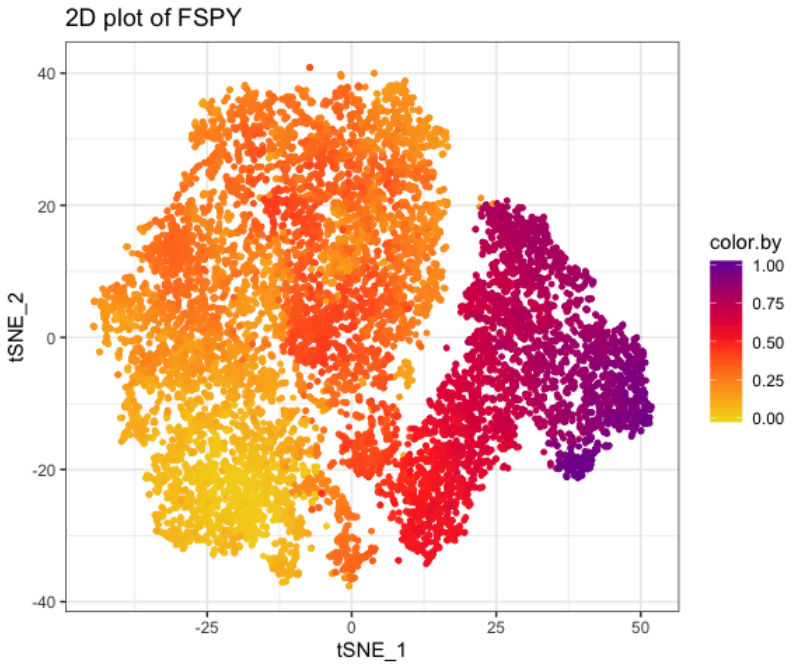
```
fspy <- runPseudotime(fspy, verbose = T, dim.type = "umap", dim.use = 1:2)
```

```
## 2019-09-09 23:15:49 [INFO] Calculating Pseudotime.
```

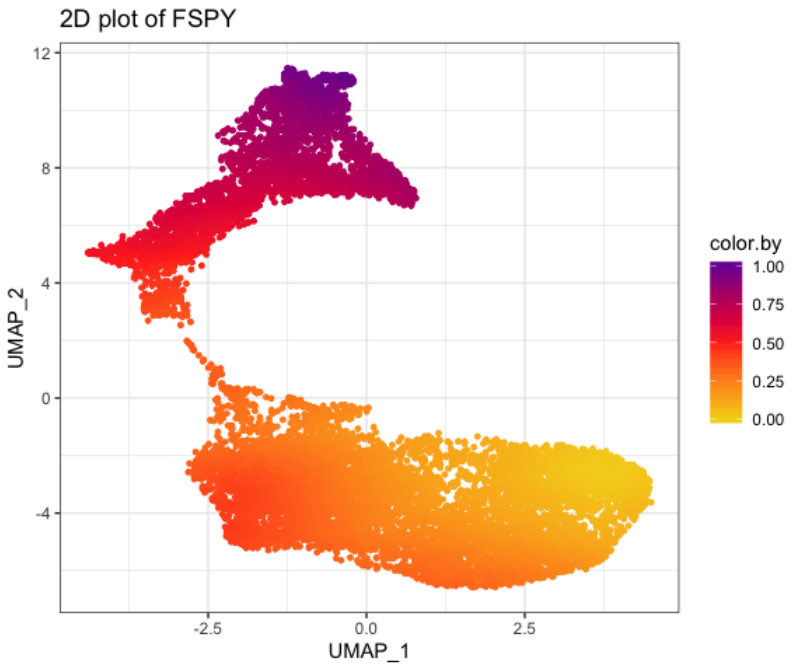
```
## 2019-09-09 23:15:49 [INFO] Pseudotime exists in meta.data, it will be replaced.
```

```
## 2019-09-09 23:16:08 [INFO] Calculating Pseudotime completed.
```

```
# 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"))
```

