scDD: A statistical approach for identifying differential distributions in single-cell RNA-seq experiments

Keegan Korthauer*

April 24, 2017

Contents

1 Introduction

The *scDD* package models single-cell gene expression data (from single-cell RNA-seq) using flexible nonparamentric Bayesian mixture models in order to explicitly handle heterogeneity within cell populations. In bulk RNA-seq data, where each measurement is an average over thousands of cells, distributions of expression over samples are most often unimodal. In single-cell RNA-seq data, however, even when cells represent genetically homogeneous populations, multimodal distributions of gene expression values over samples are common [1]. This type of heterogeneity is often treated as a nuisance factor in studies of differential expression in single-cell RNA-seq experiments. Here, we explicitly accommodate it in order to improve power to detect differences in expression distributions that are more complicated than a mean shift.

2 Background

Our aim is two-fold: (1) to detect which genes have different expression distributions across two biological conditions and (2) to classify those differences into informative patterns. Note that in (1) we explicitly say differences in 'distributions' rather than differences in 'average', which would correspond to traditional DE (differential expression) analysis in bulk
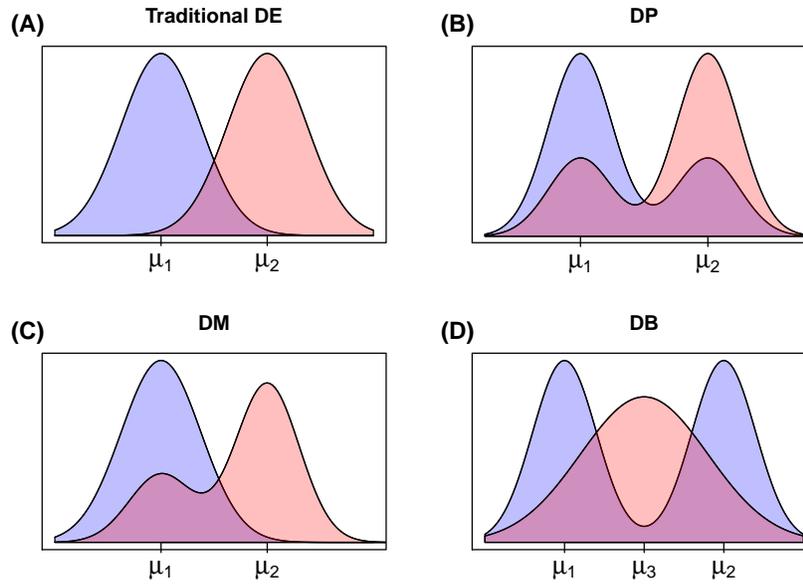
---

*keegan@jimmy.harvard.edu

Figure 1: **Illustration of informative DD patterns**

RNA-seq. By examining the entire distribution, we are able to detect more subtle differences as well as describe complex patterns, such as the existence of subgroups of cells within and across condition that express a given gene at a different level.

We start by assuming that the log-transformed nonzero expression values arise out of a Dirichlet Process Mixture of normals model. This allows us to characterize expression distributions in terms of the number of modes (or clusters). To detect differences in these distributions across conditions, an approximate Bayes Factor score is used which compares the conditional likelihood under the hypothesis of Equivalent distributions (ED) where one clustering process governs both conditions jointly, with the hypothesis of Differential distributions (DD) where each condition is generated from its own clustering process. In the full framework, significance of the scores for each gene are evaluated via empirical p-values after permutation. Optionally, a fast implementation obtains the p-values from the non-parametric Kolmogorov-Smirnov test. Zero values are considered by also implementing a χ^2 test of whether the proportion of zero values differs by condition (adjusted for overall sample detection rate). More details are provided in [1].

After the detection step is carried out, the significantly DD genes are classified into four informative patterns based on the number of clusters detected and whether they overlap. These patterns, depicted in Figure 1, include (a) DE (differential expression of unimodal genes), (b) DP (differential proportion for multimodal genes), (c) DM (differential modality), and (d) DB (both differential modality and different component means). Genes where a differential proportion of zeroes were identified are classified as DZ (differential zero). Genes that are identified as significantly differentially distributed but do not fall into one of the above categories are abbreviated NC (for no call). This includes genes with the same number of components with similar component means, but differential variance. For reasons detailed in [1], we do not aim to interpret this type of pattern.

