RISC: implementing RPCI algorithm for integration of single-cell RNA-seq datasets

Yang Liu, Tao Wang, Deyou Zheng

Albert Einstein College of Medicine, Bronx, NY, United States

2020

Introduction

Integrated analysis of single cell RNA-sequencing (scRNA-seq) data from multiple batches or studies is often necessary in order to learn functional changes in cellular states upon experimental perturbations or cell type relationships in a developmental lineage. Here we introduce a new algorithm (RPCI) that uses the gene-eigenvectors from a reference dataset to establish a global frame for projecting all datasets, with a clear advantage in preserving genuine gene expression differences in matching cell types between samples, such as those present in cells at distinct developmental stages or in perturbated vs control studies. This R package "RISC" (Robust Integration of Sinlgle Cell RNA-seq data implements the RPCI algorithm, with additional functions for scRNA-seq analysis, such as clustering cells, identifying cluster marker genes, detecting differentially expressed genes between experimental conditions, and importantly outputting integrated gene expression values for downstream data analysis.

This vignette shows the key steps in analyzing example scRNA-seq datasets from the basal or squamous carcinoma patients before and after anti-PD-1 therapy (GSE123813). The data is consisted of two parts, one was derived from T cells of basal cell carcinoma (BCC) patients by single-cell RNA and T cell receptor (TCR) sequencing, while the other from site-matched tumor cells. The marker CD45+ and CD3+ were used to label tumor-infiltrating T cells, CD45+ and CD3- for tumor-infiltrating lymphocytes, and tumor cells identified by CD45- and CD3-. In the original study, 16 scRNA-seq datasets were collected from eight patients. For each patient, the study contained scRNA-seq data before and after anti-PD-1 therapy. However, the datasets from some patients only had few cells or low number of expressed genes; therefore, we filtered them out and chose eight datasets from four patients for this test, including pre- and post- anti-PD-1 therapy.

The public data can be obtained from the GEO (GSE123813), this vignette needs three files (you could download and unzip the files):
(a) the matrix file ("bcc_scRNA_counts.txt")
https://ftp.ncbi.nlm.nih.gov/geo/series/GSE123nnn/GSE123813/suppl/GSE123813%5Fbcc%5FscRNA%5Fcounts%2Etxt%
(b) meta_cell file ("bcc_all_metadata.txt")
https://ftp.ncbi.nlm.nih.gov/geo/series/GSE123nnn/GSE123813/suppl/GSE123813%5Fbcc%5Fall%5Fmetadata%2Etxt%2
(c) annotation file ("bcc_annotation.tsv"), enclosed in our manuscript revision and the RISC release

The three files will be required in this test and need to be put in the path below:

```
## Create a directory named "GSE123813" and
## a sub-directory named "Raw_Data" in the directory "GSE123813".
# Point the path to "GSE123813"
print("PATH = '/Your path to directory/GSE123813'")
```

```
## [1] "PATH = '/Your path to directory/GSE123813'"
```

```
## The vignette needs the packages below.
# install.packages(c("Matrix", "ggplot2", "RColorBrewer"))
```

```
## The packages are required to install RISC
# install.packages(c(
#   "Matrix", "matrixStats", "irlba", "doParallel", "foreach",
#   "data.table", "Rtsne", "umap", "MASS", "pbmcapply", "Rcpp",
#   "RcppEigen", "densityClust", "FNN", "igraph", "RColorBrewer",
#   "ggplot2", "gridExtra", "pheatmap"
# ))

## Install RISC package locally,
## RISC is developed in R (v3.6.3)
# install.packages("/your path to/RISC_1.0.tar.gz", repos = NULL, type = "source")
```

Notice, RISC package is developed in R (v3.6.3), we test this vignette in the same R version.

This vignette contains two parts: (1) the codes in R environment, (2) the codes in python environment

In R environment

Prepare individual RISC objects

We first read the raw counts and filter out the cells we do not use in this test. We also check the number of expressed genes and make sure each gene expressed at least in one cell. Then, we create RISC objects, each object contains gene expression values in gene-cell matrix, the information of cells and of expressed genes. We show an example of dat1 below

```
library(RISC)
library(Matrix)
library(ggplot2)
library(RColorBrewer)


## Set your parth to directory GSE123813
# PATH = "/Your path to directory/GSE123813"

## for example, my path to the directory GSE123813
PATH = "/home/yang/Documents/Vignette/GSE123813"

### Prepare the individual datasets
## All Patients
# The gene-cell matrix file
mat0 = read.table(file = paste0(PATH, "/Raw_Data/bcc_scRNA_counts.txt"), sep = "\t")
# The cell information
coldata0 = read.table(
  file = paste0(PATH, "/Raw_Data/bcc_all_metadata.txt"),
  sep = "\t", header = T, stringsAsFactors = F
)
coldata0 = coldata0[c("cell.id", "patient", "treatment", "sort")]

# su001 pre-treatment
cell1 = coldata0$cell.id[coldata0$treatment == "pre" & coldata0$patient == "su001"]
coldata1 = coldata0[coldata0$cell.id %in% cell1, c("cell.id", "patient", "treatment", "sort")]
rownames(coldata1) = coldata1$cell.id
mat1 = mat0[,coldata1$cell.id]
keep = rowSums(mat1 > 0) > 0
mat1 = mat1[keep,]
rowdata0 = data.frame(Symbol = rownames(mat1), row.names = rownames(mat1))
coldata1 = coldata1[,-1]
## Create RISC object
# make sure the row-names of mat1 equal to row-names of rowdata0,
# and the col-names of mat1 equal to row-names of coldata1
dat1 = readscdata(mat1, coldata1, rowdata0, is.filter = F)
print(dat1)
```

```
## SingleCell-Dataset
##  RISC
##  assay: count
##  colData: (1149) scBarcode scUMI ngene mito patient treatment sort
##  rowData: (14219) Symbol RNA nCell
##  DimReduction
##  Cell-Clustering
```

