Monocle: Differential expression and time-series analysis for single-cell RNA-Seq and qPCR experiments

Cole Trapnell

University of Washington,
Seattle, Washington, USA
coletrap@uw.edu

Davide Cacchiarelli

Harvard University,
Cambridge, Massachussetts, USA
davide@broadinstitute.org

December 8, 2015

Abstract

Single cell gene expression studies enable profiling of transcriptional regulation during complex biological processes and within highly hetergeneous cell populations. These studies allow discovery of genes that identify certain subtypes of cells, or that mark a particular intermediate states during a biological process. In many single cell studies, individual cells are executing through a gene expression program in an unsynchronized manner. In effect, each cell is a snapshot of the transcriptional program under study. The package *monocle* provides tools for analyzing single-cell expression experiments. It performs differential gene expression and clustering to identify important genes and cell states. It is designed for RNA-Seq studies, but can be used with qPCR or other targeted assays. For more information on the algorithm at the core of *monocle*, or to learn more about how to use single cell RNA-Seq to study a complex biological process, see Trapnell and Cacchiarelli *et al* [1]

Contents

1 BEAM 2

2 Introduction 12

3 Single-cell expression data in Monocle 12
3.1 The CellDataSet class 12

4 Quality control of single cell RNA-Seq experiments 13

5 Identifying cells of interest 16

6 Basic differential expression analysis 17

7 Ordering cells by progress in "pseudotime" 18

8 Advanced differential expression analysis 30
8.1 Finding genes that distinguish cell type or state 30
8.2 Finding genes that change as a function of pseudotime 31
8.3 Multi-factorial differential expression analysis 32

9 Clustering genes by pseudotemporal expression pattern 33

10 Citation 34

11 Acknowledgements 34

12 Session Info 35

1 BEAM

```
baseLoc <- system.file(package="monocle")
#baseLoc <- './inst'
extPath <- file.path(baseLoc, "extdata")
load(file.path(extPath, "lung_phenotype_data.RData"))
load(file.path(extPath, "lung_exprs_data.RData"))
load(file.path(extPath, "lung_feature_data.RData"))
lung_exprs_data <- lung_exprs_data[,row.names(lung_phenotype_data)]

pd <- new("AnnotatedDataFrame", data = lung_phenotype_data)
fd <- new("AnnotatedDataFrame", data = lung_feature_data)

# Now, make a new CellDataSet using the RNA counts
lung <- newCellDataSet(as(lung_exprs_data, "sparseMatrix"),
                       phenoData = pd,
                       featureData = fd,
                       lowerDetectionLimit=1,
                       expressionFamily=negbinomial())

lung <- estimateSizeFactors(lung)
lung <- estimateDispersions(lung)

pData(lung)$Total_mRNAs <- colSums(exprs(lung))
lung <- detectGenes(lung, min_expr = 1)
expressed_genes <- row.names(subset(fData(lung), num_cells_expressed >= 5))
ordering_genes <- expressed_genes
lung <- setOrderingFilter(lung, ordering_genes)
lung <- reduceDimension(lung, use_vst = F, pseudo_expr = 1)

## setting up adjacency matrix
## **************************************
## Iteration: 0
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 1
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 2
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 3
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 4
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 5
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 6
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 7
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ****************************************
## Iteration: 8
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ****************************************
## Iteration: 9
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ***************************************
## Iteration: 10
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ***************************************
## Iteration: 11
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 12
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 13
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 14
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## *************************************
## Iteration: 15
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## *************************************
## Iteration: 16
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ************************************
## Iteration: 17
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ************************************
## Iteration: 18
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 19
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (202 x 185)
##    W : (202 x 2)
##    Z : (2 x 185)
##    L : (185 x 185)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## Clearing MST sparse matrix
## Setting up MST sparse matrix with 184

## <sparse>[ <logic> ] :  .M.sub.i.logical() maybe inefficient

lung <- orderCells(lung, num_paths=2, reverse=T)

## [1] "the cells on the end of MST: "
## logical(0)

## Error in assign_cell_state_helper(res, res$root):  object 'curr_state' not found

plot_spanning_tree(lung, color_by="Pseudotime")

## Error in `colnames<-`(`*tmp*`, value = "sample_name"):  'names' attribute [1] must be the same
length as the vector [0]
```

```
# BEAM_res <- BEAM(lung, cores = detectCores())
#
# head(BEAM_res)
```

```
#
# plot_genes_branched_heatmap(lung[row.names(subset(BEAM_res, qval < 0.05)),], num_clusters = 4,
#       cores = detectCores(), use_gene_short_name = T, show_rownames = T)
```

2 Introduction

The *monocle* package provides a toolkit for analyzing single cell gene expression experiments. It was developed to analyze single cell RNA-seq data, but can also be used with qPCR measurements. This vignette provides an overview of a single cell RNA-Seq analysis workflow with Monocle. Monocle was developed to analyze dynamic biological processes such as cell differentiation, although it also supports simpler experimental settings.

As cells differentiate, they undergo a process of transcriptional re-configuration, with some genes being silenced and others newly activated. While many studies have compared cells at different stages of differentiation, examining intermediate states has proven difficult, for two reasons. First, it is often not clear from cellular morphology or established markers what intermediate states exist between, for example, a precursor cell type and its terminally differentiated progeny. Moreover, two cells might transit through a different sequence of intermediate stages and ultimately converge on the same end state. Second, even cells in a genetically and epigenetically clonal population might progress through differentiation at different rates *in vitro*, depending on positioning and level of contacts with neighboring cells. Looking at average behavior in a group of cells is thus not necessarily faithful to the process through which an individual cell transits.

Monocle computationally reconstructs the transcriptional transitions undergone by differentiating cells. It orders a mixed, unsynchronized population of cells according to progress through the learned process of differentiation. Because the population may actually differentiate into multiple separate lineages, Monocle allows the process to branch, and can assign each cell to the correct sub-lineage. It subsequently identifies genes which distinguish different states, and genes that are differentially regulated through time. Finally, it performs clustering on all genes, to classify them according to kinetic trends. The algorithm is inspired by and and extends one proposed by Magwene et al to time-order microarray samples [2]. Monocle differs from previous work in three ways. First, single-cell RNA-Seq data differ from microarray measurements in many ways, and so Monocle must take special care to model them appropriately at several steps in the algorithm. Secondly, the earlier algorithm assumes that samples progress along a single trajectory through expression space. However, during cell differentiation, multiple lineages might arise from a single progenitor. Monocle can find these lineage branches and correctly place cells upon them. Finally, Monocle also performs differential expression analysis and clustering on the ordered cells to help a user identify key events in the biological process of interest.

3 Single-cell expression data in Monocle

The *monocle* package takes a matrix of expression values, which are typically for genes (as opposed to splice variants), as calculated by Cufflinks [3] or another gene expression estimation program. Monocle assumes that gene expression values are log-normally distributed, as is typical in RNA-Seq experiments. Monocle does not normalize these expression values to control for library size, depth of sequencing, or other sources of technical variability - whichever program that you use to calculate expression values should do that. Monocle is *not* meant to be used with raw counts, and doing so could produce nonsense results.