The rest of this vignette outlines the main functionality of the *scDD* package. This includes:

- Identifying genes that are expressed differently between two biological conditions and classifying them into informative patterns.
- Simulating single-cell RNA-seq data with differential expression that exhibits multimodal patterns.
- Preprocessing and formatting of single-cell RNA-seq data to facilitate analysis
- Visualizing the expression patterns using a violin plotting scheme

3   Identify and Classify DD genes

In this section, we demonstrate how to use the main function `scDD` to find genes with differential distributions and classify them into the patterns of interest described in the previous section.

First, we need to load the *scDD* package. For each of the following sections in this vignette, we assume this step has been carried out.

```
library(scDD)
```

Next, we load the toy simulated example *ExpressionSet* object that we will use for identifying and classifying DD genes.

```
data(scDatExSim)
```

Verify that this object is a member of the *SummarizedExperiment* class and that it contains 200 samples and 30 genes. The `colData` slot (which contains a dataframe of metadata for the cells) should have a column that contains the biological condition or grouping of interest. In this example data, that variable is the 'condition' variable. Note that the input gene set needs to be in *SummarizedExperiment* format, and should contain normalized counts. In practice, it is also advisable to filter the input gene set to remove genes that have an extremely high proportion of zeroes (see Section 6). More specifically, the test for differential distributions of the expressed measurements will not be carried out on genes where only one or fewer cells had a nonzero measurement (these genes will still be tested for differential proportion of zeroes (DZ) if the `testZeroes` parameter is set to `TRUE`, however).

```
class(scDatExSim)

## [1] "SummarizedExperiment"
## attr(,"package")
## [1] "SummarizedExperiment"

dim(scDatExSim)

## [1]  30 200
```

Next, specify the hyperparameter arguments that we'll pass to the `scDD` function. These values reflect heavy-tailed distributions over the paramaters and are robust to many different settings in simulation (see [1] for more details).

```
prior_param=list(alpha=0.01, mu0=0, s0=0.01, a0=0.01, b0=0.01)
```

Finally, call the `scDD` function to test for differential distributions, classify DD genes, and return the results. If the biological condition or grouping variable in the `colData` slot is named something other than 'condition', you'll need to specify the name of the variable as an argument to the `scDD` function (set the `condition` argument equal to the name of the relevant column). We won't perform the test for a difference in the proportion of zeroes since none exists in this simulated toy example data, but this option can be invoked by changing the `testZeroes` option to `TRUE`. Note that the default option is to use a fast test of differential distributions that involves the Kolmogorov-Smirnov test instead of the full permutation testing framework. This provides a fast implementation of the method at the cost of potentially slightly decreased power compared to the full scDD framework described in the manuscript (see Section 4 for more details).

```
scDatExSim <- scDD(scDatExSim, prior_param=prior_param, testZeroes=FALSE)

## Clustering observed expression data for each gene

## Setting up parallel back-end using 4 cores

## Notice:  Number of permutations is set to zero; using
##           Kolmogorov-Smirnov to test for differences in distributions
##           instead of the Bayes Factor permutation test

## Classifying significant genes into patterns
```

Four results objects are added to the `scDatExSim` SummarizedExperiment object in the *metadata* slot. For convenience,

the results objects can be extracted with the `results` function.

The main results object is the `"Genes"` object which is a *data.frame* containing the following nine columns:

1. `gene`: gene name (matches rownames of SCdat)
2. `nonzero.pvalue`: p-value for KS test of differential distributions
3. `nonzero.pvalue.adj`: Benjamini-Hochberg adjusted p-value for KS test of differential distributions
4. `zero.pvalue`: p-value for test of difference in dropout rate (only for non-DD genes and if `testZeroes==TRUE`)
5. `zero.pvalue.adj`: Benjamini-Hochberg adjusted p-value for test of difference in dropout rate (only for non-DD genes and if `testZeroes==TRUE`)
6. `DDcategory`: name of the DD pattern (DE, DP, DM, DB, DZ), or NC (no call), or NS (not significant).
7. `Clusters.combined`: the number of clusters identified when pooling condition 1 and 2 together
8. `Clusters.c1`: the number of clusters identified in condition 1 alone
9. `Clusters.c2`: the number of clusters identified in condition 2 alone

