Slingshot

Claire Man

3/11/2020

```
library(SingleCellExperiment, quietly = TRUE)
```

```
## Warning: package 'SummarizedExperiment' was built under R version 3.6.2
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename, cbind, colnames,
##     dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,
##     grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##     rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
##     union, unique, unsplit, which, which.max, which.min
```

```
## Warning: package 'S4Vectors' was built under R version 3.6.2
```

```
##
## Attaching package: 'S4Vectors'
```

```
## The following object is masked from 'package:base':
##
##     expand.grid
```

```
## Warning: package 'IRanges' was built under R version 3.6.2
```

```
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
```

```
## Warning: package 'DelayedArray' was built under R version 3.6.2
```

```
##
## Attaching package: 'matrixStats'
```

```
## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians
```

```
## Warning: package 'BiocParallel' was built under R version 3.6.2
```

```
##
## Attaching package: 'DelayedArray'
```

```
## The following objects are masked from 'package:matrixStats':
##
##     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges
```

```
## The following objects are masked from 'package:base':
##
##     aperm, apply, rowsum
```

1. Introduction

1.1 Overview

- minimum input to slingshot: a matrix representing the cells in a reduced-dimensional space and a vector of cluster labels
- Then, identify the global lineage structure by constructing an minimum spanning tree (MST) on the clusters, with the *getLineages* function.
- Construct smooth lineages and infer pseudotime variables by fitting simultaneous principal curves with the *getCurves* function.
- Built-in visualization tools.

1.2 Datasets

- The first dataset is a **"single-trajectory"** dataset. It is designed to represent a single lineage in which one third of the genes are associated with the transition **(?)**.

```
# generate synthetic count data representing a single lineage
means <- rbind(
    # non-DE genes
    matrix(rep(rep(c(0.1,0.5,1,2,3), each = 300),100),
        ncol = 300, byrow = TRUE),
    # early deactivation
    matrix(rep(exp(atan( ((300:1)-200)/50 )),50), ncol = 300, byrow = TRUE),
    # late deactivation
    matrix(rep(exp(atan( ((300:1)-100)/50 )),50), ncol = 300, byrow = TRUE),
    # early activation
    matrix(rep(exp(atan( ((1:300)-100)/50 )),50), ncol = 300, byrow = TRUE),
    # late activation
    matrix(rep(exp(atan( ((1:300)-200)/50 )),50), ncol = 300, byrow = TRUE),
    # transient
    matrix(rep(exp(atan( c((1:100)/33, rep(3,100), (100:1)/33) )),50),
        ncol = 300, byrow = TRUE)
)
counts <- apply(means,2,function(cell_means){
    total <- rnbinom(1, mu = 7500, size = 4)
    rmultinom(1, total, cell_means)
```

```
})
rownames(counts) <- paste0('G',1:750)
colnames(counts) <- paste0('c',1:300)
sim <- SingleCellExperiment(assays = List(counts = counts))
```

- The second dataset is a **"bifurcating"** dataset. It consists of a matrix of coordinates along with cluster labels generated by k-means clustering.

```
library(slingshot, quietly = TRUE)
```

```
data("slingshotExample")
dim(rd) # data representing cells in a reduced dimensional space
```

```
## [1] 140   2
```

```
length(cl) # vector of cluster labels
```

```
## [1] 140
```

2. Upstream Analysis

2.1 Gene Filtering

- Reduce the dimensionality of data and filtering out uninformative genes
- Keep any genes robustly expressed in at least enough cells to constitute a cluster
- "Robustly expressed": if it has a simulated count of at least 10 reads

```
# filter genes down to potential cell-type markers
# at least M (15) reads in at least N (15) cells
geneFilter <- apply(assays(sim)$counts,1,function(x){
    sum(x >= 3) >= 10
})
sim <- sim[geneFilter, ]
```

2.2 Normalization

- Remove unwanted technical or biological artifacts from the data (batch, sequencing depth, cell cycly effects, etc.)

- *scone* package
- Here we are using simulated data, no need to worry about batch effects.

```
FQnorm <- function(counts){
    rk <- apply(counts,2,rank,ties.method='min')
    counts.sort <- apply(counts,2,sort)
    refdist <- apply(counts.sort,1,median)
    norm <- apply(rk,2,function(r){ refdist[r] })
    rownames(norm) <- rownames(counts)
    return(norm)
}
assays(sim)$norm <- FQnorm(assays(sim)$counts)
```