3.1 The CellDataSet class

monocle holds single cell expression data in objects of the *CellDataSet* class. The class is derived from the Bioconductor *ExpressionSet* class, which provides a common interface familiar to those who have analyzed microarray experiments with Bioconductor. The class requires three input files:

1. `exprs`, a numeric matrix of expression values, where rows are genes, and columns are cells
2. `phenoData`, an *AnnotatedDataFrame* object, where rows are cells, and columns are cell attributes (such as cell type, culture condition, day captured, etc.)
3. `featureData`, an *AnnotatedDataFrame* object, where rows are features (e.g. genes), and columns are gene attributes, such as biotype, gc content, etc.

The expression value matrix *must* have the same number of columns as the `phenoData` has rows, and it must have the same number of rows as the `featureData` data frame has rows. Row names of the `phenoData` object should match the column names of the expression matrix. Row names of the `featureData` object should match row names of the expression matrix.

You can create a new *CellDataSet* object as follows:

```
#not run
HSMM_expr_matrix <- read.table("fpkm_matrix.txt")
HSMM_sample_sheet <- read.delim("cell_sample_sheet.txt")
HSMM_gene_annotation <- read.delim("gene_annotations.txt")
```

Once these tables are loaded, you can create the CellDataSet object like this:

```
pd <- new("AnnotatedDataFrame", data = HSMM_sample_sheet)
fd <- new("AnnotatedDataFrame", data = HSMM_gene_annotation)
HSMM <- newCellDataSet(as.matrix(HSMM_expr_matrix), phenoData = pd, featureData = fd)
```

This will create a CellDataSet object with expression values measured in FPKM, a measure of relative expression reported by Cufflinks. However, if you performed your single-cell RNA-Seq experiment using *spike-in* standards, you can convert these measurements into mRNAs per cell (RPC). RPC values are often easier to analyze than FPKM values, because have better statistical tools to model them. In fact, it's possible to convert FPKM values to RPC values even if there were no spike-in standards included in the experiment. You can convert to RPC values before creating your CellDataSet object using the `relative2abs` function, as follows:

```
pd <- new("AnnotatedDataFrame", data = HSMM_sample_sheet)
fd <- new("AnnotatedDataFrame", data = HSMM_gene_annotation)

# First create a CellDataSet from the relative expression levels
HSMM <- newCellDataSet(HSMM_expr_matrix,
                       phenoData = pd,
                       featureData = fd)

# Next, use it to estimate RNA counts
rpc_matrix <- relative2abs(HSMM, cores = detectCores())

# Now, make a new CellDataSet using the RNA counts
HSMM <- newCellDataSet(rpc_matrix,
                       phenoData = pd,
                       featureData = fd,
                       lowerDetectionLimit=1,
                       expressionFamily=negbinomial())
```

Note that since we are using RPC values, we have changed the value of `lowerDetectionLimit` to reflect the new scale of expression. Importantly, we have also set the `expressionFamily` to `negbinomial()`, which tells Monocle to use the negative binomial distribution in certain downstream statistical tests. Failing to change these two options can create problems later on, so make sure not to forget them when using RPC values.

Finally, we'll also call two functions that pre-calculate some information about the data. Size factors help us normalize for differences in mRNA recovered across cells, and "dispersion" values will help us perform differential expression analysis later.

```
HSMM <- estimateSizeFactors(HSMM)
HSMM <- estimateDispersions(HSMM)
```

We're now ready to start using the `HSMM` object in our analysis.

4 Quality control of single cell RNA-Seq experiments

The first step in any single-cell RNA-Seq analysis is identifying poor-quality libraries from further analysis. Most single-cell workflows will include at least some libraries made from dead cells or empty wells in a plate. It's also crucial to remove doublets: libraries that were made from two or more cells accidentally. These cells can disrupt downstream steps like pseudotime ordering or clustering. This section walks through typical quality control steps that should be performed as part of all analyses with Monocle.

It is often convenient to know how many express a particular gene, or how many genes are expressed by a given cell. Monocle provides a simple function to compute those statistics:

```
HSMM <- detectGenes(HSMM, min_expr = 0.1)
print(head(fData(HSMM)))

##                    gene_short_name        biotype num_cells_expressed
## ENSG00000000003.10          TSPAN6 protein_coding                 167
## ENSG00000000005.5             TNMD protein_coding                   0
## ENSG00000000419.8             DPM1 protein_coding                 189
## ENSG00000000457.8            SCYL3 protein_coding                  15
## ENSG00000000460.12        C1orf112 protein_coding                  34
## ENSG00000000938.8              FGR protein_coding                   0
##                    use_for_ordering
## ENSG00000000003.10            FALSE
## ENSG00000000005.5             FALSE
## ENSG00000000419.8             FALSE
## ENSG00000000457.8             FALSE
## ENSG00000000460.12             TRUE
## ENSG00000000938.8             FALSE

expressed_genes <- row.names(subset(fData(HSMM), num_cells_expressed >= 50))
```

The vector `expressed_genes` now holds the identifiers for genes expressed in at least 50 cells of the data set. We will use this list later when we put the cells in order of biological progress. It is also sometimes convenient to exclude genes expressed in few if any cells from the *CellDataSet* object so as not to waste CPU time analyzing them for differential expression.

Let's start trying to remove unwanted, poor quality libraries from the CellDataSet. Your single cell RNA-Seq protocol may have given you the opportunity to image individual cells after capture but prior to lysis. This image data allows you to score your cells, confirming that none of your libraries were made from empty wells or wells with excess cell debris. With some protocols and instruments, you may get more than one cell captured instead just a single cell. You should exclude libraries that you believe did not come from a single cell, if possible. Empty well or debris well libraries can be especially problematic for Monocle. It's also a good idea to check that each cell's RNA-seq library was sequenced to an acceptible degree. While there is no widely accepted minimum level for what constitutes seequencing "deeply enough", use your judgement: a cell sequenced with only a few thousand reads is unlikely to yield meaningful measurements.

CellDataSet objects provide a convenient place to store per-cell scoring data: the `phenoData` slot. Simply include scoring attributes as columns in the data frome you used to create your *CellDataSet* container. You can then easily filter out cells that don't pass quality control. You might also filter cells based on metrics from high throughput sequencing quality assessment packages such as FastQC. Such tools can often identify RNA-Seq libraries made from heavily degraded RNA, or where the library contains an abnormally large amount of ribosomal, mitochondrial, or other RNA type that you might not be interested in.