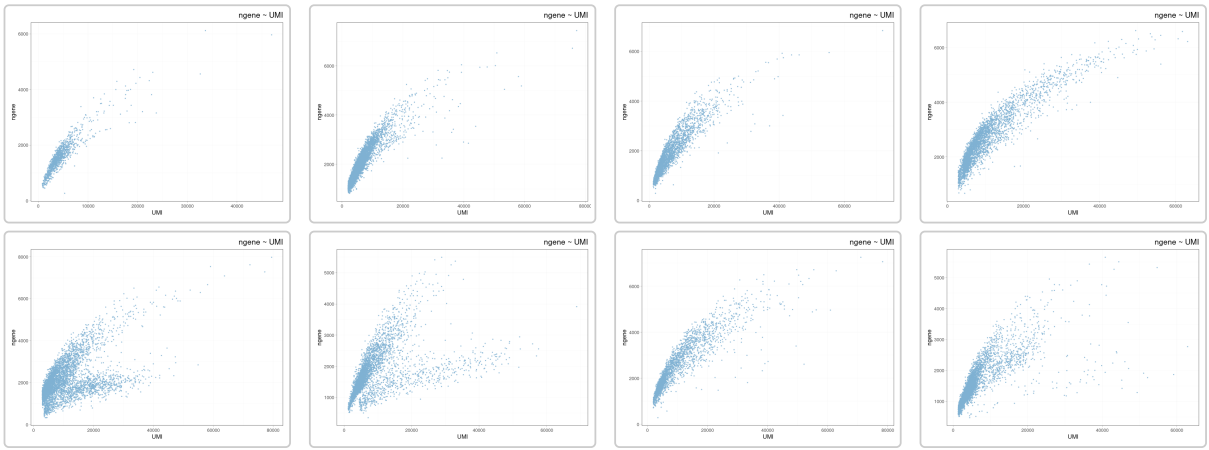
```r
# su001 post-treatment
cell1 = coldata0$cell.id[coldata0$treatment == "post" & coldata0$patient == "su001"]
coldata1 = coldata0[coldata0$cell.id %in% cell1, c("cell.id", "patient", "treatment", "sort")]
rownames(coldata1) = coldata1$cell.id
mat1 = mat0[,coldata1$cell.id]
keep = rowSums(mat1 > 0) > 0
mat1 = mat1[keep,]
rowdata0 = data.frame(Symbol = rownames(mat1), row.names = rownames(mat1))
coldata1 = coldata1[,-1]
dat2 = readscdata(mat1, coldata1, rowdata0, is.filter = F)

# su005 pre-treatment
cell1 = coldata0$cell.id[coldata0$treatment == "pre" & coldata0$patient == "su005"]
coldata1 = coldata0[coldata0$cell.id %in% cell1, c("cell.id", "patient", "treatment", "sort")]
rownames(coldata1) = coldata1$cell.id
mat1 = mat0[,coldata1$cell.id]
keep = rowSums(mat1 > 0) > 0
mat1 = mat1[keep,]
rowdata0 = data.frame(Symbol = rownames(mat1), row.names = rownames(mat1))
coldata1 = coldata1[,-1]
dat3 = readscdata(mat1, coldata1, rowdata0, is.filter = F)

# su005 post-treatment
cell1 = coldata0$cell.id[coldata0$treatment == "post" & coldata0$patient == "su005"]
coldata1 = coldata0[coldata0$cell.id %in% cell1, c("cell.id", "patient", "treatment", "sort")]
rownames(coldata1) = coldata1$cell.id
mat1 = mat0[,coldata1$cell.id]
keep = rowSums(mat1 > 0) > 0
mat1 = mat1[keep,]
rowdata0 = data.frame(Symbol = rownames(mat1), row.names = rownames(mat1))
coldata1 = coldata1[,-1]
dat4 = readscdata(mat1, coldata1, rowdata0, is.filter = F)

# su006 pre-treatment
cell1 = coldata0$cell.id[coldata0$treatment == "pre" & coldata0$patient == "su006"]
coldata1 = coldata0[coldata0$cell.id %in% cell1, c("cell.id", "patient", "treatment", "sort")]
rownames(coldata1) = coldata1$cell.id
mat1 = mat0[,coldata1$cell.id]
keep = rowSums(mat1 > 0) > 0
mat1 = mat1[keep,]
rowdata0 = data.frame(Symbol = rownames(mat1), row.names = rownames(mat1))
coldata1 = coldata1[,-1]
dat5 = readscdata(mat1, coldata1, rowdata0, is.filter = F)

# su006 post-treatment
cell1 = coldata0$cell.id[coldata0$treatment == "post" & coldata0$patient == "su006"]
coldata1 = coldata0[coldata0$cell.id %in% cell1, c("cell.id", "patient", "treatment", "sort")]
rownames(coldata1) = coldata1$cell.id
mat1 = mat0[,coldata1$cell.id]
keep = rowSums(mat1 > 0) > 0
mat1 = mat1[keep,]
rowdata0 = data.frame(Symbol = rownames(mat1), row.names = rownames(mat1))
coldata1 = coldata1[,-1]
dat6 = readscdata(mat1, coldata1, rowdata0, is.filter = F)

# su008 pre-treatment
cell1 = coldata0$cell.id[coldata0$treatment == "pre" & coldata0$patient == "su008"]
coldata1 = coldata0[coldata0$cell.id %in% cell1, c("cell.id", "patient", "treatment", "sort")]
rownames(coldata1) = coldata1$cell.id
mat1 = mat0[,coldata1$cell.id]
```

```
keep = rowSums(mat1 > 0) > 0
mat1 = mat1[keep,]
rowdata0 = data.frame(Symbol = rownames(mat1), row.names = rownames(mat1))
coldata1 = coldata1[,-1]
dat7 = readscdata(mat1, coldata1, rowdata0, is.filter = F)

# su008 post-treatment
cell1 = coldata0$cell.id[coldata0$treatment == "post" & coldata0$patient == "su008"]
coldata1 = coldata0[coldata0$cell.id %in% cell1, c("cell.id", "patient", "treatment", "sort")]
rownames(coldata1) = coldata1$cell.id
mat1 = mat0[,coldata1$cell.id]
keep = rowSums(mat1 > 0) > 0
mat1 = mat1[keep,]
rowdata0 = data.frame(Symbol = rownames(mat1), row.names = rownames(mat1))
coldata1 = coldata1[,-1]
dat8 = readscdata(mat1, coldata1, rowdata0, is.filter = F)
```

Data quality

We check the data quality before pre-processing

```
# Check Quality
FilterPlot(dat1)
FilterPlot(dat2)
FilterPlot(dat3)
FilterPlot(dat4)
FilterPlot(dat5)
FilterPlot(dat6)
FilterPlot(dat7)
FilterPlot(dat8)
```