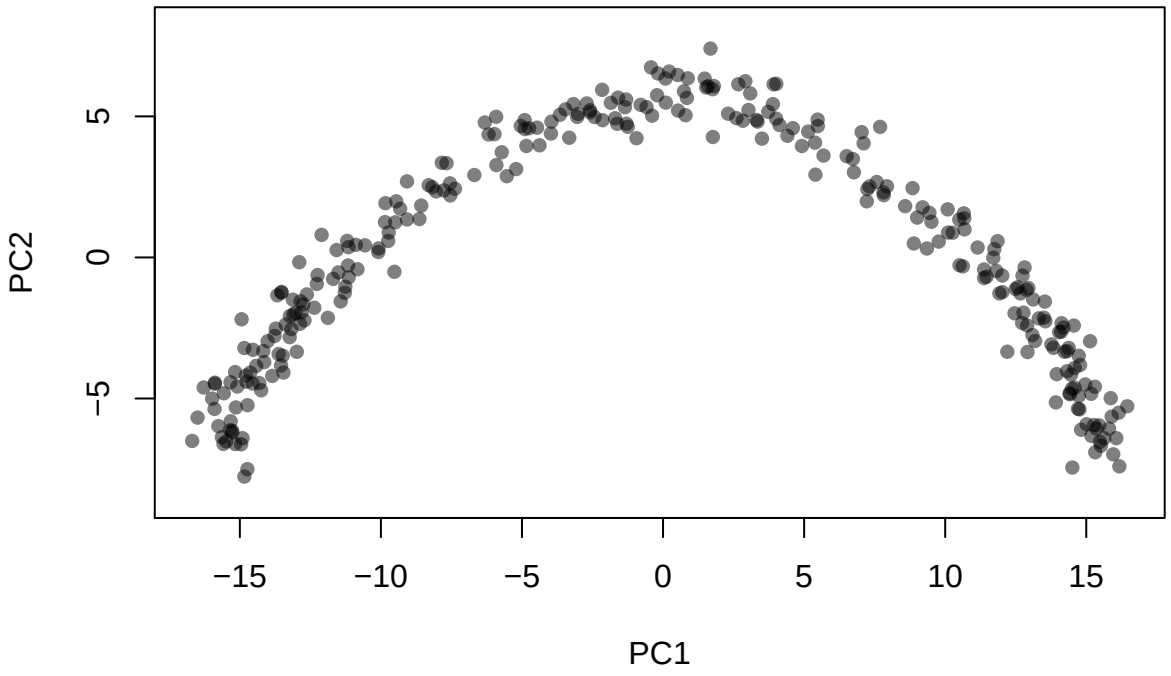
2.3 Dimensionality Reduction

- Fundamental Assumption of slingshot: cells which are transcriptionally similiar will be close to each other in some reduced-dimensional space

- PCA and diffusion maps (from *destiny* package)

PCA

```
pca <- prcomp(t(log1p(assays(sim)$norm)), scale. = FALSE)
rd1 <- pca$x[,1:2]

plot(rd1, col = rgb(0,0,0,.5), pch=16, asp = 1)
```



Diffusion Maps

```
library(destiny, quietly = TRUE)
```

```
## Warning: package 'destiny' was built under R version 3.6.2
```

```
##
## Attaching package: 'destiny'
```

```
## The following object is masked from 'package:SummarizedExperiment':
##
##     distance
```

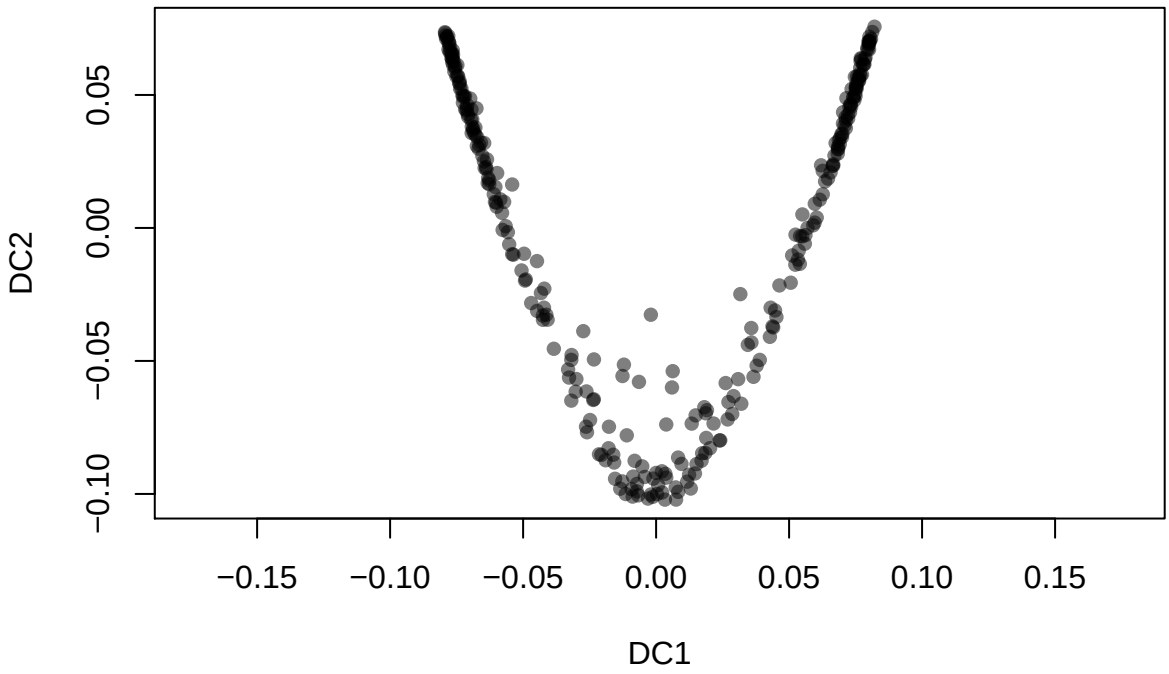
```
## The following object is masked from 'package:GenomicRanges':
##
##     distance
```

```
## The following object is masked from 'package:IRanges':
##
##     distance
```

```
dm <- DiffusionMap(t(log1p(assays(sim)$norm)))
```

```
## Warning in DiffusionMap(t(log1p(assays(sim)$norm))): You have 737 genes.
## Consider passing e.g. n_pcs = 50 to speed up computation.
```

```
rd2 <- cbind(DC1 = dm$DC1, DC2 = dm$DC2)

plot(rd2, col = rgb(0,0,0,.5), pch=16, asp = 1)
```




```
reducedDims(sim) <- SimpleList(PCA = rd1, DiffMap = rd2) #adding both dimensionality reductions to the
```