The HSMM dataset included with this package has scoring columns built in:

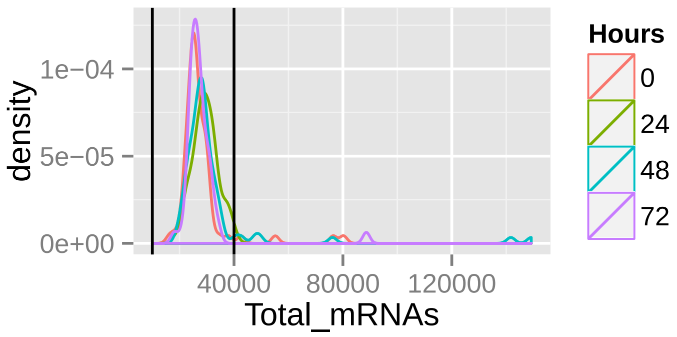
```
print(head(pData(HSMM)))

##                 Library Well Hours Media Mapped.Fragments Pseudotime State
## T0_CT_A01 SCC10013_A01  A01     0    GM          1958074  23.916673     1
## T0_CT_A03 SCC10013_A03  A03     0    GM          1930722   9.022265     1
## T0_CT_A05 SCC10013_A05  A05     0    GM          1452623   7.546608     1
## T0_CT_A06 SCC10013_A06  A06     0    GM          2566325  21.463948     1
## T0_CT_A07 SCC10013_A07  A07     0    GM          2383438  11.299806     1
## T0_CT_A08 SCC10013_A08  A08     0    GM          1472238  67.436042     2
##           Size_Factor num_genes_expressed
## T0_CT_A01   2.8512907                9957
## T0_CT_A03   1.3362363                7250
## T0_CT_A05   1.0439424                6049
## T0_CT_A06   0.8975817                5065
## T0_CT_A07   0.8570724                4494
## T0_CT_A08   1.0212473                5359
```

This dataset has already been filtered using the following commands:

```
valid_cells <- row.names(subset(pData(HSMM), Cells.in.Well == 1 & Control == FALSE & Clump == FALSE & De
HSMM <- HSMM[,valid_cells]
```

If you are using RPC values to measure expresion, as we are in this vignette, it's also good to look at the distribution of mRNA totals across the cells:

```
pData(HSMM)$Total_mRNAs <- colSums(exprs(HSMM))

qplot(Total_mRNAs, data=pData(HSMM), color=Hours, geom="density") +
  geom_vline(xintercept=10000) +
  geom_vline(xintercept=40000)
```

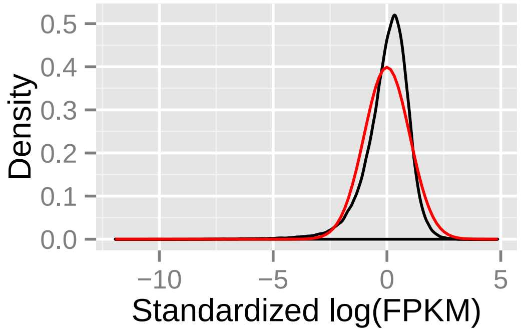


```
HSMM <- HSMM[,row.names(subset(pData(HSMM), Total_mRNAs >= 10000 & Total_mRNAs <= 40000))]
HSMM <- detectGenes(HSMM, min_expr = 0.1)
```

We've gone ahead and removed the few cells with either very low mRNA recovery or far more mRNA that the typical cell. Often, doublets or triplets have roughly twice the mRNA recovered as true single cells, so the latter filter is another means of excluding all but single cells from the analysis. Such filtering is handy if your protocol doesn't allow directly visualization of cell after they've been captured.

Once you've excluded cells that do not pass your quality control filters, you should verify that the expression values stored in your *CellDataSet* follow a distribution that is roughly lognormal:

```
# Log-transform each value in the expression matrix.
L <- log(exprs(HSMM[expressed_genes,]))

# Standardize each gene, so that they are all on the same scale,
# Then melt the data with plyr so we can plot it easily"
melted_dens_df <- melt(t(scale(t(L))))

# Plot the distribution of the standardized gene expression values.
qplot(value, geom="density", data=melted_dens_df) +  stat_function(fun = dnorm, size=0.5, color='red') +
xlab("Standardized log(FPKM)") +
ylab("Density")

## Warning:  Removed 1804710 rows containing non-finite values (stat_density).
```



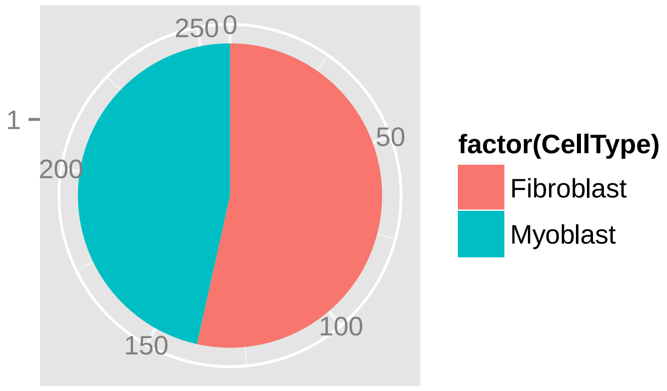
5 Identifying cells of interest

Single cell experiments are often performed on complex mixtures of multiple cell types. Dissociated tissue samples might contain two, three, or even many different cells types. In such cases, it's often nice to classify cells based on type using known markers. In the myoblast experiment, the culture contains fibroblasts that came from the original muscle biopsy used to establish the primary cell culture. Myoblasts express some key genes that fibroblasts don't. Selecting only the genes that express, for example, sufficiently high levels of *MYF5* or *MEF2C* excludes the fibroblasts. In the block of code below we first grab the values of *MYF5* and *MEF2C* for each cell, and then set these as columns in the pData table for the HSMM object. We don't have to do this, but it makes filtering a little more convenient. Next we create a vector that has a boolean flag for each cell: TRUE for a myoblast, false for a fibroblast. We then use that vector to creat a third column in the that specicies each cell's type. Finally, we subset the CellDataSet object to create $HSMM_myo$, which includes only myoblasts.

```
MEF2C_expr <- exprs(HSMM[row.names(subset(fData(HSMM), gene_short_name == "MEF2C")),])
MYF5_expr <- exprs(HSMM[row.names(subset(fData(HSMM), gene_short_name == "MYF5")),])

pData(HSMM)$MEF2C <- as.vector(MEF2C_expr)
pData(HSMM)$MYF5 <- as.vector(MYF5_expr)

MEF2C_thresh <- 5 #0.1
MYF5_thresh <- 1 #0.1

#cell_is_hsmm <- (pData(HSMM)£MEF2C > MEF2C_thresh | pData(HSMM)£MYF5 > MYF5_thresh)
cell_is_hsmm <- (pData(HSMM)$MEF2C > MEF2C_thresh | pData(HSMM)$MYF5 > MYF5_thresh)

names(cell_is_hsmm) <- row.names(pData(HSMM))
pData(HSMM)$CellType <- "Myoblast"
pData(HSMM)$CellType[cell_is_hsmm == FALSE] <-  "Fibroblast"

HSMM_myo <- HSMM[,pData(HSMM)$CellType == "Myoblast"]
```

Now that we have tagged the cells, it's easy to quantify the level of fibroblast "contaminantation" in the HSMM culture:

```
pie <- ggplot(pData(HSMM), aes(x = factor(1), fill = factor(CellType))) +
 geom_bar(width = 1)
pie + coord_polar(theta = "y") +
  theme(axis.title.x=element_blank(), axis.title.y=element_blank())
```



6 Basic differential expression analysis

Differential gene expression analysis is a common task in RNA-Seq experiments. Monocle can help you find genes that are differentially expressed between groups of cells and assesses the statistical signficance of those changes. These comparisons require that you have a way to collect your cells into two or more groups. These groups are defined by columns in the `phenoData` table of each `CellDataSet`. Monocle will assess the signficance of each gene's expression level across the different groups of cells.