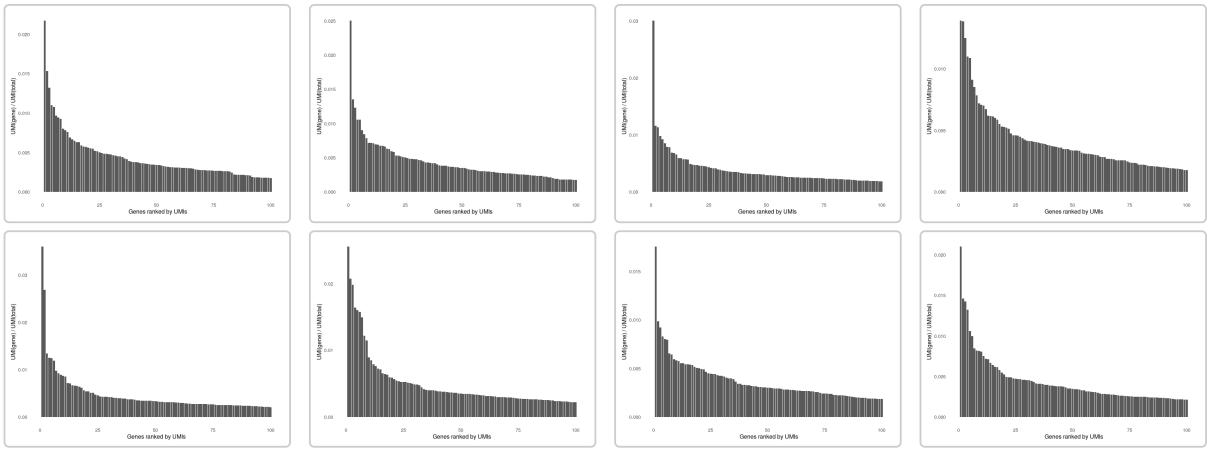


Proportion of UMIs per gene

We further examine the UMI/gene distributions in the datasets and check UMI distribution of individual genes in each dataset. In most case, the proportion < 0.1 (i.e. the UMIs of any gene is less than 10% of the total UMIs). If any gene with extremely high UMIs, we will consider to remove this gene from the gene-cell matrix. It is most likely to be artificial and will affect the biological complexity of datasets.

```
Per0 <- function(obj0){
  count0 = obj0@assay$count
  umi.gene = Matrix::rowSums(count0)
  num0 = 100 # top 100 genes
  umi.gene.per = sort(umi.gene / sum(umi.gene), decreasing = TRUE)[1:num0]
```

```
  m0 = data.frame(Num = 1:num0, Per = as.numeric(umi.gene.per))
  g0 = ggplot(m0, aes(Num, Per)) +
    geom_bar(stat = 'identity') +
    labs(x = "Genes ranked by UMIs", y = "UMI(gene) / UMI(total)") +
    theme_classic(base_size = 12, base_line_size = 0)
  print(g0)
}

Per0(dat1)
Per0(dat2)
Per0(dat3)
Per0(dat4)
Per0(dat5)
Per0(dat6)
Per0(dat7)
Per0(dat8)
```



Processing RISC objects

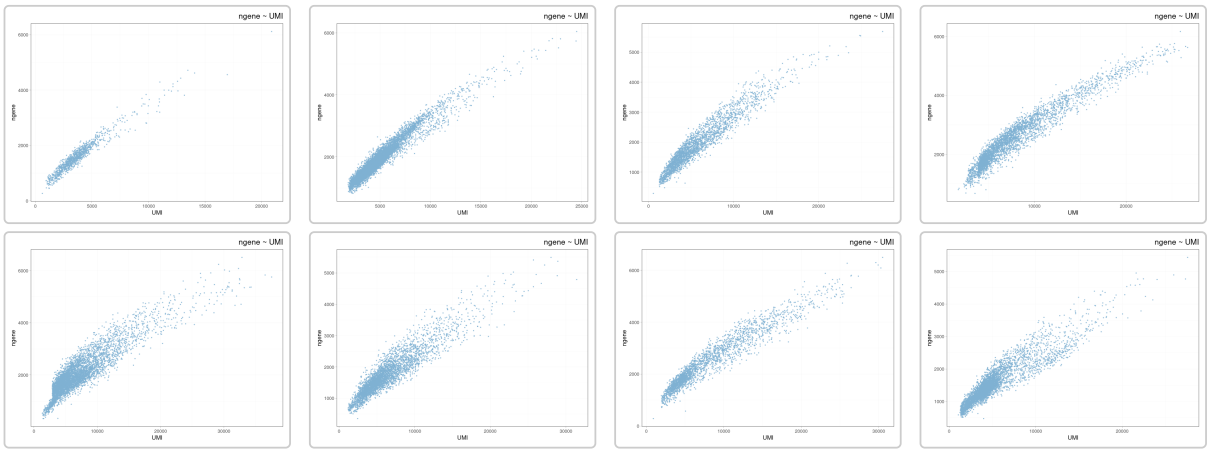
After creating RISC objects, we next process the RISC data, here we show the standard processes:
(1) Filter the cells. we remove cells with extremely low or high UMIs and discard cells with extremely low number of expressed genes.
(2) Normalize gene expression. Here we normalize the raw counts to remove the effect of RNA sequencing depth, using cosine normalization method here, and we also have other methods to normalize the counts, see help file.
(3) Find highly variable genes. We identify highly variable genes by Quasi-Poisson model, and utilize them to decompose gene-cell matrix and to generate gene-eigenvectors for data integration.

```
process0 <- function(obj0){
  # Filter cells and genes
  obj0 = scFilter(obj0, min.UMI = 1000, max.UMI = 40000, min.gene = 200, min.cell = 3)
  # Normalize the raw counts
  obj0 = scNormalize(obj0)
  # Find highly variable genes
  obj0 = scDisperse(obj0)
  # print(length(obj0@vargene))
  return(obj0)
}

dat1 = process0(dat1)
dat2 = process0(dat2)
dat3 = process0(dat3)
dat4 = process0(dat4)
dat5 = process0(dat5)
```

```
dat6 = process0(dat6)
dat7 = process0(dat7)
dat8 = process0(dat8)
```

Check data quality after processing

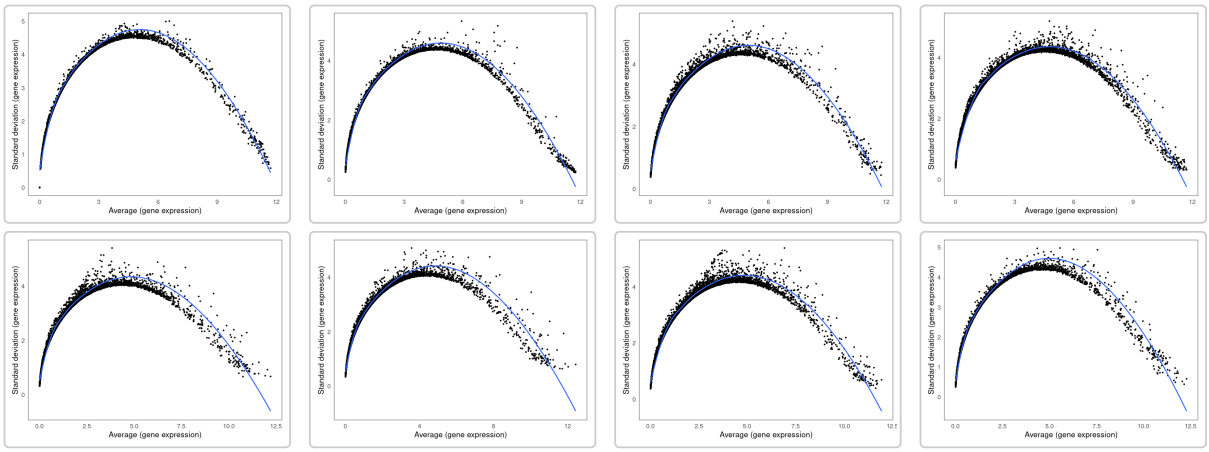
```
par(mfrow = c(2, 4))
# Check Quality
FilterPlot(dat1)
FilterPlot(dat2)
FilterPlot(dat3)
FilterPlot(dat4)
FilterPlot(dat5)
FilterPlot(dat6)
FilterPlot(dat7)
FilterPlot(dat8)
```



