Lineage Reconstruction

Kelly Street

2016-06-22

Contents

1 Introduction

This is the third part of the Bioc2016 workshop “Analysis of single-cell RNA-seq data with R and Bioconductor.”

In this part we will cover lineage reconstruction with the `Githubpkg("kstreet13/slingshot")` package.

The goal of `slingshot` is to use clusters to uncover global structure and convert this structure into smooth lineages represented by one-dimensional variables, often called ``pseudotime.'' We give tools for learning cluster relationships in an unsupervised or semi-supervised manner and constructing smooth curves representing each lineage, with visualization methods for each step.

1.1 Basic `slingshot` analysis

The minimal input to `slingshot` is a matrix representing the cells in a reduced-dimensional space and a vector of clustering results. The analysis then procedes:

- Find connections between clusters with the `get_lineages` function, optionally specifying known start and end points.
- Construct smooth curves and pseudotime variables with the `get_curves` function.
- Assess the cluster connectivity with `plot_tree` and the curve stability with `plot_curves`.

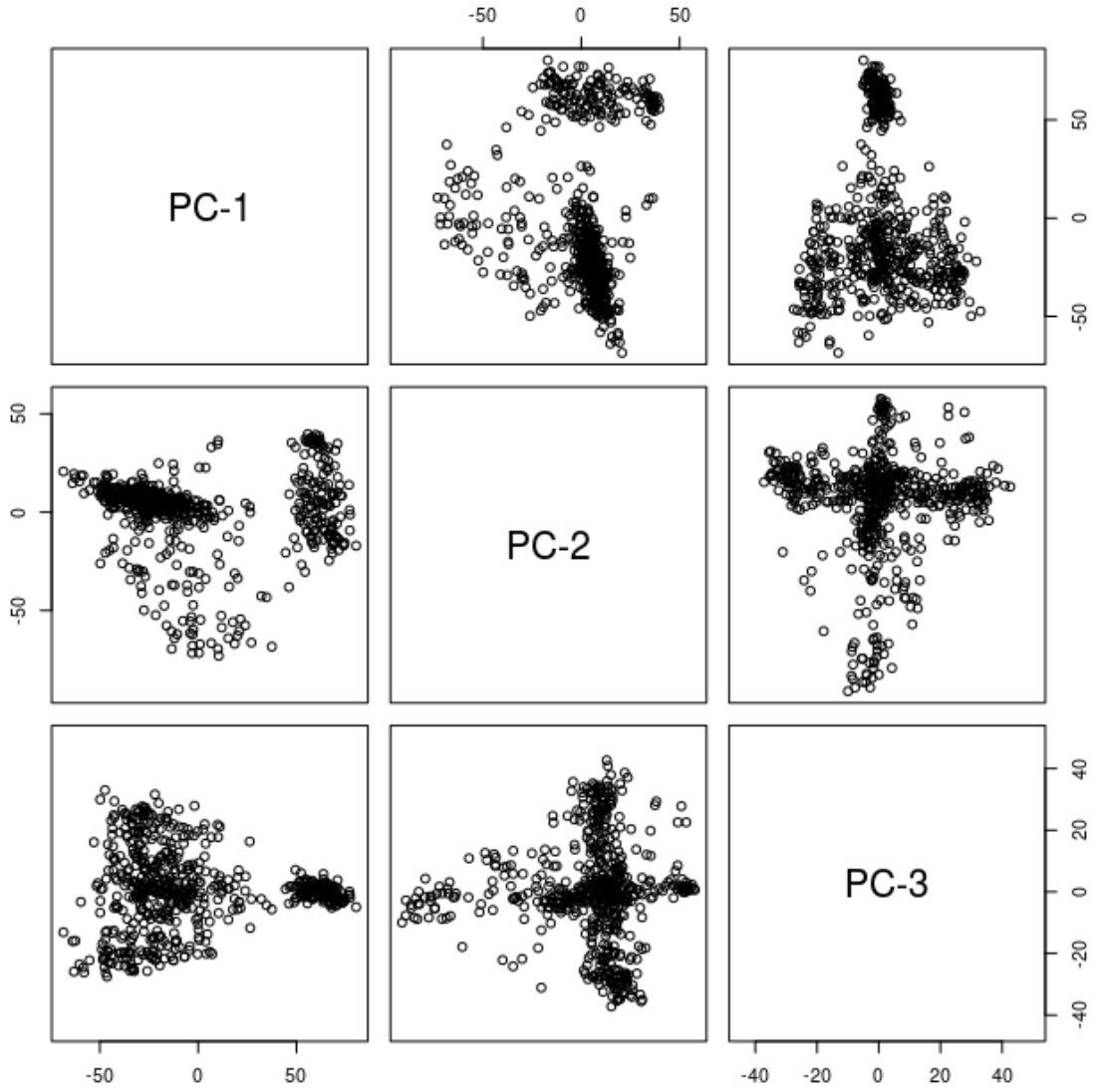
Using connections between clusters to define global structure improves the stability of inferred lineages. And our use of smooth curves in place of piecewise-linear ones reduces variability in the inferred pseudotime vectors.