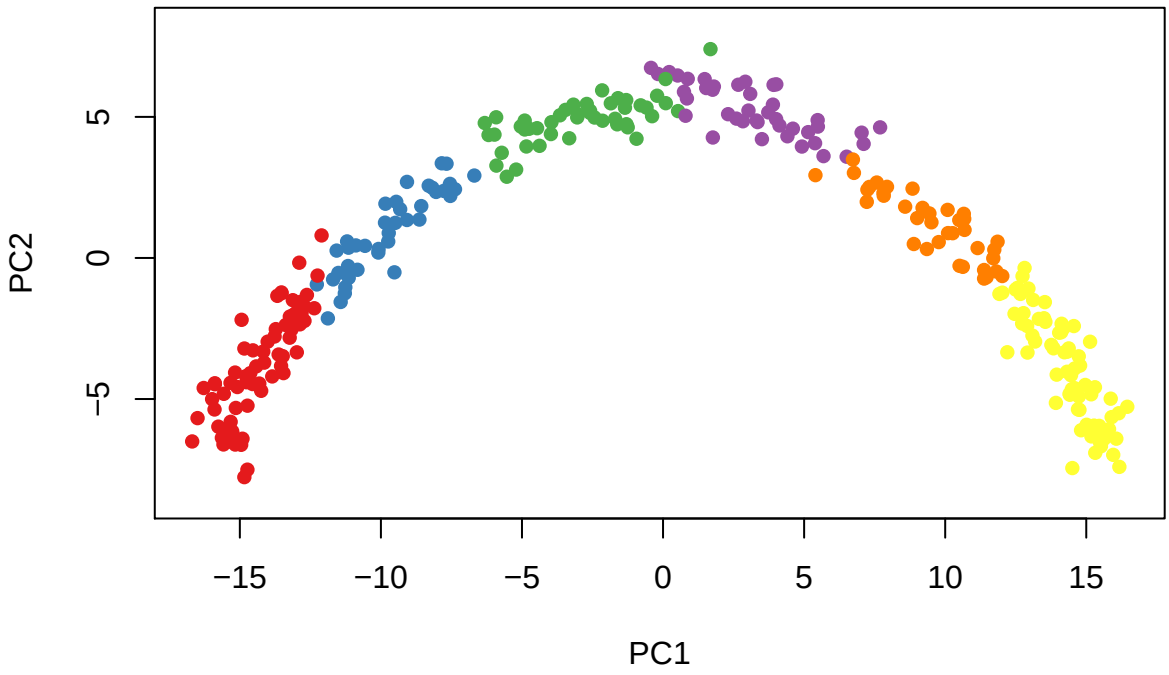
2.4 Clustering Cells

- The final input to slingshot is a vector of cluster labels for the cells.
- First clustering method is Gaussian mixture modeling: *mclust* package, features an automated method for determinging the number of clusters based on BIC

```
library(mclust, quietly = TRUE)
```