Check gene expression variance

When we search the highly variable genes, the meta-data of gene expression variance data is already calculated and kept in RISC object

```
Disper0 <- function(obj0){
  # Read the gene expression variance meta-data
  m0 = as.data.frame(obj0@metadata$dispersion)
  # Draw plots
  g0 = ggplot(m0, aes(Mean, SD)) +
    geom_point(size = 1) +
    geom_smooth(method = "loess", span = 0.85, formula = "y ~ x") +
    labs(x = "Average (gene expression)", y = "Standard deviation (gene expression)") +
    theme_bw(base_size = 16, base_line_size = 0)
  print(g0)
}

# Check Quality
Disper0(dat1)
Disper0(dat2)
Disper0(dat3)
Disper0(dat4)
Disper0(dat5)
Disper0(dat6)
Disper0(dat7)
Disper0(dat8)
```



RPCI integration

The core step. RPCI introduces an effective formula to calibrate cell similarity by a global reference gene-eigenvectors, and directly projects all cells into a reference RPCI space. We also have a function "InPlot" to help users to choose a good reference dataset and select the sufficient number of eigenvectors for data integration.

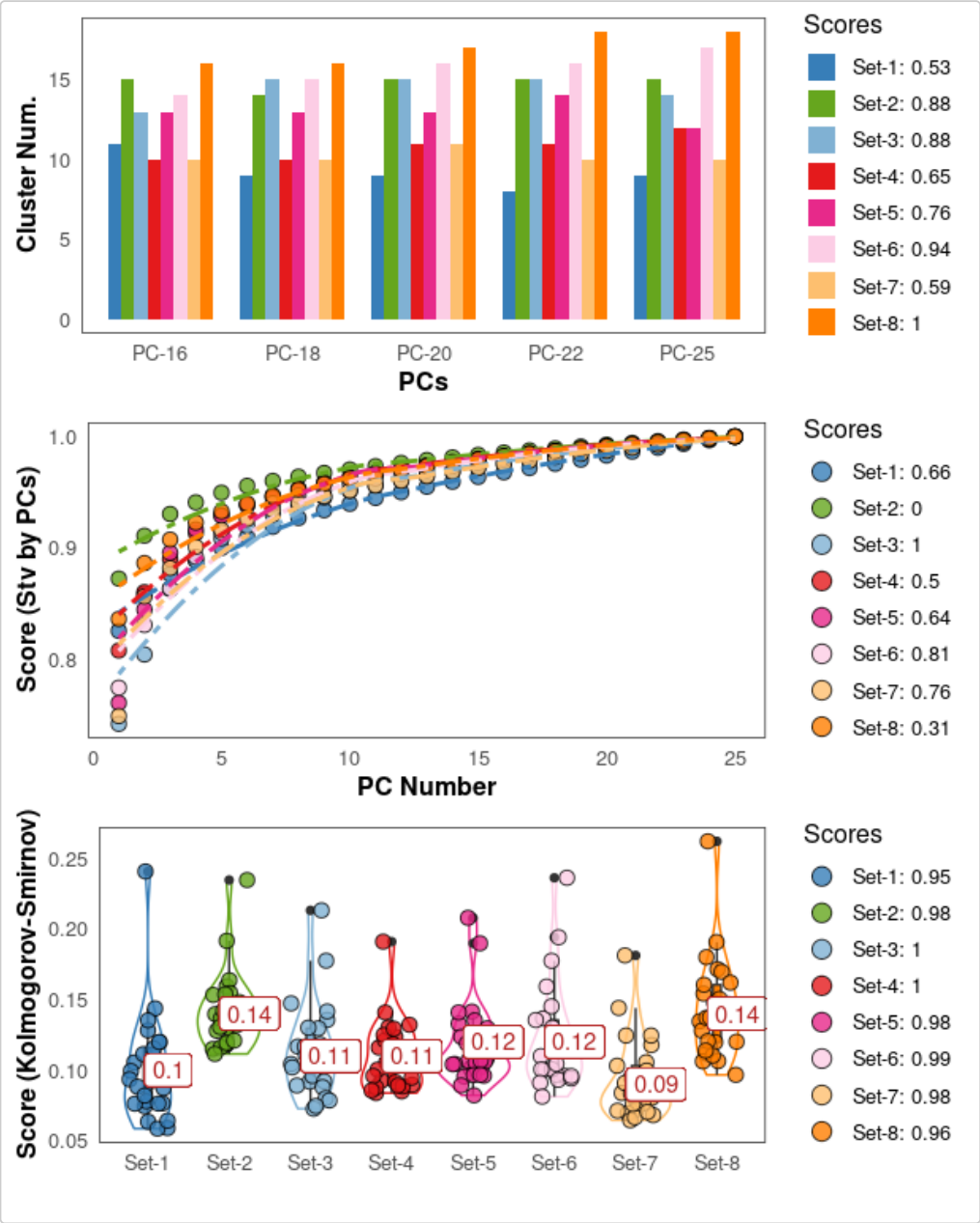
Select the reference

We utilize the "InPlot" function to find the optimal reference. The set in the first place will be the reference during data integration.

```
# The datasets from pre- and post-treatment are quite different and do not have many
# shared variable genes. Here, we use all the genes expressed in every dataset to
# to generate gene-eigenvectors.
var0 = Reduce(
  intersect, list(
    dat1@rowdata$Symbol, dat2@rowdata$Symbol, dat3@rowdata$Symbol, dat4@rowdata$Symbol,
    dat5@rowdata$Symbol, dat6@rowdata$Symbol, dat7@rowdata$Symbol, dat8@rowdata$Symbol
  )
)

# Choose a good reference dataset
data0 = list(dat1, dat2, dat3, dat4, dat5, dat6, dat7, dat8)

# Select the reference set. The we input highly variable genes (var0) into "var.gene"
# slot. The "Std.cut" slot represents the percentage of gene expression variance we
# want to use in the integration. In most cases, "Std.cut" can be 0.85~0.9 for small
# size datasets (like total cells < 10,000) and be 0.90~0.98 for large size
# datasets (total cells > 10,000). The more complicated datasets (with more cell
# populaitons and more total cells) need higher "Std.cut". The "minPC" and "nPC"
# slots are for the minimum and maximum eigenvectors.
# The details of all the parameters in help file.
InPlot(data0, var.gene = var0, Std.cut = 0.99, ncore = 4, minPC = 16, nPC = 25)
```

```
## [1] "The Number of PCs for 99% Gene Expression Variance:"
## Set-1: 23
## Set-2: 19
## Set-3: 21
## Set-4: 20
## Set-5: 20
## Set-6: 21
```

```
## Set-7: 22
## Set-8: 20
```



The outputs of "lnPlot" function show several information: (1) how many eigenvectors in each dataset can explain 95% gene expression variance, indicating the number of eigenvectors we want to use in data integration, (2) which datasets contain most cell populations, cell clustering based on "louvain" method, (3) in which dataset the gene expression variance would be explained by more eigenvectors, (4) whether the eigenvectors in each dataset are distributed normally or not. Each plot contain the corresponding score, we choose the reference based on the scores, weighted by "Cluster Num." score > "Stv by PCs" score > "Kolmogorov-Smirnov" score, but if "Kolmogorov-Smirnov" value of one set (the red values in the violin plot) are extremely high, we do not use the set as the reference, since the eigenvector distribution in this set is abnormal.