1.2 An example dataset

We will take our inputs from the previous sections: the normalized counts matrix obtained with `scone` and the cluster assignments obtained with `clusterExperiment`, both of which can be loaded directly from the workshop package.

```
data('full_pca')

## Examine dimensionality reduction
# plot3d(pcaX, aspect = 'iso')
pairs(pcaX[,1:3], asp = 1)
```



2 Step 1: Assign clusters to lineages with `get_lineages`

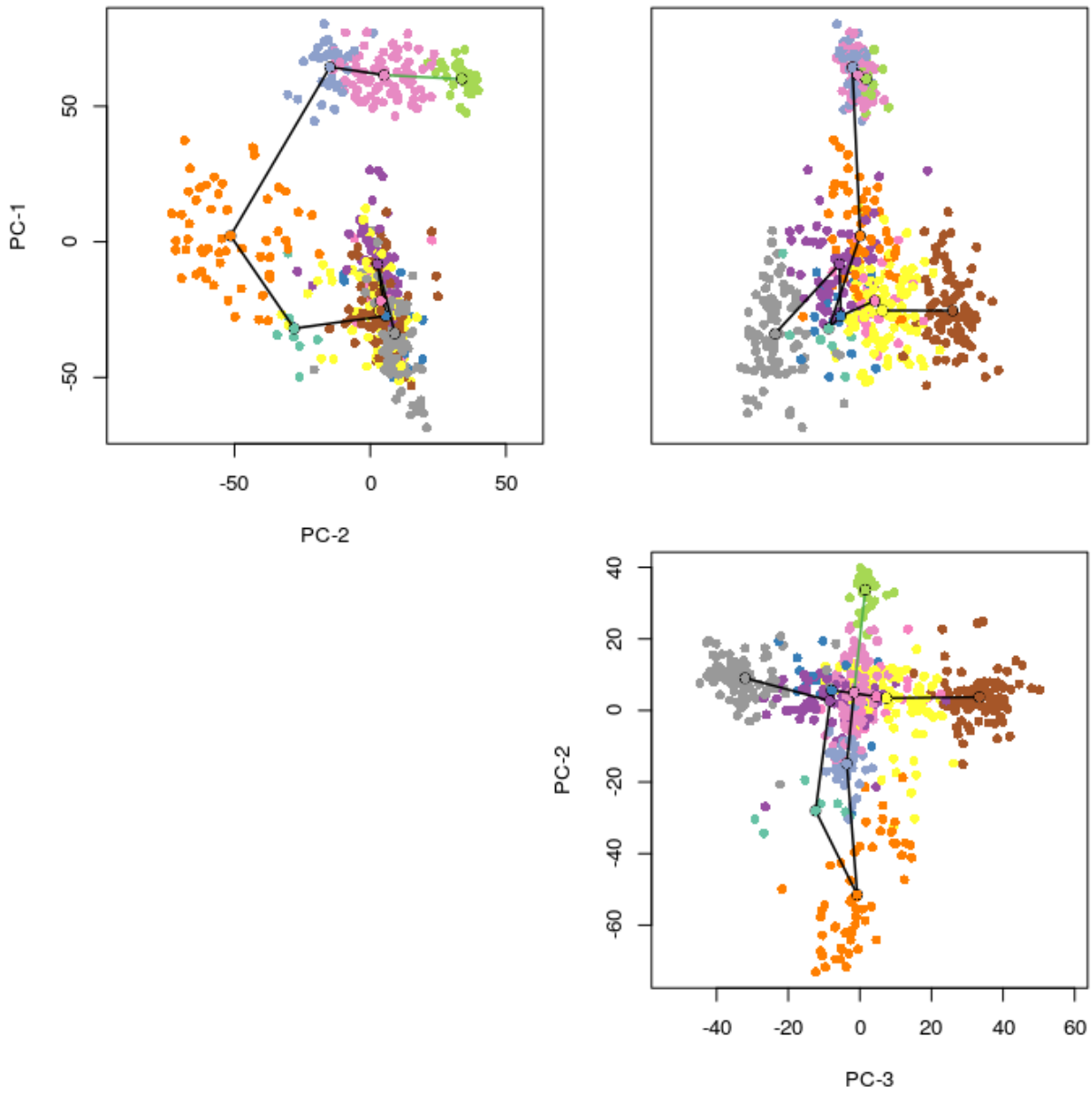
The `get_lineages` function takes as input an `n x p` matrix and a vector of clustering results of length `n`. It then maps connections between adjacent clusters using a minimum spanning tree (MST) and identifies paths through these connections that represent potential lineages.