This can be extracted using the following call to `results`:

```
RES <- results(scDatExSim)
head(RES)

##      gene nonzero.pvalue nonzero.pvalue.adj zero.pvalue zero.pvalue.adj DDcategory
## DE1  DE1    4.215186e-07       2.529112e-06          NA              NA         DB
## DE2  DE2    1.663921e-08       1.663921e-07          NA              NA         DE
## DE3  DE3    0.000000e+00       0.000000e+00          NA              NA         DE
## DE4  DE4    0.000000e+00       0.000000e+00          NA              NA         DE
## DE5  DE5    1.274147e-07       9.556106e-07          NA              NA         DE
## DP6  DP6    8.445703e-06       3.167138e-05          NA              NA         NC
##      Clusters.combined Clusters.c1 Clusters.c2
## DE1                  1           2           1
## DE2                  1           1           1
## DE3                  1           1           1
## DE4                  1           1           1
## DE5                  1           1           1
## DP6                  2           2           2
```

The remaining three results objects are matrices (first for condition 1 and 2 combined, then condition 1 alone, then condition 2 alone) that contain the cluster memberships (partition estimates) for each sample (for clusters 1,2,3,...) in columns and genes in rows. Zeroes, which are not involved in the clustering, are labeled as zero. These can be extracted by specifying an alternative `type` when calling the `results` function. For example, we can extract the partition estimates for condition 1 with the following:

```
PARTITION.C1 <- results(scDatExSim, type="Zhat.c1")
PARTITION.C1[1:5,1:5]

##      Sample1 Sample2 Sample3 Sample4 Sample5
## DE1        0       0       1       2       2
## DE2        1       1       1       1       0
## DE3        1       1       1       1       1
## DE4        1       1       1       1       1
## DE5        1       1       0       1       0
```

4   Alternate test for Differential Distributions

The first step in the scDD framework that identifies Differential Distributions was designed to have optimal power to detect differences in expression distributions, but the utilization of a permutation test on the Bayes Factor can be computationally demanding. While this is not an issue when machines with multiple cores are available since the code takes advantage of

parallel processing, we also provide the option to use an alternate test to detect distributional differences that avoides the use of a permutation test. This option (default) uses the Kolmogorov-Smirnov test, which examines the null hypothesis that two samples are generated from the same continuous distribution. While the use of this test yielded slighlty lower power in simulations than the full permutation testing framework at lower sample sizes (50-75 cells in each condition) and primarily affected the DB pattern genes, it does not require permutations and thus is orders of magnitude faster. The overall power to detect DD genes in simulation was still comparable or favorable to exisiting methods for differential expression analysis of scRNA-seq experiments.

The remaining steps of the scDD framework remain unchanged if the alternate test is used. That is, the Dirichlet process mixture model is still fit to the observed expression measurements so that the significant DD genes can be categorized into patterns that represent the major distributional changes, and results can still be visualized with violin plots using the `sideViolin` function described in the Plotting section.

The option to use the full permutation testing procedure instead of the Kolmogorov-Smirnov test is invoked by setting the number of permutations to something other than zero (the `permutations` argument in `scDD`) when calling the main `scDD` function as follows:

```
scDatExSim <- scDD(scDatExSim, prior_param=prior_param,
                   testZeroes=FALSE, permutations=100)

## Clustering observed expression data for each gene

## Setting up parallel back-end using 4 cores

## Performing permutations to evaluate independence of clustering
##               and condition for each gene

## Parallelizing by Genes

## Classifying significant genes into patterns
```

The line above will run 100 permutations of every gene. In practice, it is recommended that at least 1000 permutations are carried out if using the full permutation testing option. Note that this option will take significantly longer than the default option to use the alternate KS test, and computation time will increase with more genes and/or more permutations, but multiple cores will automatically be utilized (if available) via the *BiocParallel* package. By default, an OS appropriate back-end using the number of cores on the machine minus 2 is chosen automatically. Alternatively, you can specificy the number of cores to use by passing in a `param` argument in the `scDD` function call (where the `param` argument is an object of class `MulticoreParm` for Linux-like OS or `SnowParam` for Windows). For example, to use 12 cores on a Linux-like OS, specify `param=MulticoreParam(workers=12)`.