Integration

We change the order of datasets and make sure the reference in the first position.

```
# Select the dat6 as the reference (the first position of multiple datasets as the reference)
data0 = list(dat6, dat1, dat2, dat3, dat4, dat5, dat7, dat8)

# Data Integration, "eigens" represent the number of eigenvectors you want to use, "var.gene"
```

```
# indicating the highly variable genes which will be used to decompose the datasets, "align"
# for the algorithm to correct gene expression values, "ncore" suggesting the number of
# multiple cores in parallel calculation of integration, and "do.fast" for calculation speed.
# The details of all parameters in help file.
data0 = scMultiIntegrate(
  objects = data0, eigens = 18, add.Id = NULL, var.gene = var0,
  method = "RPCI", align = 'OLS', npc = 50, adjust = TRUE,
  ncore = 4, do.fast = "AUTO"
)
```

Annotation and cell clustering

We can directly utilized the integrated cell-eigenvectors to cluster the cells. Here we use the cell-type annotation to label the cells for convenience.

UMAP

The calculation of UMAP is based on the integrated cell-eigenvectors.

```
# Calculate UMAP.
data0 = scUMAP(data0, npc = 18, use = "PLS")
```

The integrated cell-eigenvectors.

```
# The matrices from dimension-reduction calculation,
# including the integrated cell-eigenvectors
summary(data0@DimReduction)
```

```
##           Length  Class  Mode
## cell.pls  1735350 -none- numeric
## cell.umap   69414 -none- numeric
```

```
pls0 = as.matrix(data0@DimReduction$cell.pls)
dim(pls0)
```

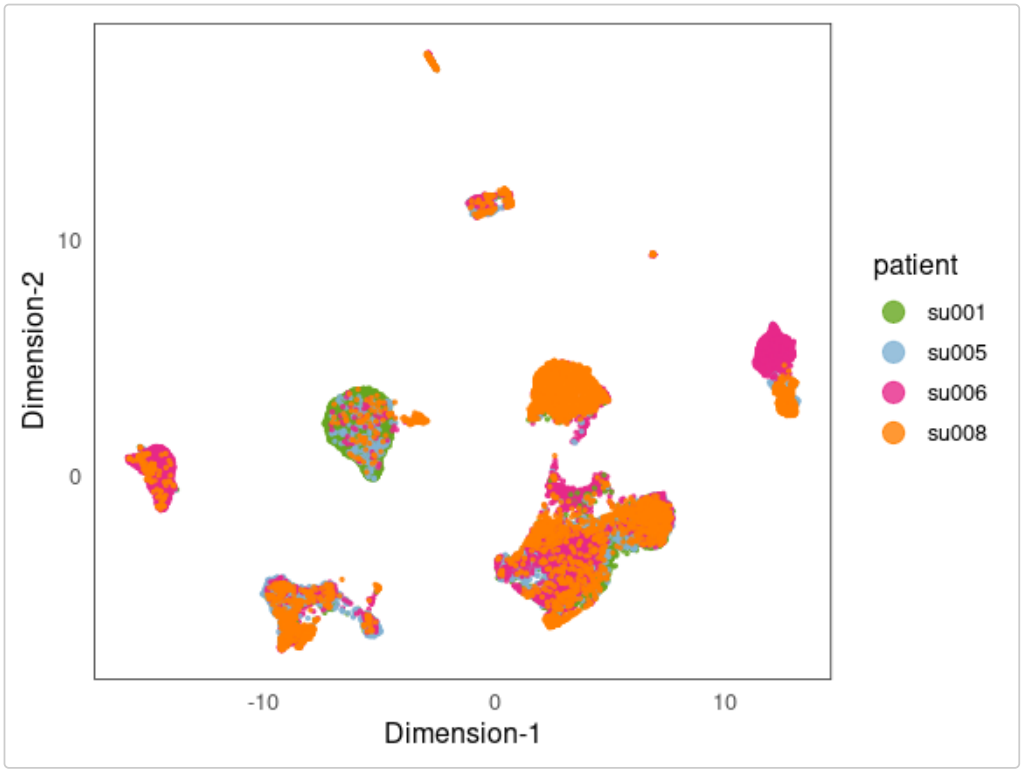
```
## [1] 34707    50
```

Annotation

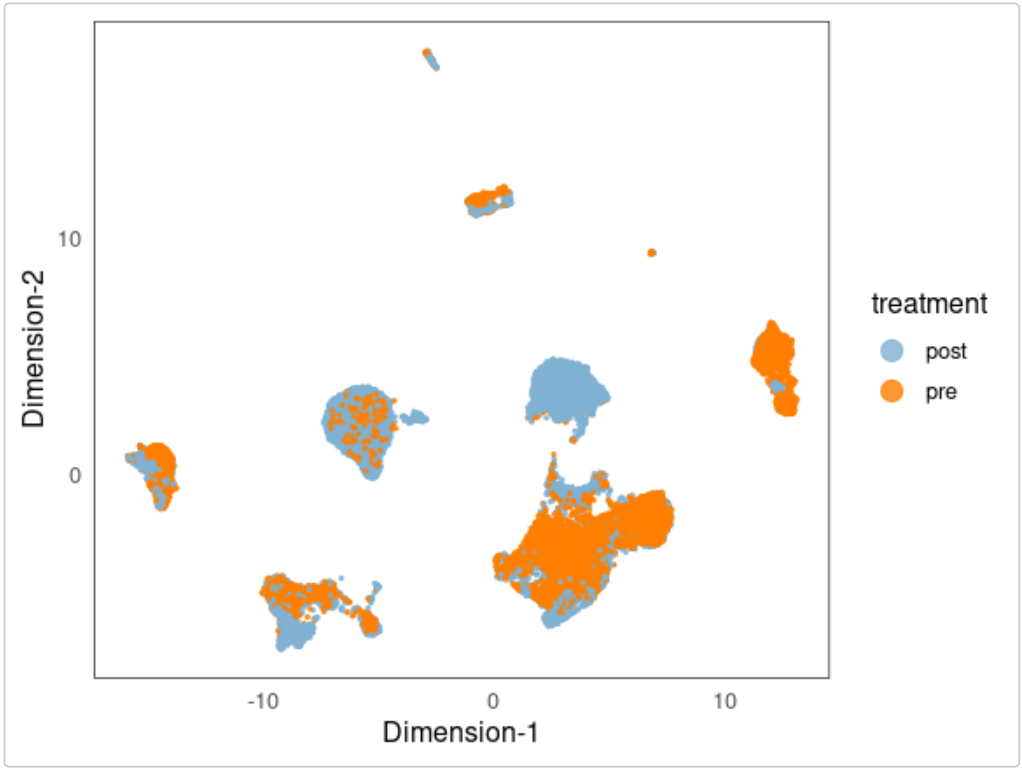
```
# Using our annotated cell-type information
Anno = read.table(
  paste0(PATH, "/Raw_Data/bcc_annotation.tsv"), sep = "\t",
  header = T, stringsAsFactors = F
)
data0@coldata$patient = as.character(Anno$patient)
data0@coldata$treatment = as.character(Anno$treatment)
data0@coldata$Type = as.character(Anno$type)
```