This analysis can be performed in an entirely unsupervised manner or in a semi-supervised manner by specifying known beginning and end point clusters. We recommend that you specify a root cluster; this will have no effect on how the clusters are connected, but it will allow for nicer curves in datasets with a branching structure. Pre-specified end point clusters will be constrained to only one connection.

```
l1 <- get_lineages(pcaX, clus)
```

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

```
# plot_tree(pcaX, clus, l1, threeD = TRUE)
plot_tree(pcaX, clus, l1, dim = 3)
```

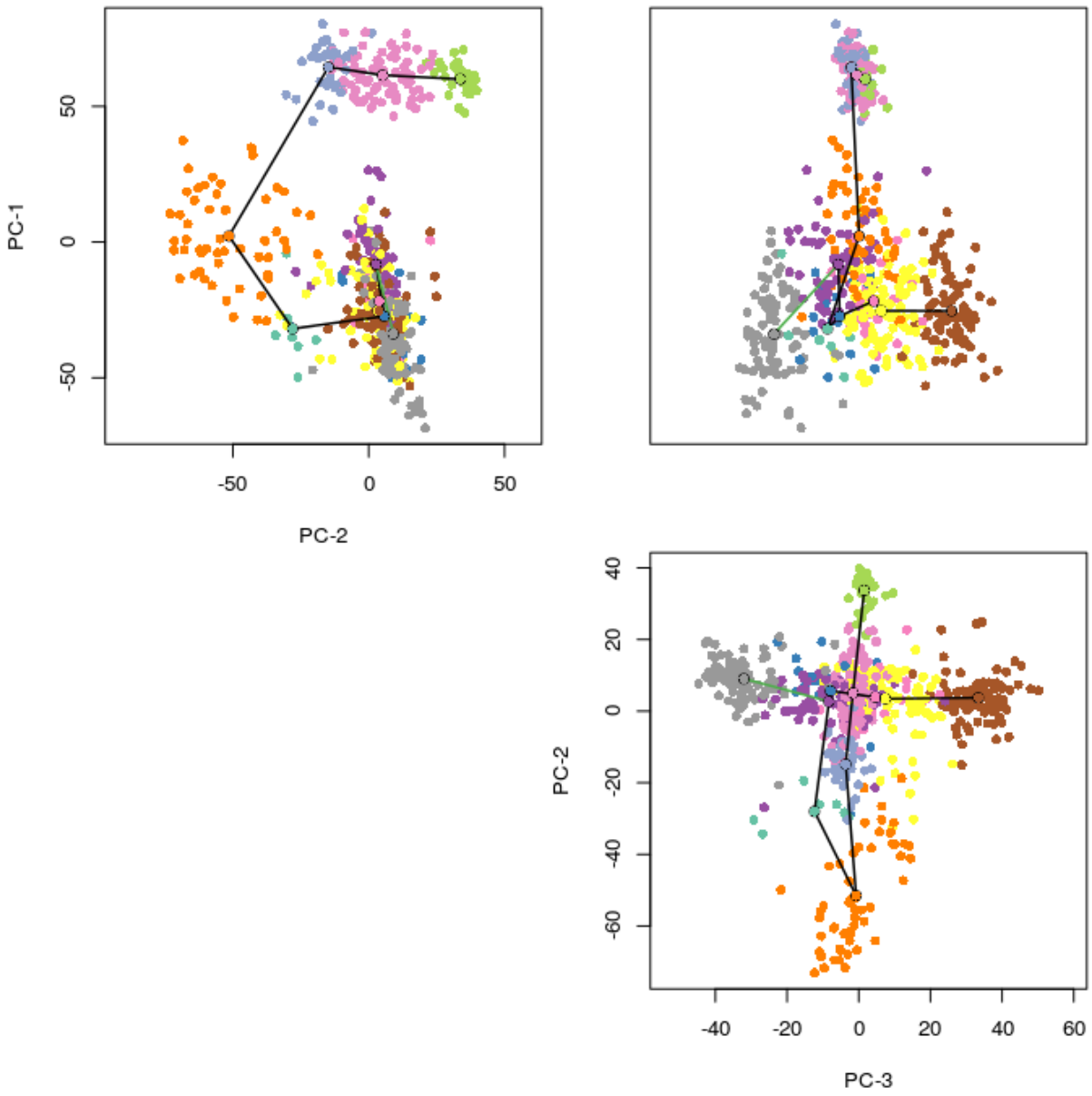


Running `get_lineages` with no supervision produces the connections shown above. Since no root cluster was specified, `slingshot` picked one of the leaf-node clusters to be the beginning, based on a simple parsimony rule. The root cluster is the leaf-node cluster connected by a green line.

```
l2 <- get_lineages(pcaX, clus, start.clus = 'm10')
```

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

```
# plot_tree(pcaX, clus, l2, threeD = TRUE)
plot_tree(pcaX, clus, l2, dim = 3)
```