The results returned by `scDD` remain exactly as described in the previous section, with the exception that the `nonzero.pvalue` and `nonzero.pvalue.adj` columns of the `Genes` data frame now contain the p-values and Benjamini-Hochberg adjusted p-values of the perumtation test of the Bayes Factor for independence of condition membership with clustering.

5   Simulation

Here we show how to generate a simulated single-cell RNA-seq dataset which contains multi-modal genes. The `simulateSet` function simulates data from a two-condition experiment with a specified number of genes that fall into each of the patterns of interest. For DD genes, these include DE (differential expression of unimodal genes), DP (differential proportion for multimodal genes), DM (differential modality), and DB (both differential modality and mean expression levels), and for ED genes these include EE (equivalent expression for unimodal genes) and EP (equivalent proportion for multimodal genes). The simulation parameters are based on observed data from two conditions, so the function requires an *ExpressionSet* formatted dataset as input.

First, we load the toy example *ExpressionSet* to simulate from

```
data(scDatEx)
```

We'll verify that this object is a member of the *ExpressionSet* class and that it contains 142 samples and 500 genes

```
class(scDatEx)

## [1] "SummarizedExperiment"
## attr(,"package")
## [1] "SummarizedExperiment"

dim(scDatEx)

## [1] 500 142
```

Next we need to set the arguments that will be passed to the `simulateSet` function. In this example we will simulate 30 genes total, with 5 genes of each type and 100 samples in each of two conditions. We also set a random seed for reproducibility.

```
nDE <- 5
nDP <- 5
nDM <- 5
nDB <- 5
nEE <- 5
nEP <- 5
numSamples <- 100
seed <- 816
```

Finally, we'll create the simulated set with specified numbers of DE, DP, DM, DM, EE, and EP genes and specified number of samples, where DE gene fold changes represent 2 standard deviations of the observed fold change distribution, and multimodal genes have cluster mean distance of 4 standard deviations.

```
SD <- simulateSet(scDatEx, numSamples=numSamples,
                  nDE=nDE, nDP=nDP, nDM=nDM, nDB=nDB,
                  nEE=nEE, nEP=nEP, sd.range=c(2,2), modeFC=4, plots=FALSE,
                  random.seed=seed)

## Setting up parallel back-end using 4 cores

## Identifying a set of genes to simulate from...

## Simulating DE fold changes...

## Simulating individual genes...

## Done!  Simulated 5 DE, 5 DP, 5 DM, 5 DB, 5 EE, and 5 EP genes
```

The matrix in the first list element of the SD object contains simulated expression values. The second element stores the fold change/modal distance values which can be useful in assessing performance of a differential expression method.

6 Formatting and Preprocessing

Before beginning an analysis using *scDD*, you will need to carry out a few preprocessing steps. This includes normalization, filtering of genes that are mostly zero, and getting the data into format that is expected by the `scDD` function. The following subsections will detail these steps.

6.1 Constructing a SummarizedExperiment object

In this subsection, we provide a quick example of how to construct an object of the *SummarizedExperiment* class. For more detailed instructions, refer to the *SummarizedExperiment* package documentation.

Here we will convert the simulated data object SD returned by simulateSet (from the previous section) into a *SummarizedExperiment* object. First, load the *SummarizedExperiment* package:

```
library(SummarizedExperiment)
```

Next, create a vector of condition membership labels (these should be 1 or 2). In our simulated dataset, we generated numSamples samples in each of two conditions.

```
condition <- c(rep(1, numSamples), rep(2, numSamples))
```

The rows and columns of the expression matrix should have unique names, and the names of the columns should to correspond to the names of the condition membership labels in condition.

```
rownames(SD[[1]]) <- paste0(rownames(SD[[1]]), 1:nrow(SD[[1]]), sep="")
colnames(SD[[1]]) <- names(condition) <- paste0("Sample",
                                                1:ncol(SD[[1]]), sep="")
```

Once our labeling is intact, we can call the SummarizedExperiment function and specify the two relevant pieces of information. Optionally, additional experiment information can be stored in additional slots; see *SummarizedExperiment* package for more details.

```
SDSumExp <- SummarizedExperiment(assays=list("NormCounts"=SD[[1]]),
                                 colData=data.frame(condition))
```

6.2   Filtering and Normalization

In this subsection, we demonstrate the utility of the preprocess function, which can be helpful if working with raw data, or data which contains genes that are predominantly zero (common in single-cell RNA-seq experiments). This function takes as input a list of data matrices, one for each condition.