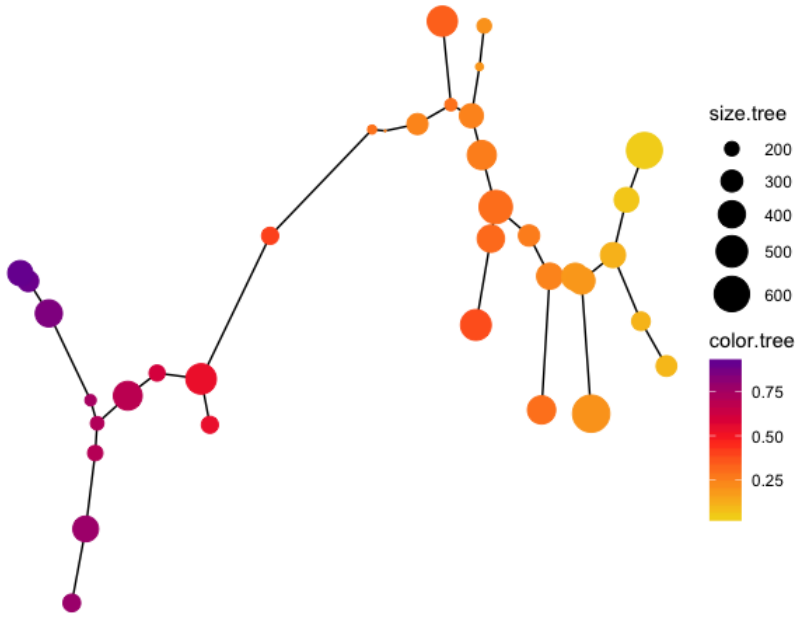



```
# UMAP plot colored by pseudotime
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), category = "numeric",
            size = 1, color.by = "pseudotime") +
 scale_colour_gradientn(colors = c("#F4D31D", "#FF3222","#7A06A0"))
```

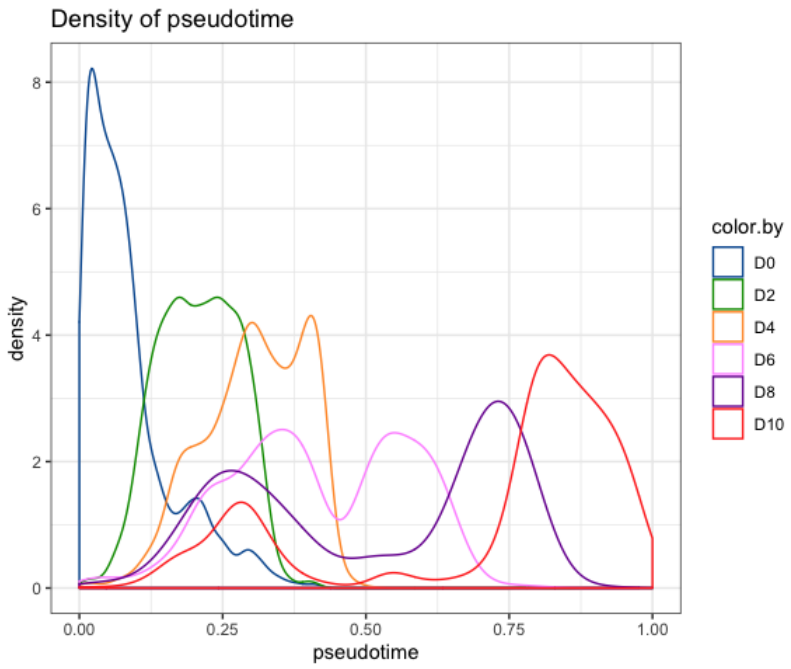


```
# Tree plot
plotTree(fspy, color.by = "pseudotime", cex.size = 1.5) +
 scale_colour_gradientn(colors = c("#F4D31D", "#FF3222","#7A06A0"))
```