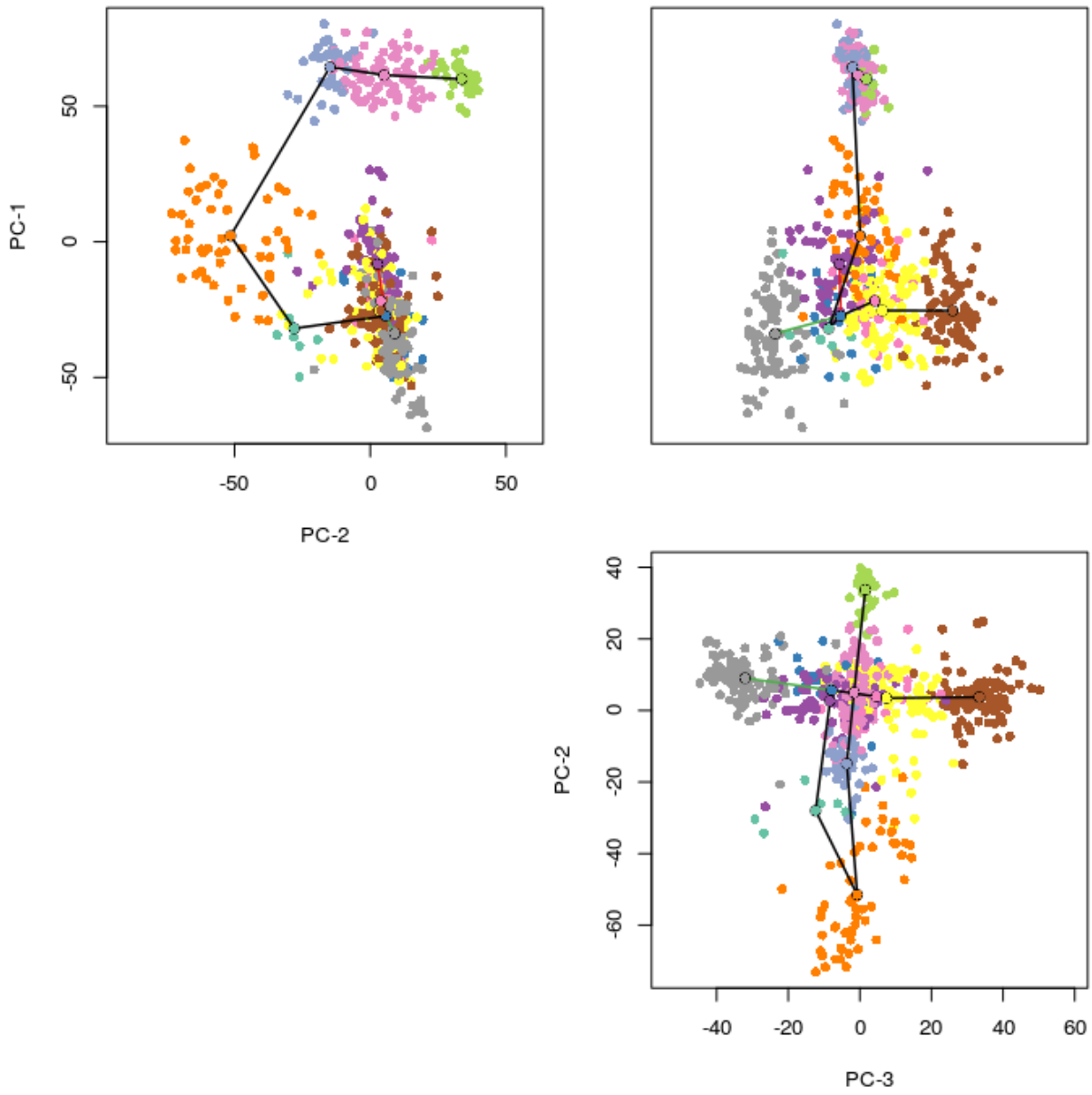


When we specify a root cluster we get the same connections and the only difference is which line is drawn in green.

```
l3 <- get_lineages(pcaX, clus, start.clus = 'm10', end.clus = 'm17')
```

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

```
# plot_tree(pcaX, clus, l3, threeD = TRUE)
plot_tree(pcaX, clus, l3, dim = 3)
```



Here we demonstrate the ability to specify end point clusters, which puts a constraing on the connections. We now draw the MST subject to the constraint that given end point clusters must be leaves. Pre-specified end point clusters are connected by red lines.

There are a few additional arguments we could have passed to `get_lineages` for more greater control:

- `dist.fun` is a function for computing distances between clusters. The default is squared distance between cluster centers normalized by their joint covariance matrix.
- `omega` is a granularity parameter, allowing the user to set an upper limit on connection distances. It takes values between 0 and 1 (or `Inf`), representing a percentage of the largest observed distance.
- `distout` is a logical value, indicating whether the user wants the pairwise cluster distance matrix to be returned with the output.

After constructing the MST, `get_lineages` identifies paths through the tree to designate as lineages. At this stage, a lineage will consist of an ordered set of cluster names, starting with the root cluster and ending with a leaf. The output of `get_lineages` is a list of these vectors, along with some additional information on how they were constructed.