First, load the toy example data list:

```
data(scDatExList)
```

Verify that the data is formatted as a list of 2 matrices (one for each of 2 conditions), that each matrix has 100 rows (one for each gene), and that the number of columns in each matrix corresponds to the number of samples in each condition (78 and 64, respectively):

```
str(scDatExList)
```

```
## List of 2
##  $ C1: num [1:100, 1:78] 0 53.3 0 2 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:100] "MKL2" "CD109" "ABTB1" "MAST2" ...
##   .. ..$ : chr [1:78] "C1.073" "C1.074" "C1.075" "C1.076" ...
##  $ C2: num [1:100, 1:64] 6 17 0 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:100] "MKL2" "CD109" "ABTB1" "MAST2" ...
##   .. ..$ : chr [1:64] "C2.001" "C2.002" "C2.003" "C2.004" ...
```

Obtain the names of the conditions to pass to the preprocess function:

```
condition.names <- names(scDatExList)
```

Finally, apply the preprocess function to reformat the data into one data matrix with 100 rows and $78 + 64 = 142$ columns. In this example, we set the zero.thresh argument to 0.9 so that genes are filtered out if they are 90 percent zero.

```
scDatExMat <- preprocess(scDatExList, ConditionNames=condition.names,
                         zero.thresh=0.9)
```

Now, apply the preprocess function again, but this time use a more stringent threshold on the proportion of zeroes and apply normalization using size factors calculated using the *scran*. In this example, we set the `zero.thresh` argument to 0.75 so that genes with more than 75 percent zeroes are filtered out and we set the `scran_norm` argument to TRUE to return *scran* normalized counts.

```
datNorm.scran <- preprocess(scDatExList, ConditionNames=condition.names,
                            zero.thresh=0.75, scran_norm=TRUE)

## Performing scran Normalization
```

Also included is the option to use median normalization, invoked by setting `median_norm` to TRUE.

7   Plotting

Next we demonstrate the plotting routine that is implemented in the `sideViolin` function. This function produces side-by-side violin plots (where the curves represent a smoothed kernel density estimate) of the log-transformed data. A count of 1 is added before log-transformation so that zeroes can be displayed, but they are not included in the density estimation. Each condition is represented by one violin plot. Individual data points are plotted (with jitter) on top.

We illustrate this function by displaying the six types of simulated genes using the toy example simulated dataset. First, load the toy simulated dataset:

```
data(scDatExSim)
```

Next, load the *SummarizedExperiment* package to facilitate subset operations on SummarizedExperiment class objects:

```
library(SummarizedExperiment)
```

The following lines will produce the figures in Figure 2.

Plot side by side violin plots for Gene 1 (DE):

```
de <- sideViolin(normExprs(scDatExSim)[1,], scDatExSim$condition,
         title.gene=rownames(scDatExSim)[1])
```

Plot side by side violin plots for Gene 6 (DP):

```
dp <- sideViolin(normExprs(scDatExSim)[6,], scDatExSim$condition,
         title.gene=rownames(scDatExSim)[6])
```

Plot side by side violin plots for Gene 11 (DM):

```
dm <- sideViolin(normExprs(scDatExSim)[11,], scDatExSim$condition,
         title.gene=rownames(scDatExSim)[11])
```

Plot side by side violin plots for Gene 16 (DB):

```
db <- sideViolin(normExprs(scDatExSim)[16,], scDatExSim$condition,
         title.gene=rownames(scDatExSim)[16])
```

Plot side by side violin plots for Gene 21 (EP):

```
ep <- sideViolin(normExprs(scDatExSim)[21,], scDatExSim$condition,
         title.gene=rownames(scDatExSim)[21])
```