Patient and treatment

```
# Using our annotated cell-type information
DimPlot(data0, slot = "cell.umap", colFactor = "patient")
```



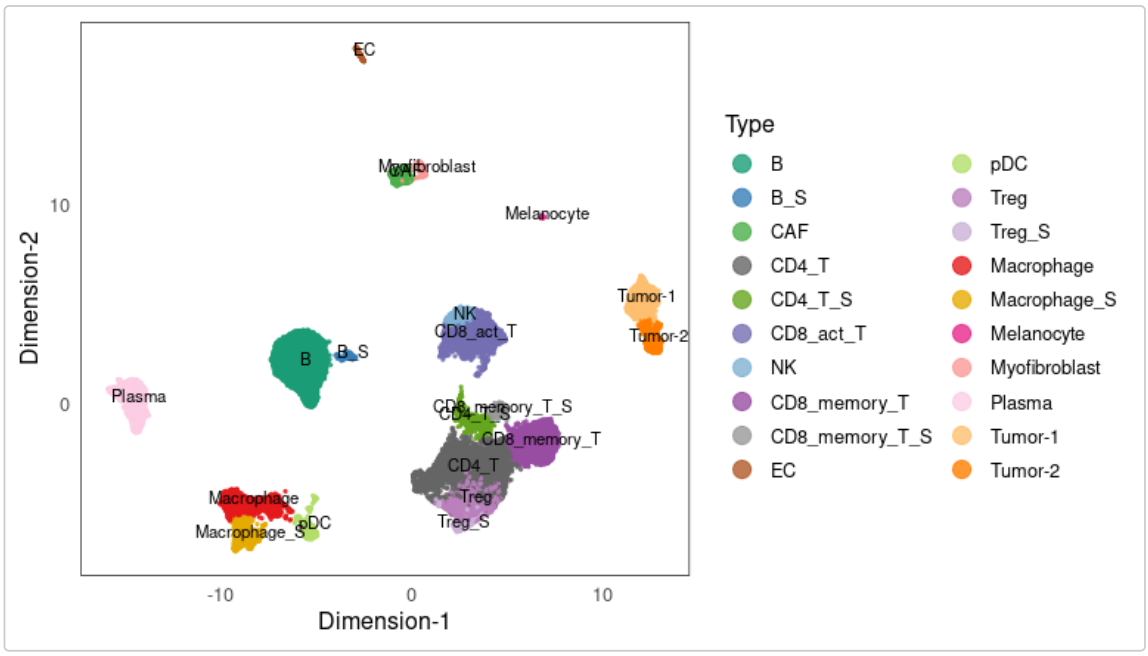

```
DimPlot(data0, slot = "cell.umap", colFactor = "treatment")
```



Cell populations

```
data0@coldata$Type = factor(
  data0@coldata$Type,
  levels = c(
    "B", "B_S", "CAF", "CD4_T", "CD4_T_S", "CD8_act_T", "NK", "CD8_memory_T",
    "CD8_memory_T_S", "EC", "pDC", "Treg", "Treg_S", "Macrophage", "Macrophage_S",
```

```
    "Melanocyte", "Myofibroblast", "Plasma", "Tumor-1", "Tumor-2"
  )
)
DimPlot(data0, slot = "cell.umap", colFactor = "Type", label = T, label.font = 3)
```



Cell clustering by igraph

The integrated cell-eigenvectors can be extracted and be used to run cell clustering, using other packages.

```
## Cell clutering by igraph
## If you already installed "FNN" and "igraph" packages, you could run the codes.
# library(FNN)
# library(igraph)

# Obtain integrated cell-eigenvectors
pls0 = as.matrix(data0@DimReduction$cell.pls[,1:18])

## Clustering cells by louvain algorithm
# clust0 = get.knn(pls0, k = 4, algorithm = 'kd_tree')
# clust1 = data.frame(
#   NodStar = rep(1L:nrow(pls0), 4),
#   NodEnd = as.vector(clust0$nn.index), stringsAsFactors = FALSE
# )
# clust1 = graph_from_data_frame(clust1, directed = FALSE)
# clust1 = simplify(clust1)
# clust1 = cluster_louvain(clust1, weights = 1/(1 + as.vector(clust0$nn.dist)))
# data0@cluster = data0@coldata$Cluster = as.factor(clust1$membership)
# names(data0@cluster) = names(data0@coldata$Cluster) = rownames(pls0)
# print(table(data0@coldata$Cluster))
```

Cell clustering

```
## Cell clutering by igraph or densityclust, wrapped in RISC

# Louvain algorithm
# "slot = cell.pls" indicates to cluster cells by the integrated cell-eigenvectors,
# "method = louvain" for "louvain" algorithm, and "npc" for the number of
```

```
# cell-eigenvectors
data0 = scCluster(data0, slot = "cell.pls", neighbor = 3, method = "louvain", npc = 18)
print(table(data0@coldata$Cluster))
```

```
##
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
## 1231  379  116 3766  804  921  532 1587 2421  689 3747 2141 1544 1784   59 1031
##   17   18   19   20   21   22   23
##  554 3718 1136  300  597 5283  367
```

```
# DimPlot(data0, slot = "cell.umap", colFactor = "Cluster", label = T)

# Density peak clustering algorithm
# "slot = cell.umap" indicates to cluster cells by UMAP geography,
# "method = density" for "the density peak clustering" algorithm, and "k" for the
# number of clusters we want.
data0 = scCluster(data0, slot = "cell.umap", method = "density", k = 35)
```

```
## Distance cutoff calculated to 0.461532
```

```
print(table(data0@coldata$Cluster))
```

```
##
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
## 3773 2005 5984 3715 1191 1041  838  774  361 5916 1771 1801  951  811  716  362
##   17   18   19   20   21   22   23   24   25   26   27   28   29   30   31   32
##  342   58  675  423  205  285  292  298   63    1    2    2    1    8    1   34
##   33   34   35
##    1    4    2
```

```
# DimPlot(data0, slot = "cell.umap", colFactor = "Cluster", label = T)
```