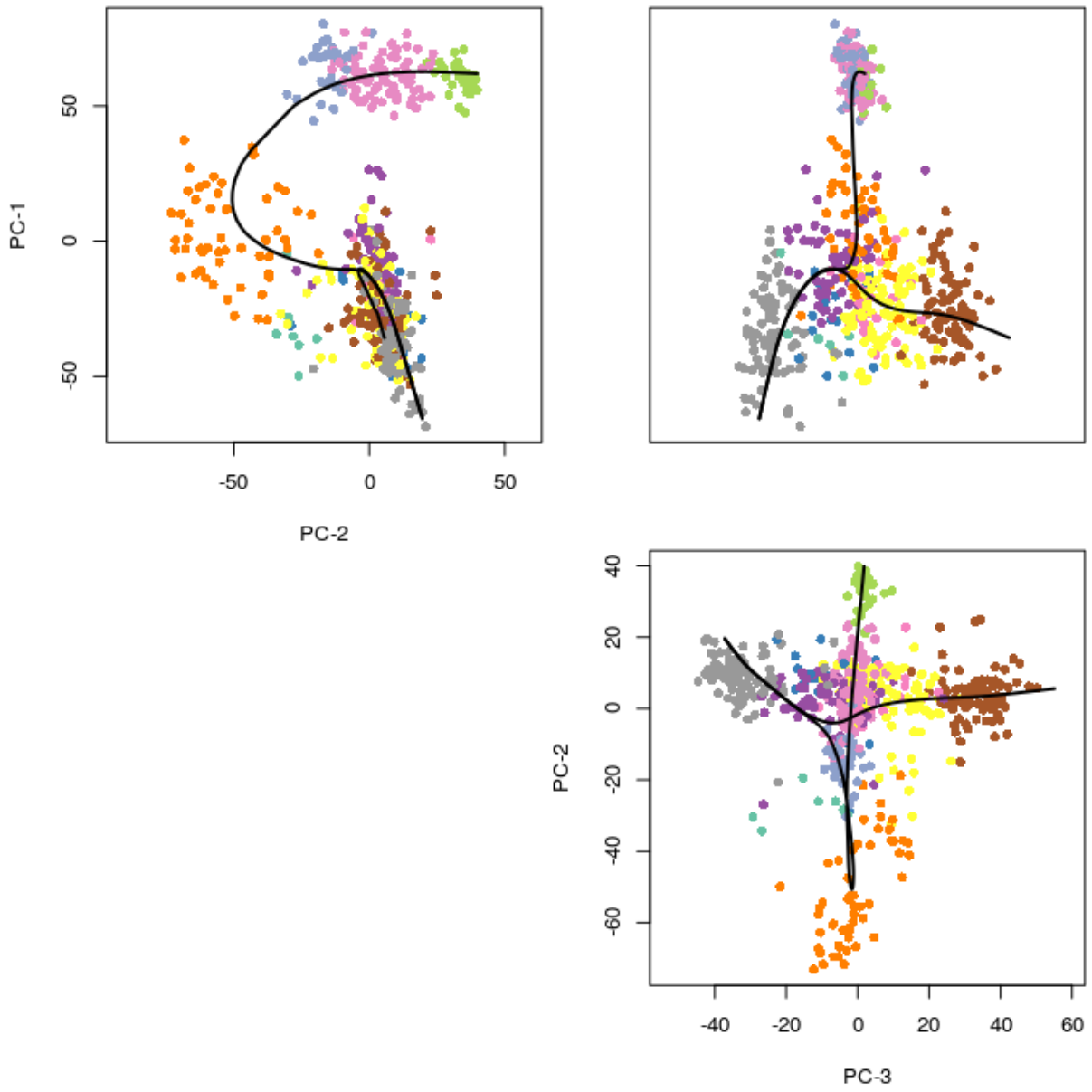
3 Step 2: Construct smooth lineages and order cells with `get_curves`

In order to model development along these various lineages, we will construct smooth curves with the function `get_curves`. Using smooth curves based on all the cells eliminates the problem of cells projecting onto vertices of piece-wise linear trajectories and makes `slingshot` more robust to noise in the clustering results.

In order to construct smooth lineages, `get_curves` follows an iterative process similar to that of principal curves presented in (Hastie and Stuetzle 1989). When there is only a single lineage, the resulting curve is simply the principal curve through the center of the data, with one adjustment: the initial curve is constructed with the linear connections between cluster centers rather than the first prinicpal component of the data. This adjustment adds stability and typically hastens the algorithm's convergence.

When there are two or more lineages, we add an additional step to the algorithm: averaging curves near shared cells. Both lineages should agree fairly well on cells that have yet to differentiate, so at each iteration we average the curves in the neighborhood of these cells. This increases the stability of the algorithm and produces smooth branching lineages.

```
crv <- get_curves(pcaX, clus, l2)
# plot_curves(pcaX, clus, c, threeD = TRUE)
plot_curves(pcaX, clus, crv, dim = 3)
```



The output of `get_curves` is a list with one element per curve. Each element is an object of the `principal.curve` class, with the following slots:

- `s`: the matrix of points that make up the curve. These correspond to the orthogonal projections of the data points, but oredered such the `lines(s)` will produce a smooth curve.
- `tag`: the indices of the original data points in `s`.
- `lambda`: arclengths of the points in `s` along the curve.
- `dist`: the total squared distance between data points and their projections onto the curve.
- `pseudotime`: the vector of pseudotime values along this lineage.