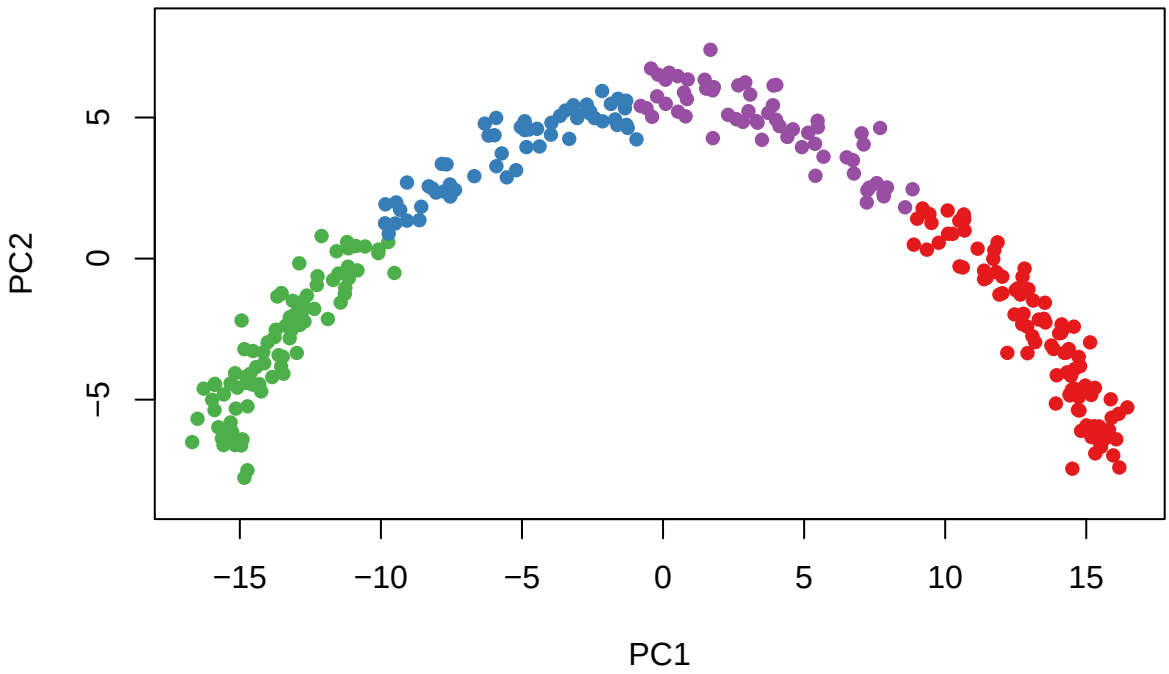
```
## Package 'mclust' version 5.4.5
## Type 'citation("mclust")' for citing this R package in publications.
```

```
cl1 <- Mclust(rd1)$classification
colData(sim)$GMM <- cl1

library(RColorBrewer)
plot(rd1, col = brewer.pal(9,"Set1")[cl1], pch=16, asp = 1)
```



- The second clustering method is k-means.

```
cl2 <- kmeans(rd1, centers = 4)$cluster
colData(sim)$kmeans <- cl2

plot(rd1, col = brewer.pal(9,"Set1")[cl2], pch=16, asp = 1)
```



3. Using Slingshot

- Slingshot processes

  - Identify the global lineage structure with a cluster-based minimum spanning tree (MST)
  - Fitting simultaneously pricipal curves to describe each lineage
- These two processes can be done together using *wrapper* function.

```
sim <- slingshot(sim, clusterLabels = 'GMM', reducedDim = "PCA")
```