Identify marker genes

As the gene expression values are already corrected during data integration, so we could use them directly to find the marker genes or detect DE genes.

```
# Using our annotated cell-type information
data0@coldata$Cluster = as.factor(data0@coldata$Type)
print(table(data0@coldata$Cluster))
```

```
##
##             B           B_S           CAF          CD4_T        CD4_T_S
##          5984           205           821          6205           812
##      CD8_act_T            NK   CD8_memory_T CD8_memory_T_S            EC
##          3447           944          3781           362           361
##           pDC          Treg        Treg_S     Macrophage   Macrophage_S
##           871          2002           228          2151           920
##     Melanocyte  Myofibroblast        Plasma       Tumor-1       Tumor-2
##            58           315          2428          1771          1041
```

```
## The funtions "scMarker" and "AllMarker" are used to find the marker
## genes, and both functions support multiple cores, but still very
## time consuming.

## Find the marker genes for one cell population (not running)
# marker1 = scMarker(data0, cluster = "Macrophage", ncore = 2)
# print(head(marker1))
## Find the marker genes for all cell populations (not running)
# marker0 = AllMarker(data0, ncore = 1)
# print(table(marker0$Cluster))
```

Expression patterns

```
UMAPlot(data0, genes = "BLK", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "PODXL", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "XCL2", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "CD8A", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "FBLN2", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "RGS5", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "NCDN", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "CD40LG", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "SLC16A10", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "FRZB", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "KLRC1", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "FOXP3", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "MNDA", size = 0.75, exp.col = "firebrick2", legend = FALSE)
UMAPlot(data0, genes = "IGHG4", size = 0.75, exp.col = "firebrick2", legend = FALSE)
```

Detect DE genes

The differentially expressed (DE) genes are detected from the integrated data directly, using "Negative Binomial" model or "Quasi-Poisson" model.

Output corrected gene expression matrix

The corrected gene expression values can also be used by other packages

```
# The corrected gene expression values
logmat0 = data0@assay$logcount
print(dim(logmat0))
```

```
## [1] 12764 34707
```

Find DE genes in macrophage before and after therapy

```
## Macrophage vs Macrophage_S

# Find DE genes of macrophage in patient su006
cell1 = rownames(data0@coldata)[
  data0@coldata$Type == "Macrophage" & data0@coldata$patient == "su006"
] # Cell barcode
cell2 = rownames(data0@coldata)[
  data0@coldata$Type == "Macrophage_S" & data0@coldata$patient == "su006"
]
# Utilize cell barcode to define the control and sample cells, then run DE gene analysis.
DE.Mac1 = scDEG(data0, cell.ctrl = cell1, cell.sam = cell2, ncore = 2)
print(dim(DE.Mac1))
```

```
## [1] 5132    6
```

```
# Find DE genes of macrophage in patient su008
cell1 = rownames(data0@coldata)[
  data0@coldata$Type == "Macrophage" & data0@coldata$patient == "su008"
]
cell2 = rownames(data0@coldata)[
  data0@coldata$Type == "Macrophage_S" & data0@coldata$patient == "su008"
]
DE.Mac2 = scDEG(data0, cell.ctrl = cell1, cell.sam = cell2, ncore = 2)
print(dim(DE.Mac2))
```

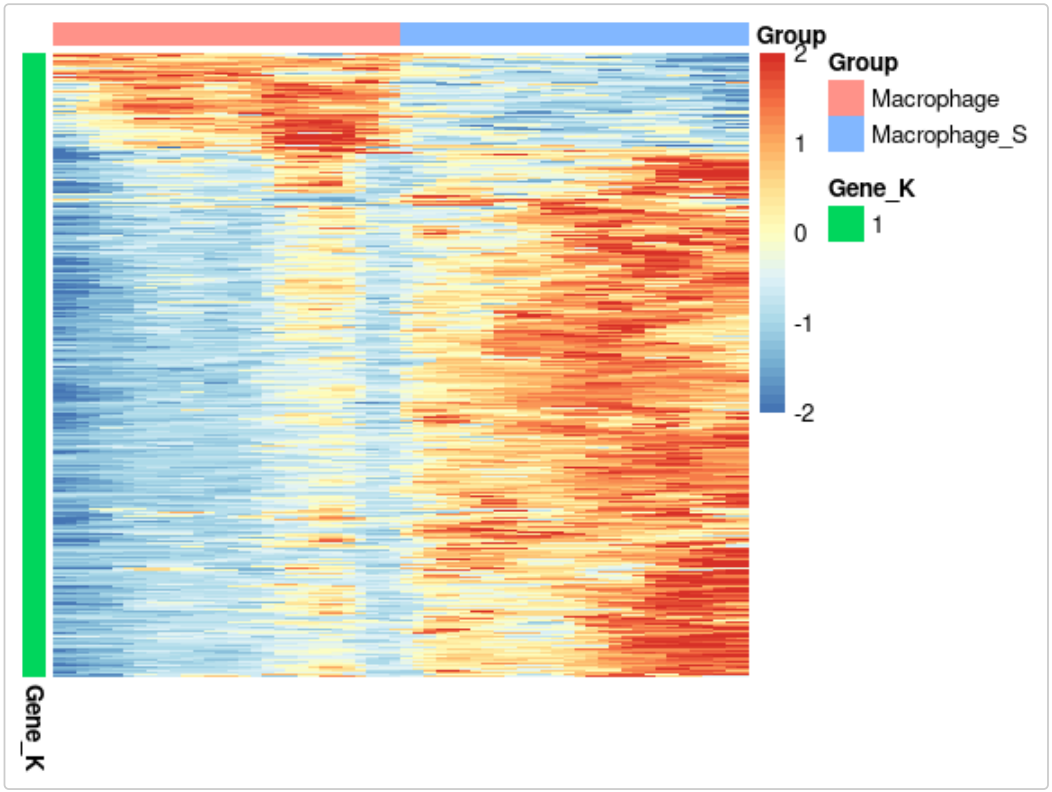
```
## [1] 5942    6
```

```
# Shared DE genes in both patients with restrictive threshold
DE.Mac1 = DE.Mac1[DE.Mac1$Padj < 0.05 & abs(DE.Mac1$logFC) > 0.5,]
DE.Mac2 = DE.Mac2[DE.Mac2$Padj < 0.05 & abs(DE.Mac2$logFC) > 0.5,]
DE1 = intersect(DE.Mac1$Symbol[DE.Mac1$logFC > 0], DE.Mac2$Symbol[DE.Mac2$logFC > 0])
DE2 = intersect(DE.Mac1$Symbol[DE.Mac1$logFC < 0], DE.Mac2$Symbol[DE.Mac2$logFC < 0])
DE0 = unique(c(DE1, DE2))

## DE genes of macrophage in all patients
# cell1 = rownames(data0@coldata)[data0@coldata$Type == "Macrophage"]
```

```
# cell2 = rownames(data0@coldata)[data0@coldata$Type == "Macrophage_S"]
# DE.Mac0 = scDEG(data0, cell.ctrl = cell1, cell.sam = cell2, ncore = 2)
# print(dim(DE.Mac0))
```