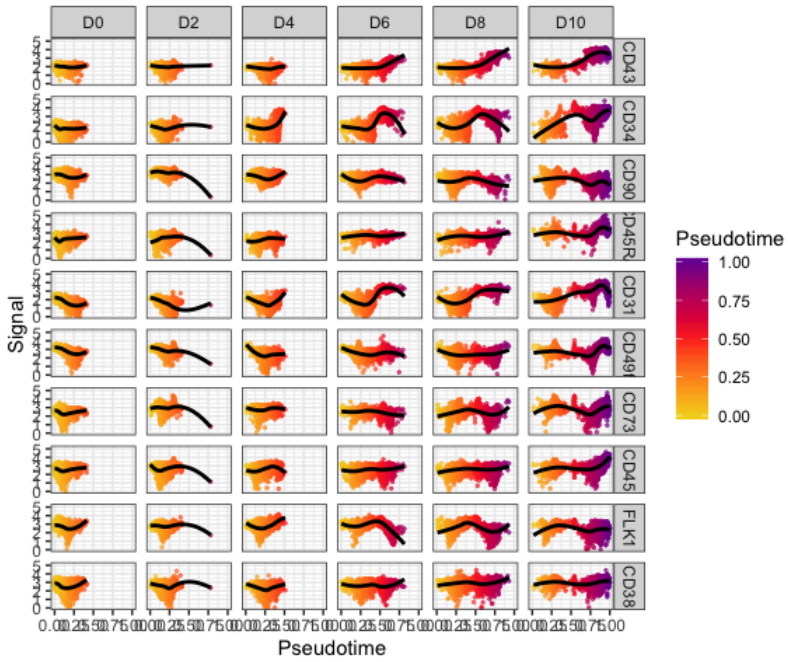
Tree plot, color.by: pseudotime, size.by: cell.number



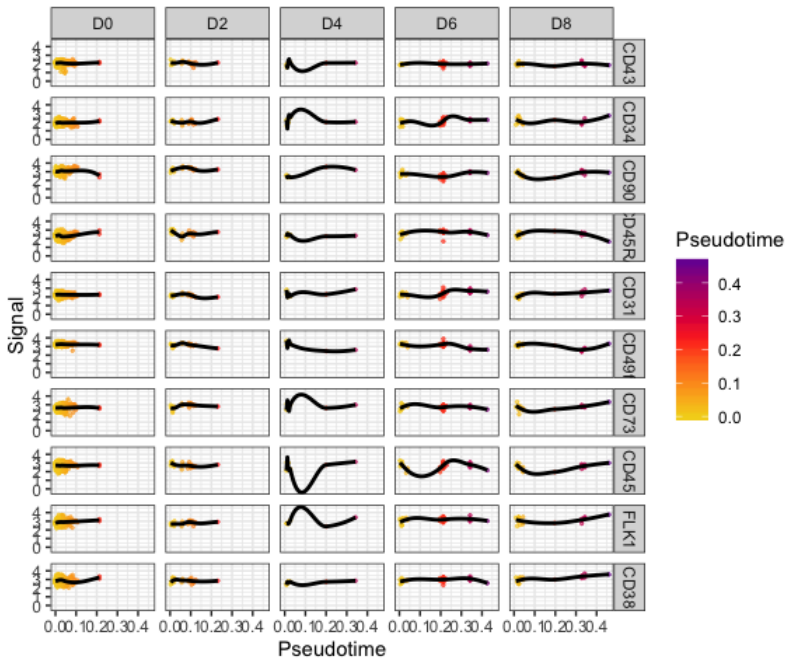

```
# denisty plot by different stage
plotPseudotimeDensity(fspy, adjust = 1) +
 scale_color_manual(values = c("#00599F","#009900","#FF9933",
                               "#FF99FF","#7A06A0","#FF3222"))
```