4 Step 3: Find temporally expressed genes

Typically, the next step will be to find genes that change their expression as a function of developmental time. This can be done using the full genes-by-samples data matrix, but we will use the subset consisting of the 1,000 most variable genes.

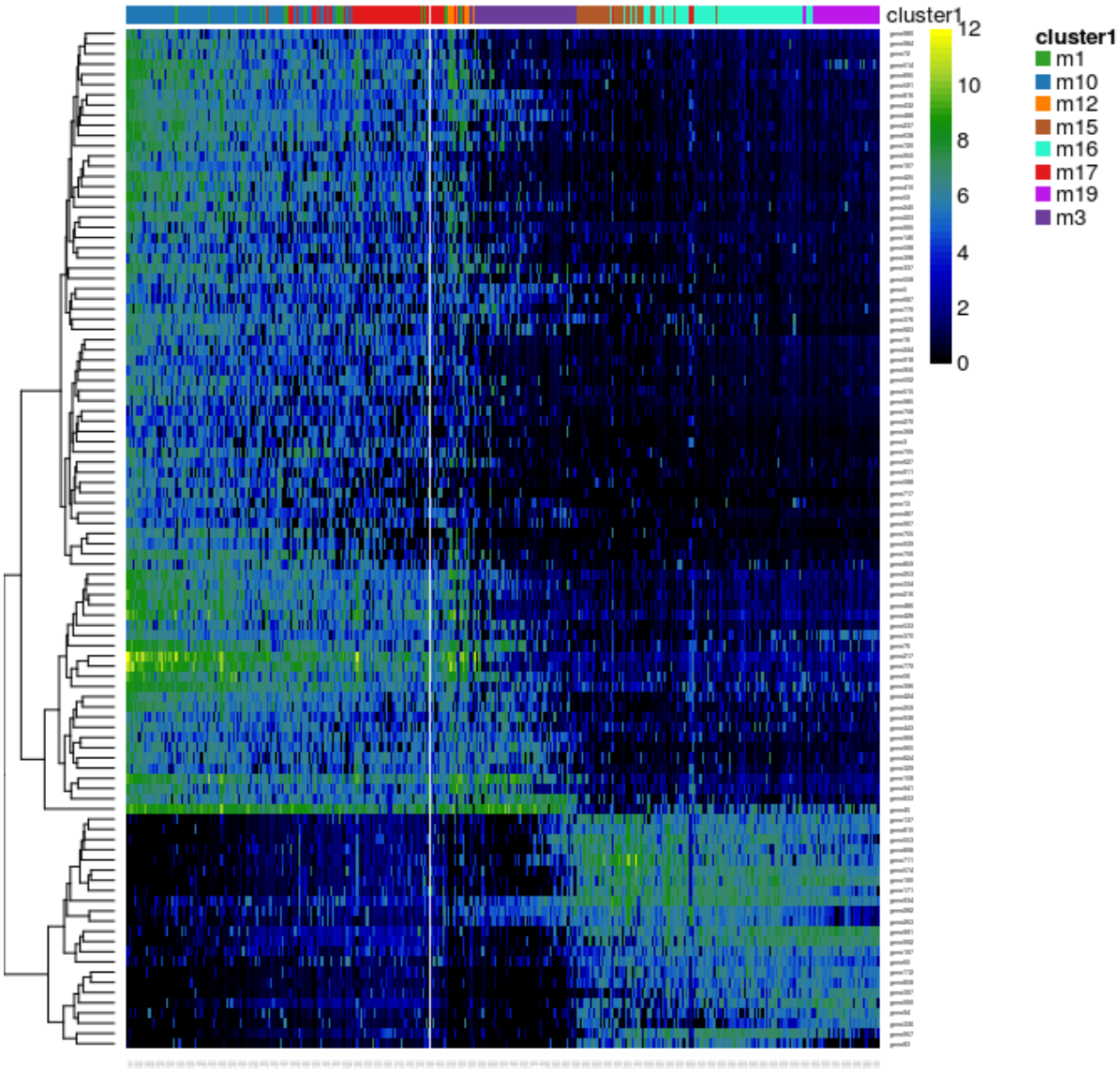
```
data('var_genes')
```

For a quick analysis, we will regress each gene on the two pseudotime vectors we have generated, using a general additive model (GAM). This allows us to detect non-linear patterns in gene expression over developmental time.