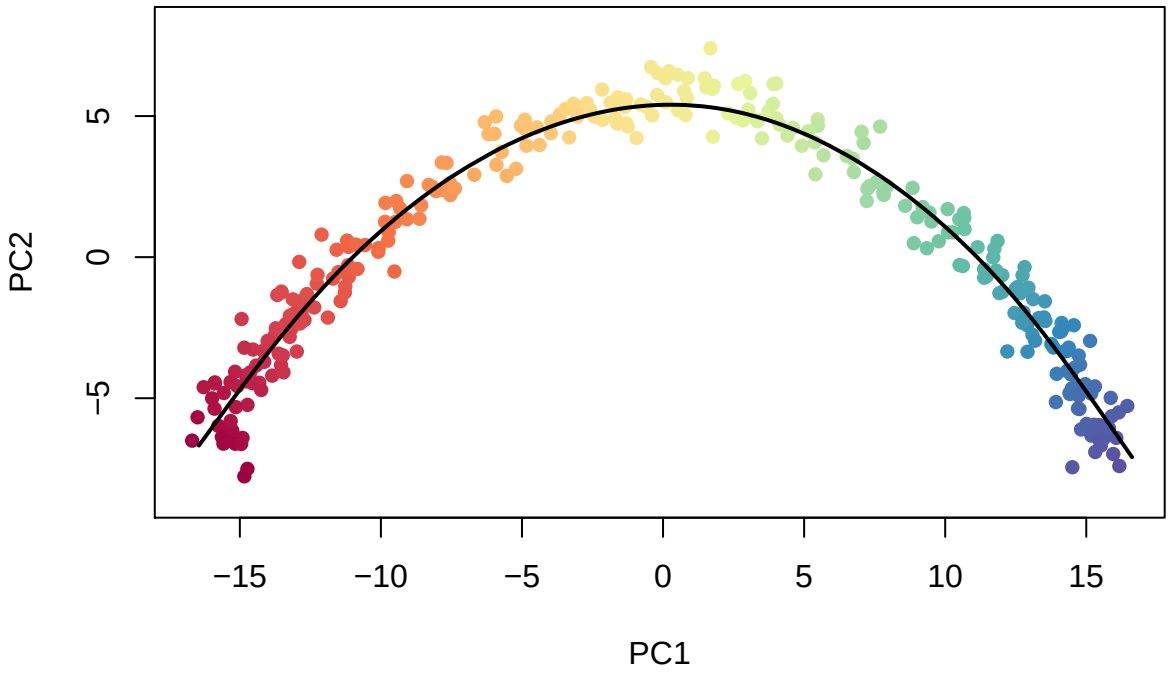
```
## Using full covariance matrix
```

- Next, visualize the inferred lineage for the single-trajectory data with points colored by pseudotime.

```
summary(sim$slingPseudotime_1)
```

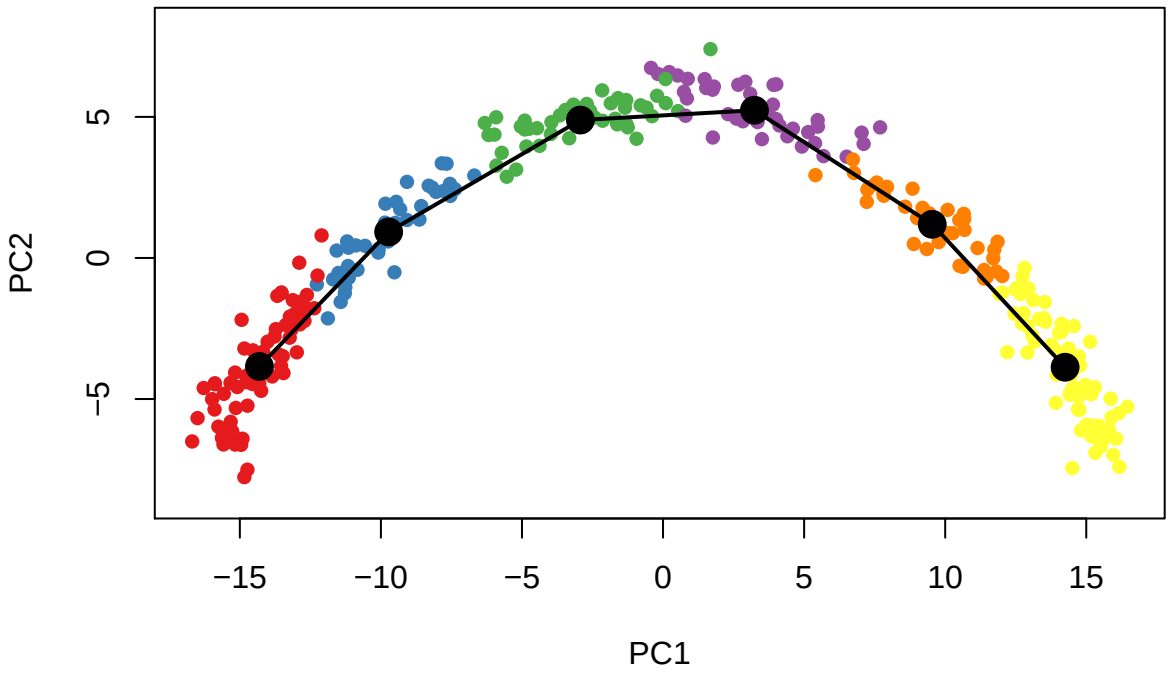
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   8.386  21.155  21.198  34.157  42.877
```

```
colors <- colorRampPalette(brewer.pal(11,'Spectral')[-6])(100)
plotcol <- colors[cut(sim$slingPseudotime_1, breaks=100)]

plot(reducedDims(sim)$PCA, col = plotcol, pch=16, asp = 1)
lines(SlingshotDataSet(sim), lwd=2, col='black')
```



- see how the lineage structure was intially estimated by the cluster-based minimum spanning tree by using the *type* argument (type = lineages).