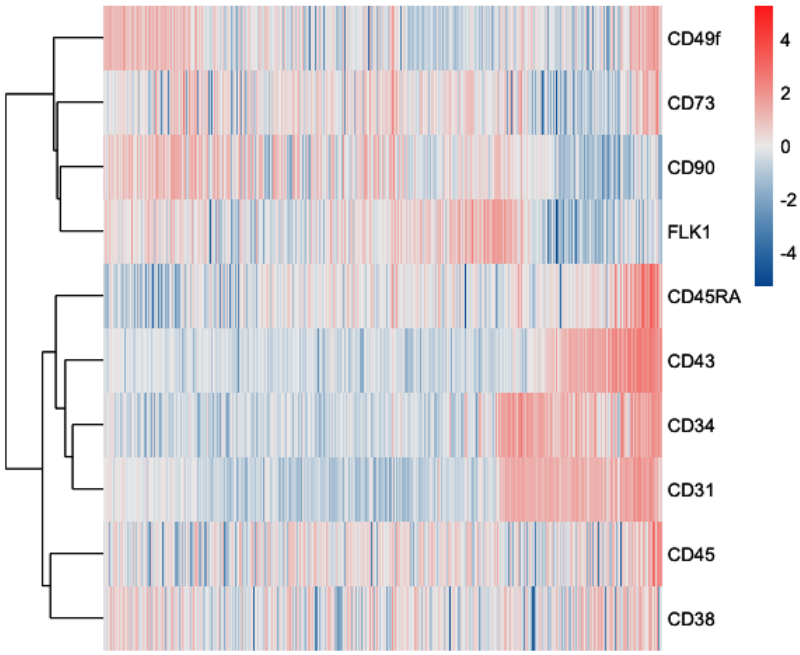

```
# trajectory value
plotPseudotimeTraj(fspy, var.cols = T) +
 scale_colour_gradientn(colors = c("#F4D31D", "#FF3222","#7A06A0"))
```




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




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




```
##########################################
# Subset FSPY
##########################################

cell.inter <- fetchCell(fspy, cluster.id = c(26,25,36,19,4,8,31,20,29,6,16))
cell.inter <- cell.inter[grep("D6|D8|D10", cell.inter)]
sub.fspy <- subsetFSPY(fspy, cells = cell.inter)

set.seed(1)
sub.fspy <- runCluster(sub.fspy, cluster.method = "som", xdim = 4, ydim = 4, verbose = T)
```

```
## 2019-09-09 23:16:21 [INFO] Calculating FlowSOM.
```

```
## 2019-09-09 23:16:21 [INFO] Calculating FlowSOM completed.
```

```
# Do not perform downsampling
set.seed(1)
sub.fspy <- processingCluster(sub.fspy, perplexity = 2, downsampling.size = 1, force.resampl

# run Diffusion map
set.seed(1)
sub.fspy <- runDiffusionMap(sub.fspy, verbose = T)
```

```
## 2019-09-09 23:16:21 [INFO] Calculating Diffusion Map.
```

```
## 2019-09-09 23:16:21 [INFO] Destiny determined an optimal global sigma: 0.811
```

```
## 2019-09-09 23:16:23 [INFO] Calculating Diffusion Map completed
```

```
sub.fspy <- defRootCells(sub.fspy, root.cells = c(13), verbose = T)
```

```
## 2019-09-09 23:16:23 [INFO] 348 cells will be added to root.cells .
```

```
sub.fspy <- runPseudotime(sub.fspy, verbose = T, dim.type = "raw", dim.use = 1:2)
```