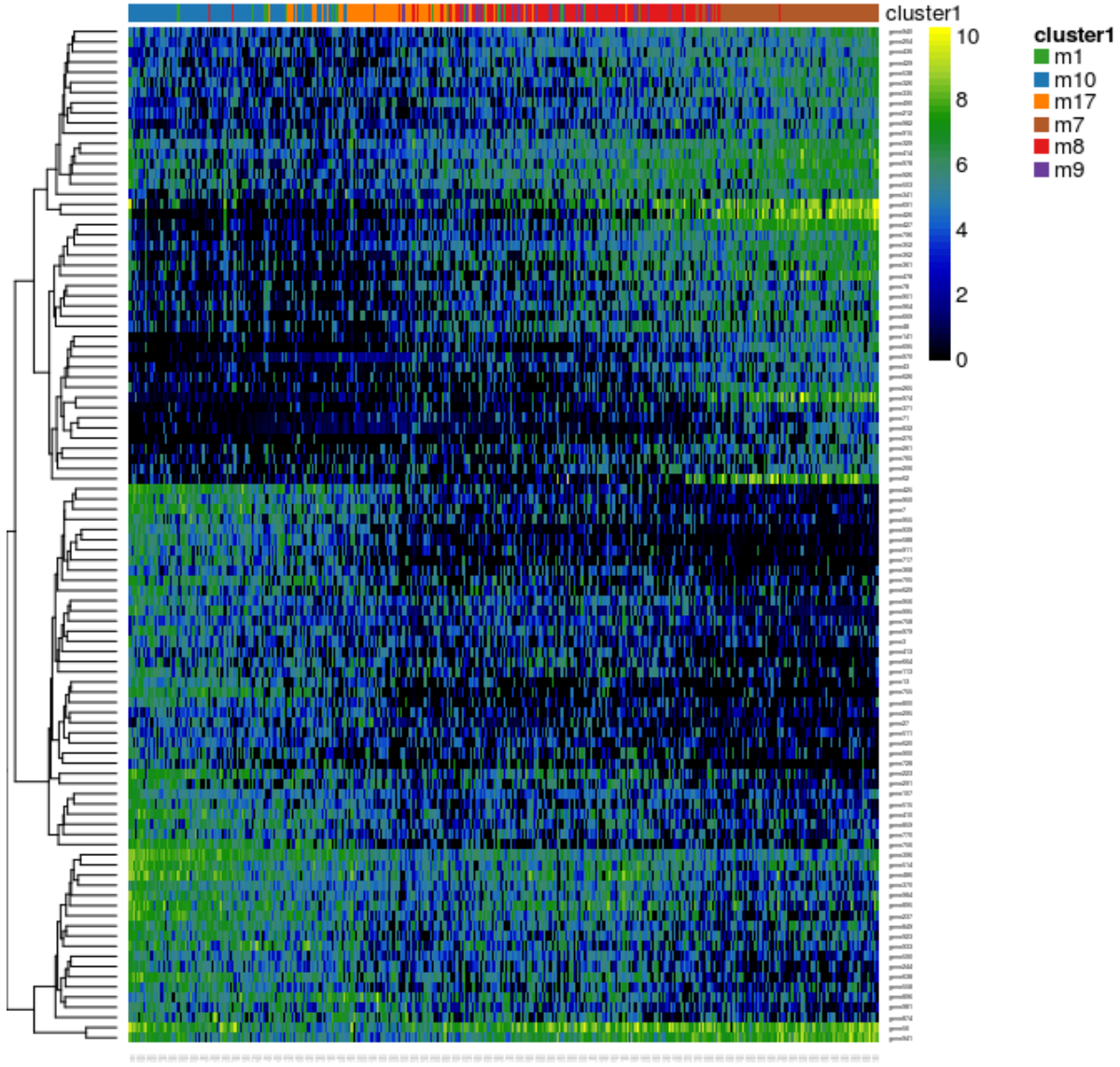
```
gam.pval <- vector("list",length(crv))
for(l in 1:length(crv)){
  t <- crv[[l]]$pseudotime
  y <- vargenes[,! is.na(t)]
  t <- t[! is.na(t)]
  gam.pval[[l]] <- apply(y,1,function(z){
    d <- data.frame(z=z, t=t)
    tmp <- gam(z ~ lo(t), data=d)
    p <- summary(tmp)[4][[1]][1,5]
    p
  })
}
```

We can then pick out the top genes for each lineage and visualize their expression over developmental time with a heatmap.

```
topgenes1 <- names(sort(gam.pval[[1]], decreasing = FALSE))[1:100]
heatdata1 <- vargenes[rownames(vargenes) %in% topgenes1, order(crv[[1]]$
heatclus1 <- clus[order(crv[[1]]$pseudotime, na.last = NA)]
ce1 <- clusterExperiment(heatdata1, heatclus1, transformation=identity)
plotHeatmap(ce1, clusterSamplesData="orderSamplesValue")
```




```
topgenes2 <- names(sort(gam.pval[[2]], decreasing = FALSE))[1:100]
heatdata2 <- vargenes[rownames(vargenes) %in% topgenes2, order(crv[[2]]$
heatclus2 <- clus[order(crv[[2]]$pseudotime, na.last = NA)]
ce2 <- clusterExperiment(heatdata2, heatclus2, transformation=identity)
plotHeatmap(ce2, clusterSamplesData="orderSamplesValue")
```