Performing differential expression analysis on all genes in the human genome can take a substantial amount of time. For a dataset as large as the myoblast data from [1], which contains several hundred cells, the analysis can take several hours on a single CPU. Let's select a small set of genes that we know are important in myogenesis to demonstrate Monocle's capabilities:

```r
marker_genes <- row.names(subset(fData(HSMM_myo),
                                 gene_short_name %in% c("MEF2C", "MEF2D", "MYF5",
                                                        "ANPEP", "PDGFRA","MYOG",
                                                        "TPM1",  "TPM2",  "MYH2",
                                                        "MYH3",  "NCAM1", "TNNT1",
                                                        "TNNT2", "TNNC1", "CDK1",
                                                        "CDK2",  "CCNB1", "CCNB2",
                                                        "CCND1", "CCNA1", "ID1")))
```

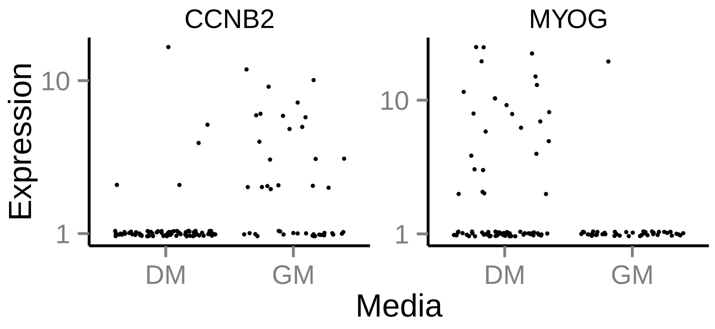
In the myoblast data, the cells collected at the outset of the experiment were cultured in "growth medium" (GM) to prevent them from differentiating. After they were harvested, the rest of the cells were switched over to "differentiation medium" (DM) to promote differentiation. Let's have monocle find which of the genes above are affected by this switch:

```r
diff_test_res <- differentialGeneTest(HSMM_myo[marker_genes,],
                                      fullModelFormulaStr="~Media")

# Select genes that are significant at an FDR < 10%
sig_genes <- subset(diff_test_res, qval < 0.1)

sig_genes[,c("gene_short_name", "pval", "qval")]
```

```
##                      gene_short_name         pval         qval
## ENSG00000081189.9              MEF2C 1.488239e-08 7.813256e-08
## ENSG00000105048.12             TNNT1 5.191657e-05 1.211387e-04
## ENSG00000109063.9               MYH3 8.213617e-06 2.156075e-05
## ENSG00000111049.3               MYF5 7.182938e-10 5.028057e-09
## ENSG00000114854.3              TNNC1 2.762425e-06 9.668487e-06
## ENSG00000118194.14             TNNT2 1.269131e-16 2.665175e-15
## ENSG00000122180.4               MYOG 4.191264e-03 6.286896e-03
## ENSG00000123374.6               CDK2 3.233138e-03 5.222761e-03
## ENSG00000125414.13              MYH2 1.193437e-03 2.088514e-03
## ENSG00000125968.7                ID1 7.132934e-08 2.995832e-07
## ENSG00000134057.10             CCNB1 2.810908e-02 3.689316e-02
## ENSG00000140416.15              TPM1 3.923939e-06 1.177182e-05
## ENSG00000149294.12             NCAM1 1.157988e-15 1.215888e-14
## ENSG00000157456.3              CCNB2 8.315063e-04 1.587421e-03
## ENSG00000166825.9              ANPEP 6.102415e-03 8.543381e-03
## ENSG00000198467.8               TPM2 5.289995e-04 1.110899e-03
```

So 16 of the 22 genes are significant at a 10% false discovery rate! This isn't surprising, as most of the above genes are highly relevant in myogenesis. Monocle also provides some easy ways to plot the expression of a small set of genes grouped by the factors you use during differential analysis. This helps you visualize the differences revealed by the tests above. One type of plot is a "jitter" plot.

```
MYOG_ID1 <- HSMM_myo[row.names(subset(fData(HSMM_myo),
                                  gene_short_name %in% c("MYOG", "CCNB2"))),]
plot_genes_jitter(MYOG_ID1, grouping="Media", ncol=2)
```



Note that we can control how to layout the genes in the plot by specifying the number of rows and columns. See the man page on `plot_genes_jitter` for more details on controlling its layout. Most if not all of Monocle's plotting routines return a plot object from the *ggplot2*. This package uses a grammar of graphics to control various aspects of a plot, and makes it easy to customize how your data is presented. See the *ggplot2* book [**?**] for more details.

7 Ordering cells by progress in "pseudotime"

In many biological processes, cells do not progress in perfect synchrony. In single-cell expression studies of processes such as cell differentiation, captured cells might be widely distributed in terms of progress. That is, in a population of cells captured at exactly the same time, some cells might be far along, while others might not yet even have begun the process. Monocle can informatically put the cells "in order" of how far they have progressed through the process you're studying. Monocle may even be able to find where cells diverge, with groups of cells proceeding down distinct paths. In this section, we will put a set of differentiating myoblasts in order of progress through myogenesis.

First, we must decide which genes we will use to define a cell's progress through myogenesis. Monocle orders cells by examining the pattern of expression of these genes across the cell population. Monocle looks for genes that vary in "interesting" ways (that is aren't just noisy), and uses these to structure the data. We ultimately want a set of genes that increase (or decrease) in expression as a function of progress through the process we're studying.