```
plot(reducedDims(sim)$PCA, col = brewer.pal(9,'Set1')[sim$GMM], pch=16, asp = 1)
lines(SlingshotDataSet(sim), lwd=2, type = 'lineages', col = 'black')
```



4. Downstream Analysis

4.1 Identifying temporally expressed genes

- Next step: find genes that change their expression over the course of development
- Demonstration: on the 100 most variable genes
  - Regress each gene on the pseudotime variable we have generated, using a general additive model (GAM, generalized linear model)

```
sim # investigate what sim is
```

```
## class: SingleCellExperiment
## dim: 737 300
## metadata(0):
## assays(2): counts norm
## rownames(737): G2 G3 ... G749 G750
## rowData names(0):
## colnames(300): c1 c2 ... c299 c300
## colData names(3): GMM kmeans slingPseudotime_1
## reducedDimNames(2): PCA DiffMap
## spikeNames(0):
## altExpNames(0):
```

```
# sim$slingPseudotime_1
```

```
require(gam)
```

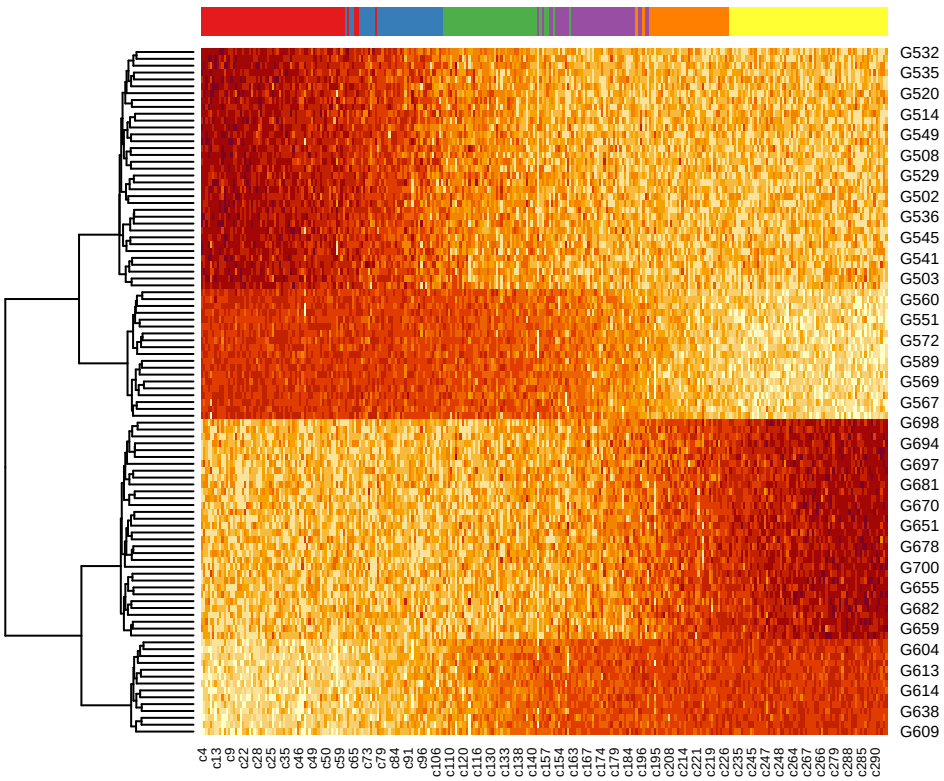
```
## Loading required package: gam
## Loading required package: splines
## Loading required package: foreach
```

```
## Loaded gam 1.16.1
```

```
t <- sim$slingPseudotime_1

# for time, only look at the 100 most variable genes
Y <- log1p(assays(sim)$norm) # log1p computes log(1+x) accurately
var100 <- names(sort(apply(Y,1,var),decreasing = TRUE))[1:100]
Y <- Y[var100,]

# fit a GAM with a loess term for pseudotime
gam.pval <- apply(Y,1,function(z){
    d <- data.frame(z=z, t=t)
    suppressWarnings({
      tmp <- suppressWarnings(gam(z ~ lo(t), data=d))
    })
    p <- summary(tmp)[3][[1]][2,3]
    p
})
```

- Pick out the top genes based on p-values and visualize their expression over developmental time with a heatmap
  Has different output

```
topgenes <- names(sort(gam.pval, decreasing = FALSE))[1:100]
heatdata <- assays(sim)$norm[topgenes, order(t, na.last = NA)]
heatclus <- sim$GMM[order(t, na.last = NA)]