5 Session Info

```
sessionInfo()
```

```
## R version 3.3.0 (2016-05-03)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.4 LTS
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [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] splines   parallel  stats4    stats     graphics  grDevices util
##  [8] datasets  methods   base
##
## other attached packages:
##  [1] gam_1.12                       foreach_1.4.3
##  [3] slingshot_0.0.0.9005           RColorBrewer_1.1-2
##  [5] scone_0.0.5                    MAST_0.934
##  [7] reshape_0.8.5                  cluster_2.0.4
##  [9] clusterExperiment_0.99.1-9003 SummarizedExperiment_1.3.5
## [11] Biobase_2.33.0                 GenomicRanges_1.25.8
## [13] GenomeInfoDb_1.9.1             IRanges_2.7.11
## [15] S4Vectors_0.11.7               BiocGenerics_0.19.1
## [17] bioc2016singlecell_0.0.1       devtools_1.11.1
## [19] BiocStyle_2.1.10
##
## loaded via a namespace (and not attached):
##   [1] colorspace_1.2-6        hwriter_1.3.2
##   [3] class_7.3-14            modeltools_0.2-21
##   [5] mclust_5.2              XVector_0.13.2
##   [7] flexmix_2.3-13          AnnotationDbi_1.35.3
##   [9] mvtnorm_1.0-5           xml2_0.1.2
##  [11] codetools_0.2-14        R.methodsS3_1.7.1
##  [13] doParallel_1.0.10       bold_0.3.5
##  [15] DESeq_1.25.0            robustbase_0.92-6
##  [17] geneplotter_1.51.0      knitr_1.13
##  [19] ade4_1.7-4              jsonlite_0.9.22
##  [21] locfdr_1.1-8            Rsamtools_1.25.0
##  [23] phylobase_0.8.2         gridBase_0.4-7
##  [25] annotate_1.51.0         kernlab_0.9-24
##  [27] R.oo_1.20.0             rentrez_1.0.2
##  [29] httr_1.2.0              assertthat_0.1
##  [31] Matrix_1.2-6            lazyeval_0.2.0
##  [33] rotl_3.0.0              limma_3.29.10
```

```
##  [35] formatR_1.4             htmltools_0.3.5
##  [37] tools_3.3.0             igraph_1.0.1
##  [39] gtable_0.2.0            taxize_0.7.8
##  [41] reshape2_1.4.1          dplyr_0.4.3
##  [43] ShortRead_1.31.0        Rcpp_0.12.5
##  [45] NMF_0.20.6              trimcluster_0.1-2
##  [47] Biostrings_2.41.4       gdata_2.17.0
##  [49] ape_3.5                 nlme_3.1-128
##  [51] rtracklayer_1.33.7      iterators_1.0.8
##  [53] fpc_2.1-10              stringr_1.0.0
##  [55] rngtools_1.2.4          gtools_3.5.0
##  [57] XML_3.98-1.4            dendextend_1.2.0
##  [59] princurve_1.1-12        edgeR_3.15.0
##  [61] DEoptimR_1.0-4          zlibbioc_1.19.0
##  [63] MASS_7.3-45             scales_0.4.0
##  [65] aroma.light_3.3.0       yaml_2.1.13
##  [67] RUVSeq_1.7.2            memoise_1.0.0
##  [69] ggplot2_2.1.0           pkgmaker_0.22
##  [71] segmented_0.5-1.4       biomaRt_2.29.2
##  [73] latticeExtra_0.6-28     stringi_1.1.1
##  [75] RSQLite_1.0.0           genefilter_1.55.2
##  [77] GenomicFeatures_1.25.14 caTools_1.17.1
##  [79] boot_1.3-18             BiocParallel_1.7.4
##  [81] chron_2.3-47            prabclus_2.2-6
##  [83] matrixStats_0.50.2      bitops_1.0-6
##  [85] rncl_0.6.0              evaluate_0.9
##  [87] lattice_0.20-33         GenomicAlignments_1.9.4
##  [89] rredlist_0.1.0          plyr_1.8.4
##  [91] magrittr_1.5            R6_2.1.2
##  [93] gplots_3.0.1            DBI_0.4-1
##  [95] whisker_0.3-2           withr_1.0.2
##  [97] mixtools_1.0.4          RCurl_1.95-4.8
##  [99] survival_2.39-4         abind_1.4-3
## [101] nnet_7.3-12             tibble_1.0
## [103] EDASeq_2.7.2            uuid_0.1-2
## [105] KernSmooth_2.23-15      howmany_0.3-1
## [107] rmarkdown_0.9.6         RNeXML_2.0.6
## [109] grid_3.3.0              data.table_1.9.6
## [111] digest_0.6.9            diptest_0.75-7
## [113] xtable_1.8-2            tidyr_0.5.1
## [115] R.utils_2.3.0           munsell_0.4.3
## [117] registry_0.3
```

References

Hastie, Trevor, and Werner Stuetzle. 1989. “Principal Curves.” *Journal of the American Statistical Association* 84 (406): 502–16.