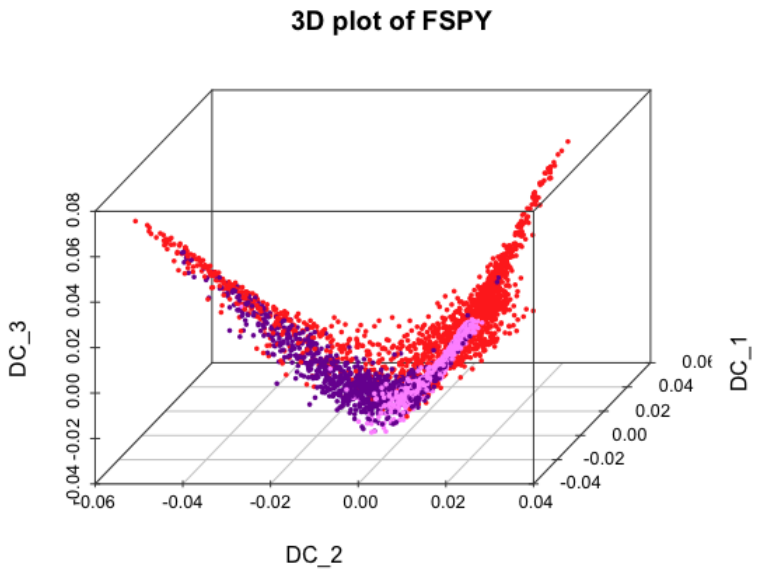
```
## 2019-09-09 23:16:23 [INFO] Calculating Pseudotime.
```

```
## 2019-09-09 23:16:23 [INFO] Pseudotime exists in meta.data, it will be replaced.
```

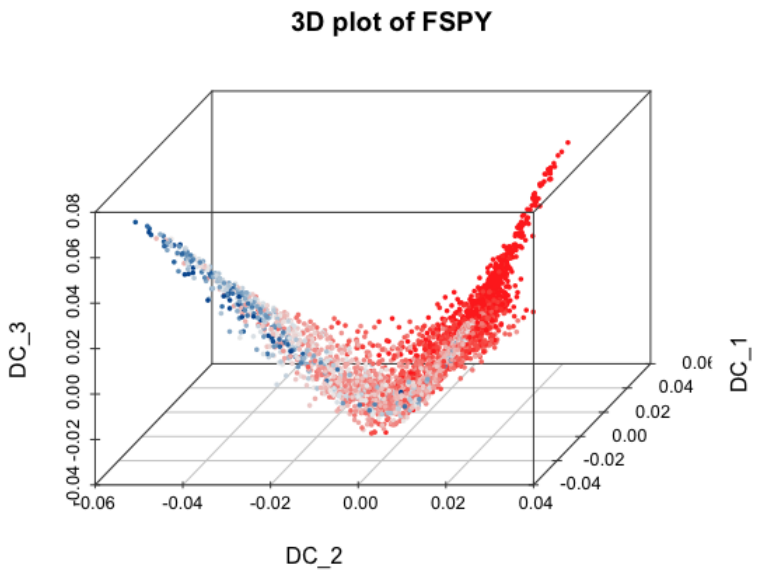
```
## 2019-09-09 23:16:23 [INFO] The log data will be used to calculate trajectory
```

```
## 2019-09-09 23:16:24 [INFO] Calculating Pseudotime completed.
```

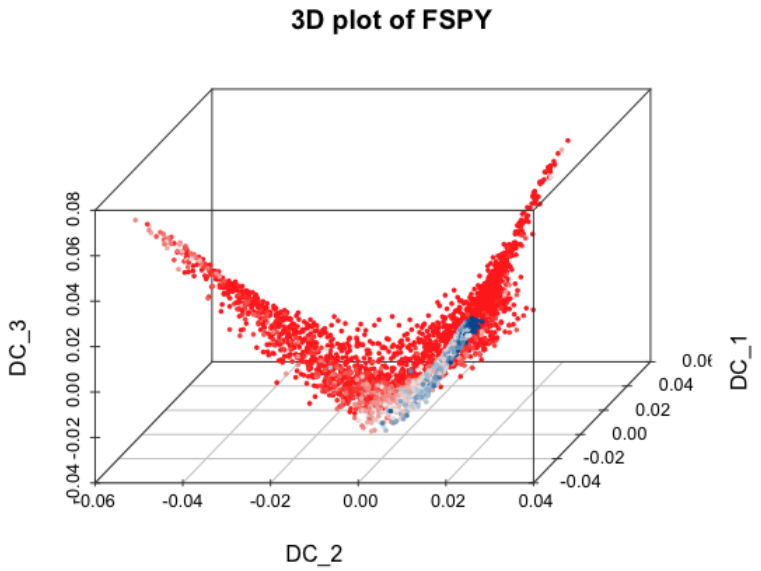
```
# 3D plot for FSPY
plot3D(sub.fspy, item.use = c("DC_2","DC_1","DC_3"), color.by = "stage",
       size = 0.5, angle = 60, color.theme = c("#FF99FF","#7A06A0","#FF3222"))
```