Ideally, we'd like to use as little prior knowledge of the biology of the system under study as possible. We'd like to discover the important ordering genes from the data, rather than relying on literature and textbooks, because that might introduce bias in the ordering. One effective way to isolate a set of ordering genes is to simply compare the cells collected at the beginning of the process to those at the end and find the differentially expressed genes, as described above. The command below will find all genes that are differentially expressed in response to the switch from growth medium to differentiation medium:

```
#not run
diff_test_res <- differentialGeneTest(HSMM_myo[expressed_genes,], fullModelFormulaStr="~Media")
ordering_genes <- row.names (subset(diff_test_res, qval < 0.01))
```

We could also leverage the developmental biology community's extensive knowledge of expression dynamics during skeletal myogenesis, and use the small set of genes discussed above.

```
#not run
ordering_genes <- intersect (marker_genes, expressed_genes)
```

In most cases, however, we want to use all genes that are expressed above a reasonable threshold. Including genes that are expressed at extremely low levels in only a few cells just injects noise into the ordering analysis, so we'll exclude them. Monocle provides a function to collect the list of genes that are expressed above or below given thresholds. The function `selectGenesInExpressionRange` allows you to select genes that have, for example, a median expression level across cells of at least 2 RPC. This function is quite flexible: see the man page for ?selectGenesInExpressionRange for more details

```
ordering_genes <- selectGenesInExpressionRange(HSMM_myo,
                                               min_expression_threshold=2,
                                               max_expression_threshold=Inf,
                                               detectionLimit=0.1,
                                               stat_fun=function(x) { median(round(x)) })
```

Once we have a list of gene ids to be used for ordering, we need to set them in the `HSMM` object, because the next several functions will depend on them.

```
HSMM_myo <- setOrderingFilter(HSMM_myo, ordering_genes)
```

The genes we've chosen to use for ordering define the *state space* of the cells in our data set. Each cell is a point in this space, which has dimensionality equal to the number of genes we've chosen. So if there are 500 genes used for ordering, each cell is a point in a 500-dimensional space. For a number of reasons, Monocle works better if we can *reduce* the dimensionality of that space before we try to put the cells in order. In this case, we will reduce the space down to one with two dimensions, which we will be able to easily visualize and interpret while Monocle is ordering the cells.

```
HSMM_myo <- reduceDimension(HSMM_myo)

## setting up adjacency matrix
## **************************************
## Iteration: 0
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 1
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 2
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 3
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 4
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 5
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 6
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 7
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ************************************
## Iteration: 8
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ************************************
## Iteration: 9
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 10
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 11
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ************************************
## Iteration: 12
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ************************************
## Iteration: 13
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 14
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## **************************************
## Iteration: 15
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ***************************************
## Iteration: 16
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ***************************************
## Iteration: 17
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## ***************************************
## Iteration: 18
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## *************************************
## Iteration: 19
## updating weights in graph
## Finding MST
## Refreshing B matrix
## Computing distZY
## Computing min_dist
## Computing tmpR
## Computing R
## Computing Gamma
## Computing obj1
## Computing obj2
##    X : (1579 x 120)
##    W : (1579 x 2)
##    Z : (2 x 120)
##    L : (120 x 120)
## Checking termination criterion
## Computing tmp
## ... stage 1
## ... stage 2
## Pre-computing LLT analysis
## Computing LLT
## Computing Q
## Computing C
## Computing W
## Computing Z
## Computing Y
## Clearing MST sparse matrix
## Setting up MST sparse matrix with 119

## <sparse>[ <logic> ] :  .M.sub.i.logical() maybe inefficient
```

Now that the space is reduced, it's time to order the cells. The call below has two important optional arguments. The first `num_paths` allows Monocle to assign cells to one of several alternative fates. In this case, we know the cells are transitioning along a single path, so by setting `num_paths`$= 1$, we allow only one cell fate. However, if we were dealing with a stem cell population that is differentiating and making fate decisions, we might allow for multiple outcomes. For example, we might be watching embryonic stem cells differentiate into ectoderm, mesoderm, and endoderm, and thus would set num_paths$= 3$. The second important argument is the reverse flag. Monocle won't be able to tell without some help which cells are at the beginning of the process and which are at the end. The reverse flag tells Monocle to reverse the orientation of the entire process as it's being discovered from the data, so that the cells that would have been assigned to the end are instead assigned to the beginning, and so on. Setting reverse to true will reverse the ordering of the cells in pseudotime, which also has a significant impact on the orientation of branches in the trajectory associated with cell fate decisions.

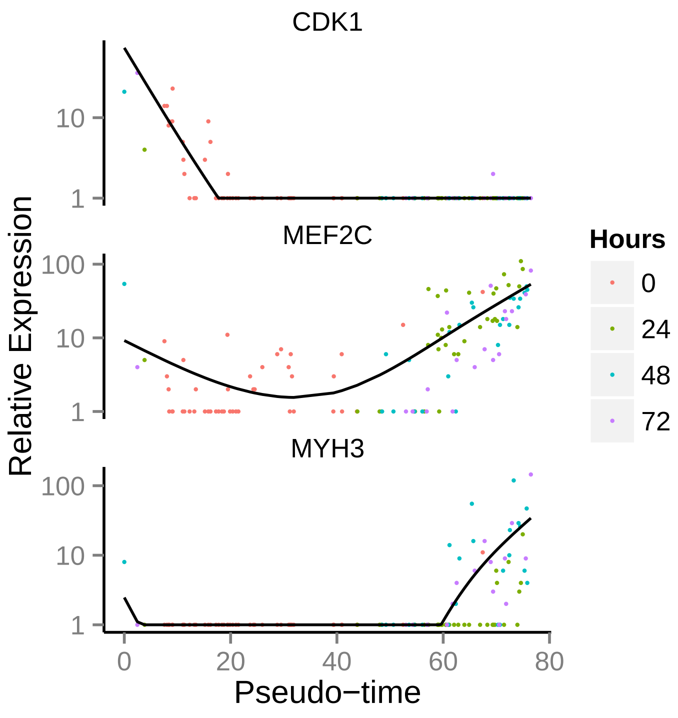
```
HSMM_myo <- orderCells(HSMM_myo, reverse=FALSE)

## [1] "the cells on the end of MST: "
## logical(0)

## Error in assign_cell_state_helper(res, res$root):  object 'curr_state' not found
```

Once the cells are ordered, we can visualize the trajectory in the reducted dimesional space.

```
plot_spanning_tree(HSMM_myo, color_by="Hours")

## Error in `colnames<-`(`*tmp*`, value = "sample_name"):  'names' attribute [1] must be the same
length as the vector [0]
```

To confirm that the ordering is correct, and to verify that we don't need to flip around the ordering using the reverse flag in orderCells, we can select a couple of markers of myogenic progress. Plotting these genes demonstrates that ordering looks good:

```
HSMM_filtered <- HSMM_myo[expressed_genes,]

my_genes <- row.names(subset(fData(HSMM_filtered),
                             gene_short_name %in% c("CDK1", "MEF2C", "MYH3")))

cds_subset <- HSMM_filtered[my_genes,]
plot_genes_in_pseudotime(cds_subset, color_by="Hours")
```



8 Advanced differential expression analysis

In this section, we'll explore how to use Monocle to find genes that are differentially expressed according to several different criteria. First, we'll look at how to use our previous classification of the cells by type to find genes that distinguish fibroblasts and myoblasts. Second, we'll look at how to find genes that are differentially expressed as a function of pseudotime, such as those that become activated or repressed during differentiation. Finally, you'll see how to perform multi-factorial differential analysis, which can help subtract the effects of confounding variables in your experiment.