Plot side by side violin plots for Gene 26 (EE):

```
ee <- sideViolin(normExprs(scDatExSim)[26,], scDatExSim$condition,
         title.gene=rownames(scDatExSim)[26])
```

The plot objects returned by `sideViolin` are standard *ggplot2* objects, and thus can be manipulated into multipanel figures with the help of the *gridExtra* or *cowplot* packages. Here we use `grid.arrange` from the *gridExtra* package to visualize all the plots generated above. The end result is shown in Figure 2.

```r
library(gridExtra)
grid.arrange(de, dp, dm, db, ep, ee, ncol=2)
```

8   Session Info

Here is the output of `sessionInfo` on the system where this document was compiled:

```r
sessionInfo()
```

```
## R version 3.4.0 (2017-04-21)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.2 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.5-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.5-bioc/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8
##  [4] LC_COLLATE=C               LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
## [10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets  methods
## [9] base
##
## other attached packages:
##  [1] gridExtra_2.2.1           SummarizedExperiment_1.6.0 DelayedArray_0.2.0
##  [4] matrixStats_0.52.2        Biobase_2.36.0             GenomicRanges_1.28.0
##  [7] GenomeInfoDb_1.12.0       IRanges_2.10.0             S4Vectors_0.14.0
## [10] BiocGenerics_0.22.0       scDD_1.0.0
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-131         bitops_1.0-6         blockmodeling_0.1.8
##  [4] rprojroot_1.2        dynamicTreeCut_1.63-1 tools_3.4.0
##  [7] backports_1.0.5      R6_2.2.0             DT_0.2
## [10] vipor_0.4.5          KernSmooth_2.23-15   DBI_0.6-1
## [13] lazyeval_0.2.0       colorspace_1.3-2     compiler_3.4.0
## [16] labeling_0.3         EBSeq_1.16.0         caTools_1.17.1
## [19] scales_0.4.1         stringr_1.2.0        digest_0.6.12