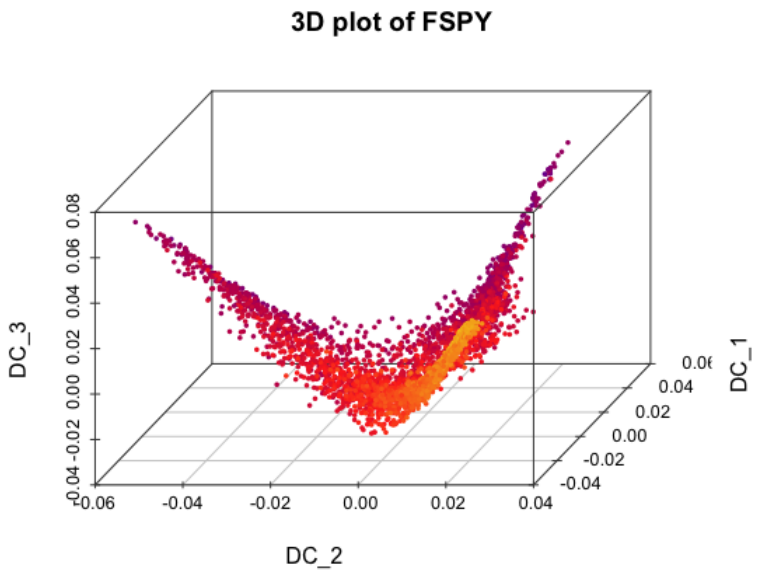

```
plot3D(sub.fspy, item.use = c("DC_2","DC_1","DC_3"),
       size = 0.5, color.by = "CD49f", angle = 60,  category = "numeric",
       color.theme = c("#00599F","#00599F","#EEEEEE","#FF3222","#FF3222"))
```




```
plot3D(sub.fspy, item.use = c("DC_2","DC_1","DC_3"),
       size = 0.5, color.by = "CD43", angle = 60, category = "numeric",
       color.theme = c("#00599F","#00599F","#EEEEEE","#FF3222","#FF3222"))
```




```
plot3D(sub.fspy, item.use = c("DC_2","DC_1","DC_3"),
       size = 0.5, color.by = "pseudotime", angle = 60,  category = "numeric",
       color.theme = c("#F4D31D", "#FF3222","#7A06A0"))
```