To keep the vignette simple and fast, we'll be working with small sets of genes. Rest assured, however, that Monocle can analyze many thousands of genes even in large experiments, making it useful for discovering dynamically regulated genes during the biological process you're studying.

8.1 Finding genes that distinguish cell type or state

During a dynamic biological process such as differentiation, cells might assume distinct intermediate or final states. Recall that earlier we distinguished myoblasts from contaminating fibroblasts on the basis of several key markers. Let's look at several other genes that should distinguish between fibroblasts and myoblasts.

```r
to_be_tested <- row.names(subset(fData(HSMM),
                             gene_short_name %in% c("UBC", "NCAM1", "ANPEP")))
cds_subset <- HSMM[to_be_tested,]
```

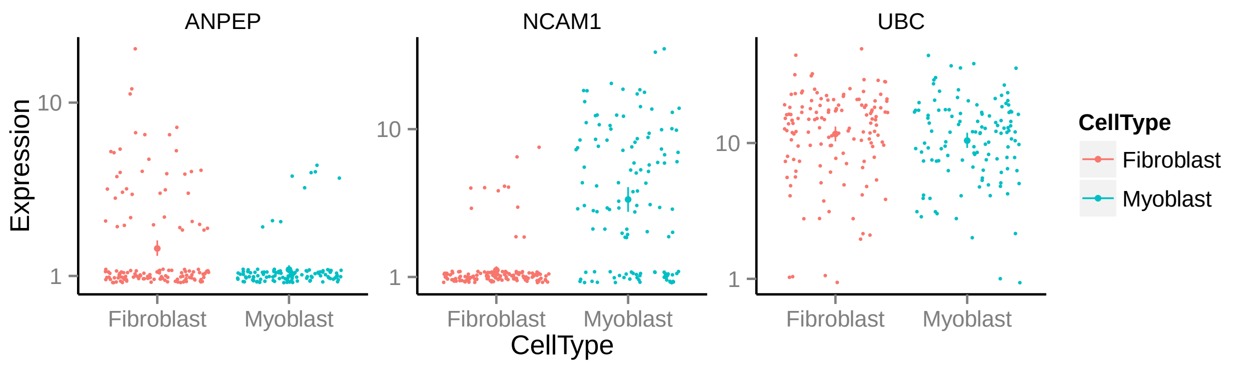
To test the effects of `CellType` on gene expression, we simply call `differentialGeneTest` on the genes we've selected. However, we have to specify a *model formula* in the call to tell Monocle that we care about genes with expression levels that depends on *CellType*. Monocle's differential expression analysis works essentially by fitting two models to the expression values for each gene, working through each gene independently. The first of the two models is called the *full* model. This model is essentially a way of predicting the expression value of each gene in a given cell knowing only whether that cell is a fibroblast or a myoblast. The second model, called the *reduced* model, does the same thing, but it doesn't know the `CellType` for each cell. It has to come up with a reasonable prediction of the expression value for the gene that will be used for *all* the cells. Because the full model has more information about each cell, it will do a better job of predicting the expression of the gene in each cell. The question Monocle must answer for each gene is *how much better* the full model's prediction is than the reduced model's. The greater the improvement that comes from knowing the `CellType` of each cell, the more significant the differential expression result. This is a common strategy in differential analysis, and we leave a detailed statistical exposition of such methods to others.

To set up the test based on `CellType`, we simply call `differentialGeneTest` with a string specifying `fullModelFormulaStr`. We don't have to specify the reduced model in this case, because the default of 1 is what we want here.

```r
diff_test_res <- differentialGeneTest(cds_subset, fullModelFormulaStr="~CellType", )
diff_test_res[,c("gene_short_name", "pval", "qval")]

##                    gene_short_name         pval         qval
## ENSG00000149294.12           NCAM1 1.082374e-24 3.247121e-24
## ENSG00000150991.10             UBC 1.547155e-01 1.547155e-01
## ENSG00000166825.9            ANPEP 8.423212e-06 1.263482e-05
```

Note that all the genes are significantly differentially expressed as a function of `CellType` except the housekeeping gene TBP, which we're using a negative control. However, we don't know which genes correspond to myoblast-specific genes (those more highly expressed in myoblasts versus fibroblast specific genes. We can again plot them with a jitter plot to see:

```r
plot_genes_jitter(cds_subset, grouping="CellType", color_by="CellType",
                  nrow=1, ncol=NULL, plot_trend=TRUE)

## geom_path:  Each group consist of only one observation.  Do you need to adjust the group aes-
thetic?
## geom_path:  Each group consist of only one observation.  Do you need to adjust the group aes-
thetic?
## geom_path:  Each group consist of only one observation.  Do you need to adjust the group aes-
thetic?
```



Note that we could also simply compute summary statistics such as mean or median expression level on a per-CellType basis to see this, which might be handy if we are looking at more than a handful of genes. Of course, we could test for genes that change as a function of Hours to find time-varying genes, or Media to identify genes that are responsive to the serum switch. In general, model formulae can contain terms in the pData table of the CellDataSet.

The differentialGeneTest function is actually quite simple "under the hood". The call above is equivalent to:

```
full_model_fits <- fitModel(cds_subset,  modelFormulaStr="~CellType")
reduced_model_fits <- fitModel(cds_subset, modelFormulaStr="~1")
diff_test_res <- compareModels(full_model_fits, reduced_model_fits)
diff_test_res
```

Occassionally, as we'll see later, it's useful to be able to call fitModel directly.

8.2 Finding genes that change as a function of pseudotime

Monocle's main job is to put cells in order of progress through a biological process (such as cell differentiation) without knowing which genes to look at ahead of time. Once it's done so, you can analyze the cells to find genes that changes as the cells make progress. For example, you can find genes that are significantly upregulated as the cells "mature". Let's look at a panel of genes important for myogenesis:

```
to_be_tested <- row.names(subset(fData(HSMM),
                              gene_short_name %in% c("MYH3", "MEF2C", "CCNB2", "TNNT1")))
cds_subset <- HSMM_myo[to_be_tested,]
```

Again, we'll need to specify the model we want to use for differential analysis. This model will be a bit more complicated than the one we used to look at the differences between CellType. Monocle assigns each cell a "pseudotime" value, which records its progress through the process in the experiment. The model can test against changes as a function of this value. Monocle uses the *VGAM* package to model a gene's expression level as a smooth, nonlinear function of pseudotime:

```
diff_test_res <- differentialGeneTest(cds_subset,  fullModelFormulaStr="~sm.ns(Pseudotime)")
```