## [22] minqa_1.2.4          rmarkdown_1.4        XVector_0.16.0
## [25] scater_1.4.0         htmltools_0.3.5      lme4_1.1-13
## [28] highr_0.6            limma_3.32.0         maps_3.1.1
## [31] htmlwidgets_0.8      RSQLite_1.1-2        FNN_1.1
## [34] shiny_1.0.2          zoo_1.8-0            mclust_5.2.3
## [37] BiocParallel_1.10.0  gtools_3.5.0         dplyr_0.5.0
## [40] RCurl_1.95-4.8       magrittr_1.5         GenomeInfoDbData_0.99.0
## [43] Matrix_1.2-9         ggbeeswarm_0.5.3     Rcpp_0.12.10
```
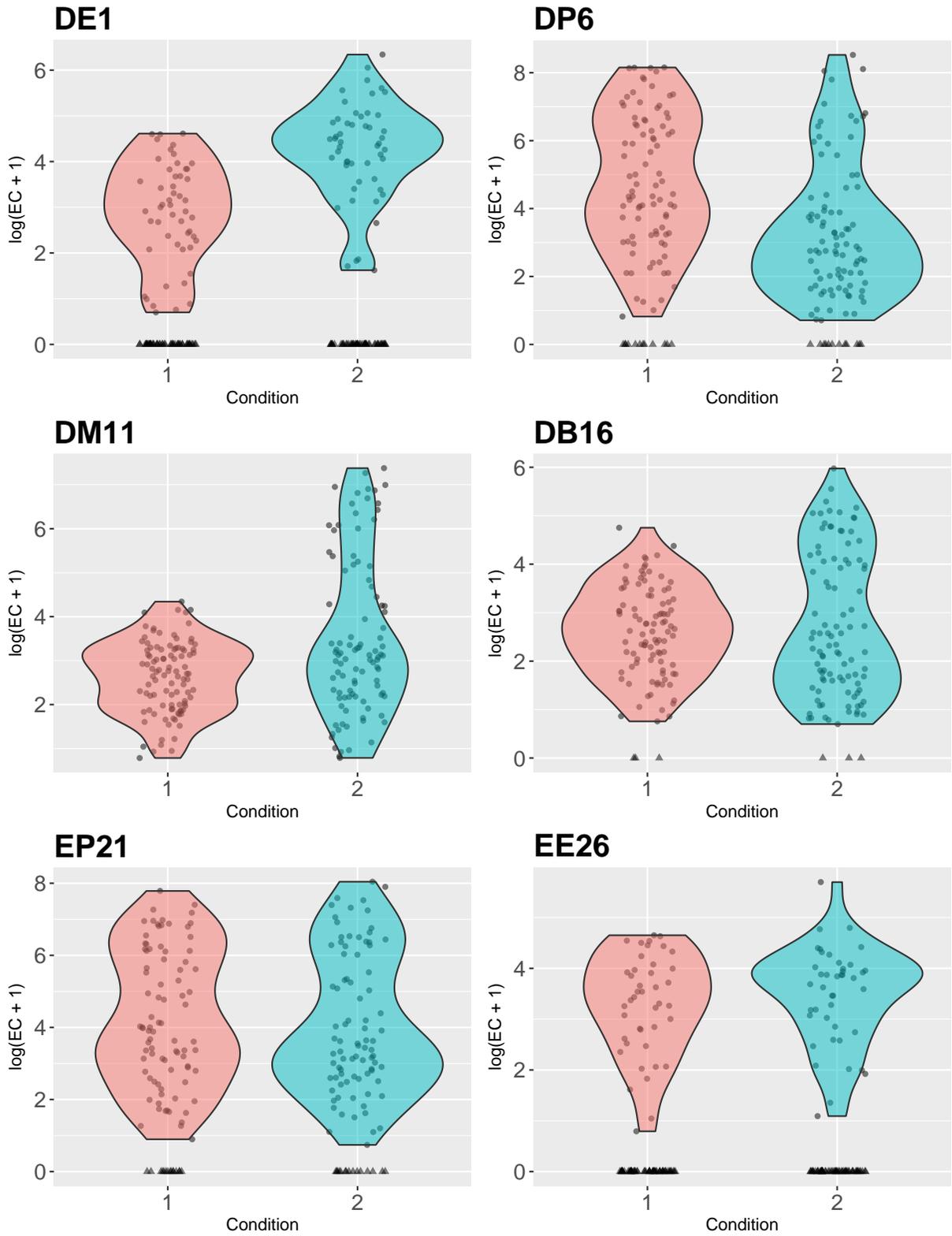
Figure 2: **Example Simulated DD genes**

```
## [46] munsell_0.4.3         abind_1.4-5            viridis_0.4.0
## [49] stringi_1.1.5         yaml_2.1.14            edgeR_3.18.0
## [52] MASS_7.3-47           zlibbioc_1.22.0        rhdf5_2.20.0
## [55] gplots_3.0.1          plyr_1.8.4             grid_3.4.0
## [58] gdata_2.17.0          shinydashboard_0.5.3   crayon_1.3.2
## [61] lattice_0.20-35       splines_3.4.0          locfit_1.5-9.1
## [64] knitr_1.15.1          igraph_1.0.1           rjson_0.2.15
## [67] reshape2_1.4.2        biomaRt_2.32.0         XML_3.98-1.6
## [70] evaluate_0.10         outliers_0.14          data.table_1.10.4
## [73] scran_1.4.0           spam_1.4-0             nloptr_1.0.4
## [76] httpuv_1.3.3          testthat_1.0.2        gtable_0.2.0
## [79] assertthat_0.2.0      ggplot2_2.2.1          mime_0.5
## [82] xtable_1.8-2          coda_0.19-1            viridisLite_0.2.0
## [85] tibble_1.3.0          arm_1.9-3              beeswarm_0.2.3
## [88] memoise_1.1.0         AnnotationDbi_1.38.0   tximport_1.4.0
## [91] fields_8.10           statmod_1.4.29         BiocStyle_2.4.0
```

References

[1] Keegan D Korthauer, Li-Fang Chu, Michael A. Newton, Yuan Li, James Thomson, Ron Stewart, and Christina Kendziorski. A statistical approach for identifying differential distributions in single-cell RNA-seq experiments. *Genome Biology*, 17:222, 10 2016.