heatmap(log1p(heatdata), Colv = NA,
        ColSideColors = brewer.pal(9,"Set1")[heatclus])
```



5. Detailed Slingshot Functionality

5.1 Identifying global lineage structure

- *getLineages* function:
  - Takes an n*p matrix and a vector of clustering results of length n
  - Maps connections between adjacent clusters using a minimum spanning tree (MST) and identifies paths through these connections that represent lineages
  - Output: a **SlingshotDataSet** containing the inputs as well as the inferred MST (represented by an adjacency matrix) and lineages (ordered vectors of cluster names)

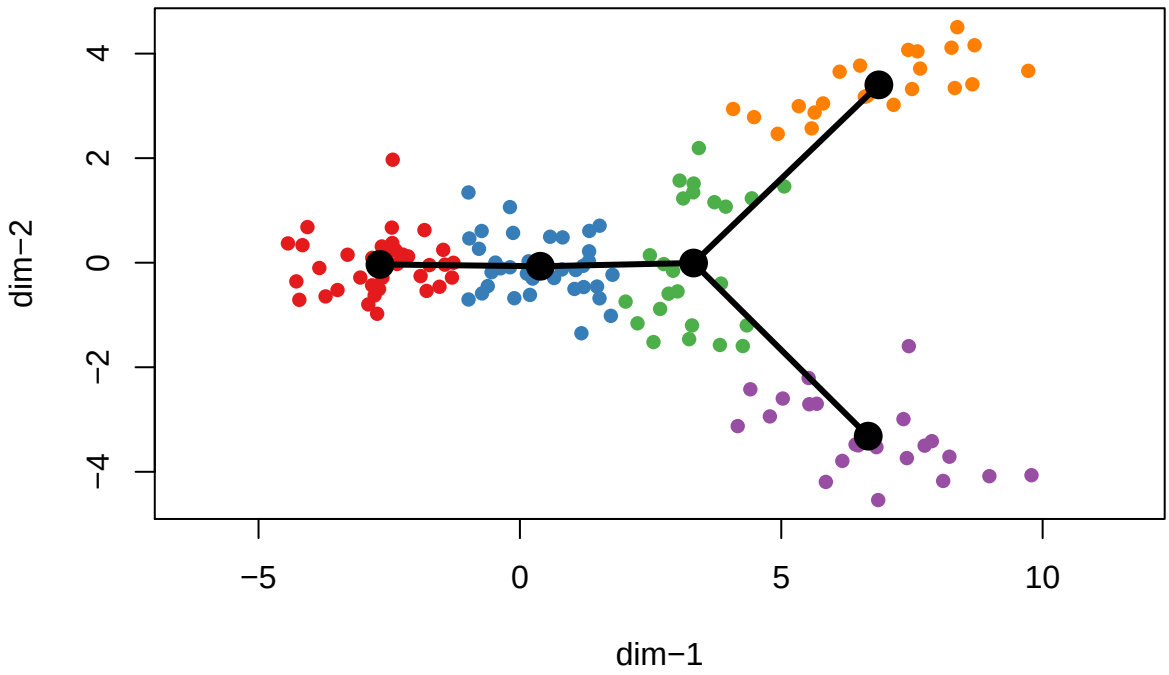
```
lin1 <- getLineages(rd, cl, start.clus = '1') # not specifying the endpoints
```

```
## Using full covariance matrix
```

```
lin1
```

```
## class: SlingshotDataSet
##
##  Samples Dimensions
##      140          2
##
## lineages: 2
## Lineage1: 1  2  3  5
## Lineage2: 1  2  3  4
##
## curves: 0
```

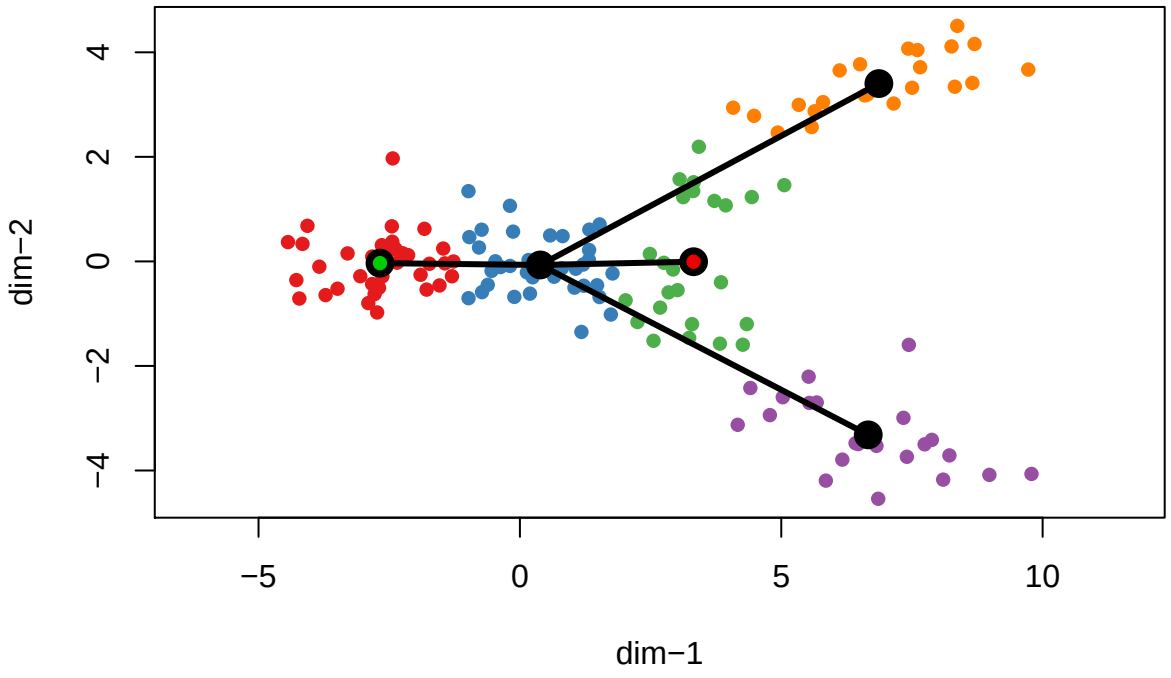
```
plot(rd, col = brewer.pal(9,"Set1")[cl], asp = 1, pch = 16)
lines(lin1, lwd = 3, col = 'black')
```