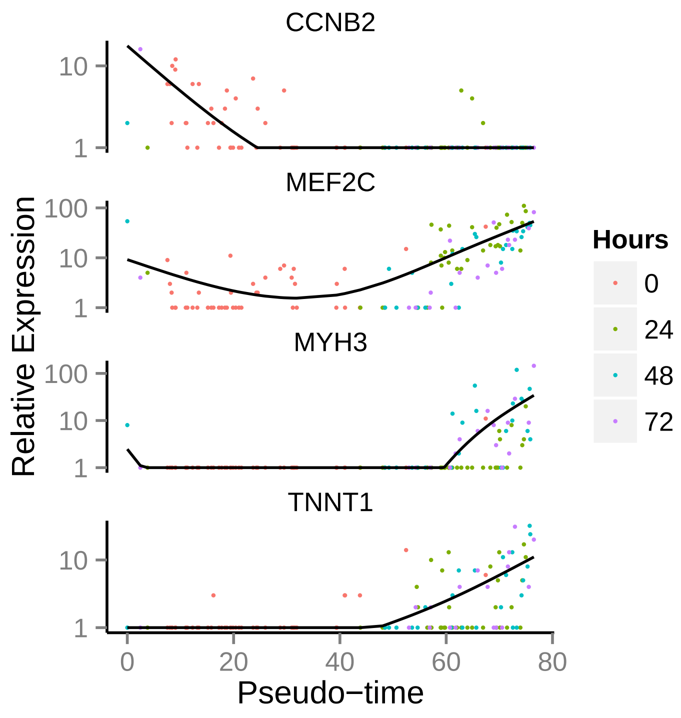
The sm.ns function states that Monocle should fit a natural spline through the expression values to help it describe the changes in expression as a function of progress. We'll see what this trend looks like in just a moment. Other smoothing functions are available.

Once again, let's add in the gene annotations so it's easy to see which genes are significant.

```
diff_test_res[,c("gene_short_name", "pval", "qval")]

##                     gene_short_name         pval         qval
## ENSG00000081189.9             MEF2C 1.138610e-08 1.518147e-08
## ENSG00000105048.12            TNNT1 1.020930e-13 4.083718e-13
## ENSG00000109063.9              MYH3 1.351282e-06 1.351282e-06
## ENSG00000157456.3             CCNB2 8.836854e-10 1.767371e-09
```

We can plot the expression levels of these genes, all of which show significant changes as a function of differentiation, using the function plot_genes_in_pseudotime. This function has a number of cosmetic options you can use to control the layout and appearance of your plot.

```
plot_genes_in_pseudotime(cds_subset, color_by="Hours")
```

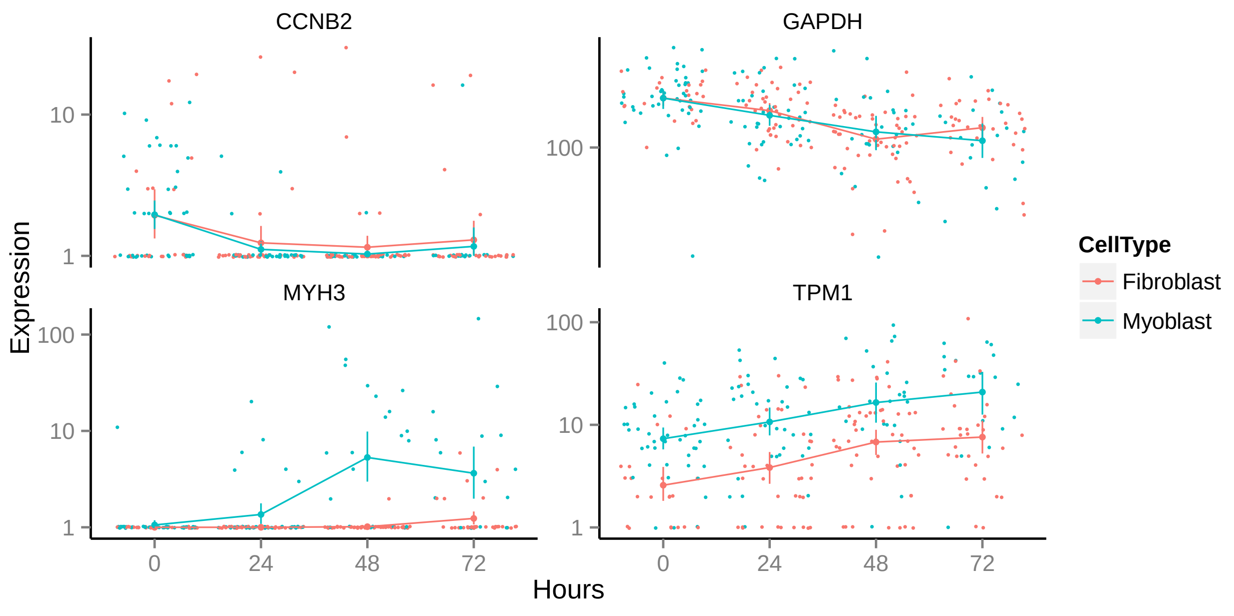


8.3 Multi-factorial differential expression analysis

Monocle can perform differential analysis in the presence of multiple factors, which can help you subtract some factors to see the effects of others. In the simple example below, Monocle tests three genes for differential expression between myoblasts and fibroblasts, while subtracting the effect of `Hours`, which encodes the day on which each cell was collected. To do this, we must specify both the full model and the reduced model. The full model captures the effects of both `CellType` and `Hours`, while the reduced model only knows about `Hours`.

When we plot the expression levels of these genes, we can modify the resulting object returned by `plot_genes_jitter` to allow them to have independent y-axis ranges, to better highlight the differences between cell states.

```
to_be_tested <- row.names(subset(fData(HSMM),
                              gene_short_name %in% c("TPM1", "MYH3", "CCNB2", "GAPDH")))
cds_subset <- HSMM[to_be_tested,]

diff_test_res <- differentialGeneTest(cds_subset,
                                      fullModelFormulaStr="~CellType + Hours",
                                      reducedModelFormulaStr="~Hours")
diff_test_res[,c("gene_short_name", "pval", "qval")]

##                     gene_short_name         pval         qval
## ENSG00000109063.9              MYH3 2.788118e-15 1.115247e-14
## ENSG00000111640.10            GAPDH 6.293919e-01 6.293919e-01
## ENSG00000140416.15             TPM1 1.788738e-12 3.577477e-12
## ENSG00000157456.3             CCNB2 3.968834e-02 5.291779e-02

plot_genes_jitter(cds_subset,
                  grouping="Hours", color_by="CellType", plot_trend=TRUE) +
  facet_wrap( ~ feature_label, scales="free_y")
```



9 Clustering genes by pseudotemporal expression pattern

A common question that arises when studying time-series gene expression studies is: "which genes follow similar kinetic trends"? Monocle can help you answer this question by grouping genes that have similar trends, so you can analyze these groups to see what they have in common. To do this, we'll first fit a smooth curve for each gene's expression trend as a function of pseudotime, then we'll group the genes according to similarity of these curves.

We start by using the model fitting function that's used during differential expression testing. This fitting procedure works gene by gene and can take a while, so we'll just work with 100 randomly chosen genes to keep the vignette small and fast.