Heatmap of DE genes

```
# Select the macrophage
cell0 = rownames(data0@coldata)[
  data0@coldata$Type %in% c("Macrophage", "Macrophage_S")
]
# Subset the integrated data with macrophage
dats = SubSet(data0, cells = cell0)
# Show heatmap of DE genes in macrophage
Heat(dats, colFactor = "Type", genes = DE0, lim = 2, num = 30)
```



Output for PAGA

We output the integrated data (cell eigenvectors and gene expression values) for scanpy.

```
# Select the macrophage
cell0 = rownames(data0@coldata)[
  data0@coldata$Type %in% c("Macrophage", "Macrophage_S")
]
# Subset the integrated data with macrophage
dats = SubSet(data0, cells = cell0)

# Output the selected eigenvectors
eigen0 = 18
# Output the integrated cell-eigenvectors
pls0 = as.matrix(dats@DimReduction$cell.pls)[,1:eigen0]
write.table(
  pls0, file = paste0(PATH, "/Raw_Data/pls.tsv"),
```

```
  sep = "\t", col.names = F, row.names = F, quote = F
)
# Output the corrected gene expression values
logmat0 = dats@assay$logcount
writeMM(logmat0, file = paste0(PATH, "/Raw_Data/logmat.mtx"))
```

```
## NULL
```

```
# Output the cell information
coldatas = as.data.frame(
  dats@coldata[,c("Barcode", "patient", "treatment", "sort", "Type")]
)
write.table(
  coldatas, file = paste0(PATH, "/Raw_Data/cell.tsv"),
  sep = "\t", col.names = T, row.names = T, quote = F
)
# Output the gene information
rowdatas = as.data.frame(dats@rowdata)
write.table(
  rowdatas, file = paste0(PATH, "/Raw_Data/gene.tsv"),
  sep = "\t", col.names = T, row.names = T, quote = F
)
```

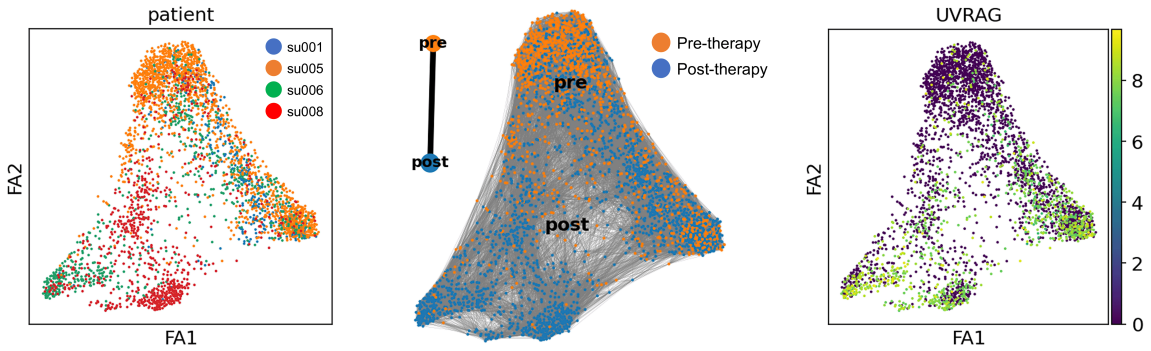
The part in python environment is shown in next page.

In Python environment

Trajectory of macrophage

We construct a trajectory of macrophage according to integrated cell-eigenvectors and corrected gene expression values.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as pl
import scanpy as sc

sc.settings.set_figure_params(dpi=100, format='png')
PATH = "/home/yang/Documents/Vignette/GSE123813"

# Creat scanpy object from the integrated data
adata = sc.read_mtx(PATH + "/Raw_Data/logmat.mtx").T
adata.obs = pd.read_csv(PATH + "/Raw_Data/cell.tsv", sep="\t", index_col=0)
adata.var = pd.read_csv(PATH + "/Raw_Data/gene.tsv", sep="\t", index_col=0)
adata.obs['treatment'] = adata.obs['treatment'].astype('category')
print(adata)

# Input the integrated cell-eigenvectors
PC0 = pd.read_csv(PATH + "/Raw_Data/pls.tsv", sep="\t", header=None, index_col=False)
# PC0 = PC0.iloc
adata.obsm["X_pca"] = PC0.values

# Calculate FA from the integrated cell-eigenvectors
sc.pp.neighbors(adata, n_neighbors=30, n_pcs=18)
sc.tl.draw_graph(adata)

# PAGA trajectory
sc.tl.paga(adata, groups="treatment")
sc.pl.paga_compare(
  adata, threshold=0.15, size=15, title="", legend_fontsize=12, fontsize=12,
  frameon=False, edges=True, show=False, save="connet"
)
sc.pl.draw_graph(adata, color="patient", size=15, show=False, save="patient")
sc.pl.draw_graph(adata, color="UVRAG", size=15, show=False, save="Gene")
```



Session Information

```
sessionInfo()
```

```
## R version 3.6.3 (2020-02-29)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Linux Mint 20
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] reticulate_1.16    RColorBrewer_1.1-2 ggplot2_3.3.2      Matrix_1.2-18
## [5] RISC_1.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.1.0   xfun_0.16          purrr_0.3.4        splines_3.6.3
##  [5] lattice_0.20-41    colorspace_1.4-1   vctrs_0.3.3        generics_0.0.2
##  [9] htmltools_0.5.0    mgcv_1.8-33        yaml_2.2.1         rlang_0.4.7
## [13] pillar_1.4.6       glue_1.4.2         withr_2.2.0        matrixStats_0.56.0
## [17] foreach_1.5.0      lifecycle_0.2.0    stringr_1.4.0      umap_0.2.6.0
## [21] munsell_0.5.0      gtable_0.3.0       codetools_0.2-16   evaluate_0.14
## [25] labeling_0.3       knitr_1.29         doParallel_1.0.15  irlba_2.3.3
## [29] parallel_3.6.3     pbmcapply_1.5.0    Rcpp_1.0.5         openssl_1.4.2
## [33] scales_1.1.1       densityClust_0.3   jsonlite_1.7.0     farver_2.0.3
## [37] FNN_1.1.3          RSpectra_0.16-0    gridExtra_2.3      askpass_1.1
## [41] digest_0.6.25      stringi_1.4.6      Rtsne_0.15         dplyr_1.0.2
## [45] ggrepel_0.8.2      grid_3.6.3         tools_3.6.3        magrittr_1.5
## [49] tibble_3.0.3       crayon_1.3.4       pkgconfig_2.0.3    MASS_7.3-52
## [53] ellipsis_0.3.1     pheatmap_1.0.12    data.table_1.13.0  rmarkdown_2.3
## [57] iterators_1.0.12   R6_2.4.1           nlme_3.1-149       igraph_1.2.5
## [61] compiler_3.6.3
```