## Session information

```
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## 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   flowSpy_1.2.7   igraph_1.2.4.1  pheatmap_1.0.12
## [5] flowCore_1.50.0 ggplot2_3.2.0
##
## loaded via a namespace (and not attached):
##   [1] readxl_1.3.1                backports_1.1.4
##   [3] RcppEigen_0.3.3.5.0         plyr_1.8.4
##   [5] ConsensusClusterPlus_1.48.0 lazyeval_0.2.2
##   [7] sp_1.3-1                    splines_3.6.1
##   [9] BiocParallel_1.18.1         GenomeInfoDb_1.20.0
##  [11] sva_3.32.1                  digest_0.6.20
##  [13] htmltools_0.3.6             gdata_2.18.0
##  [15] magrittr_1.5                memoise_1.1.0
##  [17] cluster_2.1.0               openxlsx_4.1.0.1
##  [19] limma_3.40.6                annotate_1.62.0
##  [21] matrixStats_0.54.0          gmodels_2.18.1
##  [23] xts_0.11-2                  colorspace_1.4-1
##  [25] blob_1.2.0                  rrcov_1.4-7
##  [27] haven_2.1.1                 xfun_0.8
##  [29] dplyr_0.8.3                 crayon_1.3.4
##  [31] RCurl_1.95-4.12             jsonlite_1.6
##  [33] graph_1.62.0                scatterpie_0.1.2
```

```
##  [35] genefilter_1.66.0           zeallot_0.1.0
##  [37] survival_2.44-1.1           zoo_1.8-6
##  [39] glue_1.3.1                  polyclip_1.10-0
##  [41] gtable_0.3.0                zlibbioc_1.30.0
##  [43] XVector_0.24.0              DelayedArray_0.10.0
##  [45] car_3.0-3                   BiocGenerics_0.30.0
##  [47] DEoptimR_1.0-8              abind_1.4-5
##  [49] VIM_4.8.0                   scales_1.0.0
##  [51] mvtnorm_1.0-11              DBI_1.0.0
##  [53] ggthemes_4.2.0              Rcpp_1.0.2
##  [55] xtable_1.8-4                laeken_0.5.0
##  [57] reticulate_1.13             foreign_0.8-72
##  [59] bit_1.1-14                  proxy_0.4-23
##  [61] mclust_5.4.5                FlowSOM_1.16.0
##  [63] stats4_3.6.1                tsne_0.1-3
##  [65] umap_0.2.2.0                vcd_1.4-4
##  [67] RColorBrewer_1.1-2          pkgconfig_2.0.2
##  [69] XML_3.98-1.20               farver_1.1.0
##  [71] nnet_7.3-12                 labeling_0.3
##  [73] reshape2_1.4.3              tidyselect_0.2.5
##  [75] rlang_0.4.0                 AnnotationDbi_1.46.0
##  [77] munsell_0.5.0               cellranger_1.1.0
##  [79] tools_3.6.1                 RSQLite_2.1.2
##  [81] ranger_0.11.2               evaluate_0.14
##  [83] yaml_2.2.0                  knitr_1.24
##  [85] bit64_0.9-7                 zip_2.0.3
##  [87] robustbase_0.93-5           purrr_0.3.2
##  [89] RANN_2.6.1                  nlme_3.1-141
##  [91] compiler_3.6.1              curl_4.0
##  [93] e1071_1.7-2                 smoother_1.1
##  [95] tibble_2.1.3                tweenr_1.0.1
##  [97] pcaPP_1.9-73                stringi_1.4.3
##  [99] RSpectra_0.15-0             forcats_0.4.0
## [101] lattice_0.20-38             Matrix_1.2-17
## [103] vctrs_0.2.0                 pillar_1.4.2
## [105] RUnit_0.4.32                lmtest_0.9-37
## [107] BiocNeighbors_1.2.0         data.table_1.12.2
## [109] bitops_1.0-6                corpcor_1.6.9
## [111] GenomicRanges_1.36.0        R6_2.4.0
## [113] rio_0.5.16                  IRanges_2.18.1
## [115] flowUtils_1.48.0            boot_1.3-23
## [117] MASS_7.3-51.4               gtools_3.8.1
## [119] assertthat_0.2.1            destiny_2.14.0
## [121] SummarizedExperiment_1.14.1 withr_2.1.2
## [123] S4Vectors_0.22.0            GenomeInfoDbData_1.2.1
## [125] mgcv_1.8-28                 parallel_3.6.1
## [127] hms_0.5.0                   grid_3.6.1
## [129] prettydoc_0.3.0             tidyr_0.8.3
## [131] class_7.3-15                rmarkdown_1.14
## [133] rvcheck_0.1.3               carData_3.0-2
## [135] Rtsne_0.15                  TTR_0.23-4
## [137] ggforce_0.2.2               scatterplot3d_0.3-41
## [139] Biobase_2.44.0
```

## Reference

[1] Wang C, Tang X, Sun X, Miao Z, Lv Y, Yang Y, Zhang H, Zhang P, Liu Y, Du L, et al: TGFbeta inhibition enhances the generation of hematopoietic progenitors from human ES cell-derived hemogenic endothelial cells using a stepwise strategy. Cell Res 2012, 22:194-207.