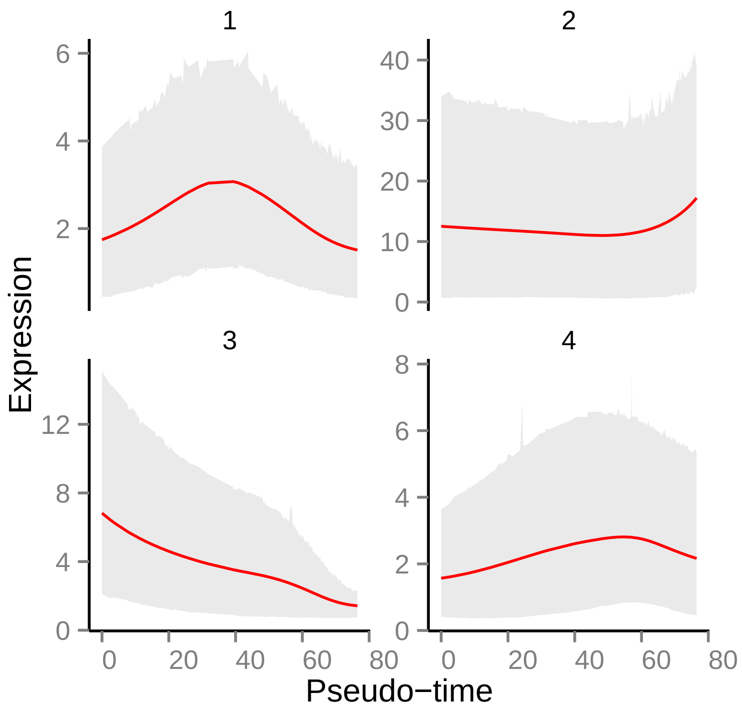
```
sampled_gene_cds <- HSMM_filtered[sample(nrow(fData(HSMM_filtered)), 100),]
full_model_fits <- fitModel(sampled_gene_cds,  modelFormulaStr="~sm.ns(Pseudotime, df=3)")
```

The `full_model_fits` list contains a model for each gene that we've chosen. We can generate a matrix of values where each row holds the predicted exression values for a gene over each cell, which correspond to the columns. Monocle provides the `responseMatrix` function to make this easy.

```
expression_curve_matrix <- responseMatrix(full_model_fits)
dim(expression_curve_matrix)

## [1] 100 120
```

Now we'll feed this matrix to a function, `clusterGenes`, that will cluster the genes into four groups:

```
clusters <- clusterGenes(expression_curve_matrix, k=4)
plot_clusters(HSMM_filtered, clusters)

## Warning:  'str_join' is deprecated.
## Use 'str_c' instead.
## See help("Deprecated")
```



The `plot_clusters` function returns a ggplot2 object showing the shapes of the expression patterns followed by the 100 genes we've picked out. The topographic lines highlight the distributions of the kinetic patterns relative to the overall trend lines, shown in red.

10 Citation

If you use Monocle to analyze your experiments, please cite:

```
citation("monocle")

##
##   Cole Trapnell and Davide Cacchiarelli et al (2014): The dynamics
##   and regulators of cell fate decisions are revealed by
##   pseudo-temporal ordering of single cells. Nature Biotechnology
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {The dynamics and regulators of cell fate decisions are revealed by pseudo-temporal order
##     author = {Cole Trapnell and Davide Cacchiarelli and Jonna Grimsby and Prapti Pokharel and Shuqian
##     year = {2014},
##     journal = {Nature Biotechnology},
##   }
```

11 Acknowledgements

Monocle was built by Cole Trapnell and Davide Cacchiarelli, with substantial design input John Rinn and Tarjei Mikkelsen. We are grateful to Sharif Bordbar, Chris Zhu, Amy Wagers and the Broad RNAi platform for technical assistance, and Magali Soumillon for helpful discussions. Cole Trapnell is a Damon Runyon Postdoctoral Fellow. Davide Cacchiarelli is a Human Frontier Science Program Fellow. Cacchiarelli and Mikkelsen were supported by the Harvard Stem Cell Institute. John Rinn is the Alvin and Esta Star Associate Professor. This work was supported by NIH grants 1DP2OD00667, P01GM099117, and P50HG006193-01. This work was also supported in part by the Single Cell Genomics initiative, a collaboration between the Broad Institute and Fluidigm Inc. This vignette was created

from Wolfgang Huber's Bioconductor vignette style document, and patterned after the vignette for *DESeq*, by Simon Anders and Wolfgang Huber.

12 Session Info

```
sessionInfo()

## R version 3.2.1 (2015-06-18)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.10.5 (Yosemite)
##
## 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] grid      splines   stats4    parallel  stats     graphics  grDevices
##  [8] utils     datasets  methods   base
##
## other attached packages:
##  [1] Hmisc_3.17-0           Formula_1.2-1          survival_2.38-3
##  [4] lattice_0.20-33        monocle_1.99.0         DDRTree_0.1
##  [7] irlba_2.0.0            Matrix_1.2-3           VGAM_1.0-0
## [10] RColorBrewer_1.1-2     HSMMSingleCell_0.102.0 ggplot2_1.0.1
## [13] reshape2_1.4.1         Biobase_2.28.0         BiocGenerics_0.14.0
## [16] knitr_1.11
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.2        formatR_1.2.1      plyr_1.8.3
##  [4] highr_0.5.1        tools_3.2.1        rpart_4.1-10
##  [7] digest_0.6.8       evaluate_0.8       gtable_0.1.2
## [10] igraph_1.0.1       DBI_0.3.1          proto_0.3-10
## [13] gridExtra_2.0.0    fastICA_1.2-0      stringr_1.0.0
## [16] dplyr_0.4.3        cluster_2.0.3      nnet_7.3-11
## [19] combinat_0.0-8     R6_2.1.1           foreign_0.8-66
## [22] pheatmap_1.0.7     latticeExtra_0.6-26 limma_3.24.15
## [25] magrittr_1.5       matrixStats_0.15.0 scales_0.3.0
## [28] MASS_7.3-45        assertthat_0.1     colorspace_1.2-6
## [31] labeling_0.3       stringi_1.0-1      acepack_1.3-3.3
## [34] munsell_0.4.2
```

References

[1] Cole Trapnell, Davide Cacchiarelli, Jonna Grimsby, Prapti Pokharel, Shuqiang Li, Michael Morse, Niall J. Lennon, Kenneth J. Livak, Tarjei S. Mikkelsen, and John L. Rinn. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nature Biotechnology*, 2014.

[2] P M Magwene, P Lizardi, and J Kim. Reconstructing the temporal ordering of biological samples using microarray data. *Bioinformatics*, 19(7):842–850, May 2003.

[3] Cole Trapnell, Adam Roberts, Loyal Goff, Geo Pertea, Daehwan Kim, David R Kelley, Harold Pimentel, Steven L Salzberg, John L Rinn, and Lior Pachter. Differential gene and transcript expression analysis of RNA-seq experiments with TopHat and Cufflinks. *Nature Protocols*, 7(3):562–578, March 2012.