```
lin2 <- getLineages(rd, cl, start.clus= '1', end.clus = '3')
```

```
## Using full covariance matrix
```

```
plot(rd, col = brewer.pal(9,"Set1")[cl], asp = 1, pch = 16)
lines(lin2, lwd = 3, col = 'black', show.constraints = TRUE)
```



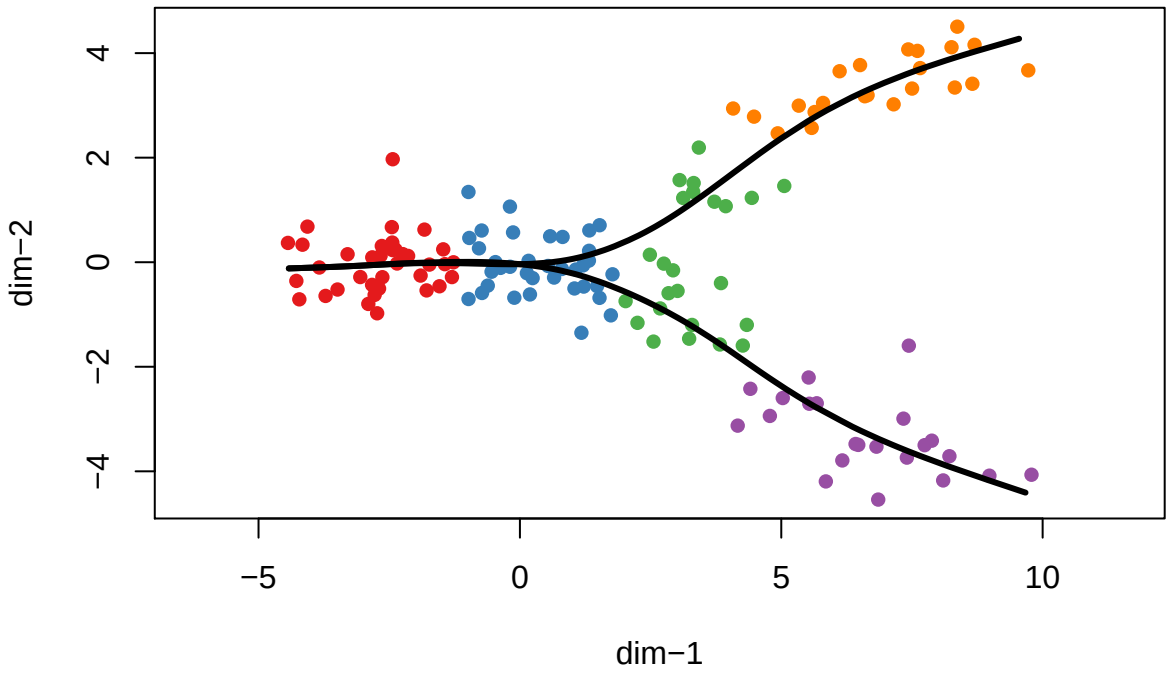
5.2 Constructing smooth curves and ordering cells

- Use *getCurves* to fit smooth curves

```
crv1 <- getCurves(lin1)
crv1
```

```
## class: SlingshotDataSet
##
##  Samples Dimensions
##      140          2
##
## lineages: 2
## Lineage1: 1  2  3  5
## Lineage2: 1  2  3  4
##
## curves: 2
## Curve1: Length: 15.045   Samples: 100.6
## Curve2: Length: 15.126   Samples: 103.5
```

```
plot(rd, col = brewer.pal(9,"Set1")[cl], asp = 1, pch = 16)
lines(crv1, lwd = 3, col = 'black')
```



Compare with Last Time

- Comparing with the workflow from Chapter 23.1-7, the basic process remains the same for both for upstream analysis.
  - Both methods had the processes of gene filtering, normalization and dimensionality reduction.
  - But Slingshot process required an extra process which is clustering cells because it is an assumption for slingshot.
- Besides, slingshot methods are more fully implemented and can easily get the pseudotime.