Comparison of single-cells versus pool-and-split samples

Catalina A. Vallejos, Sylvia Richardson and John C. Marioni

17 December 2015

To demonstrate the efficacy of our method, we use the control experiment described in Grun et al (2014), where single mouse embryonic stem cells (mESCs) are compared against *pool-and-split* samples, consisting of pooled RNA from thousands of mESCs split into single-cell equivalent volumes. Such a controlled setting provides a situation where substantial changes in overall expression are not expected as, on average, the overall expression of single cells should match the levels measured on pool-and-split samples. Additionally, the design of pool-and-split samples removes biological variation, leading to a homogenous set of samples. Hence, pool-and-split samples are expected to show a genuine reduction in biological cell-to-cell heterogeneity when compared to single-cells.

In this document, we provide the R code used to perform the analysis described in the manuscript. To start the analysis, the following data must be dowloaded and stored in `data.path` directory.

- Expression counts. File 'GSE54695_data_transcript_counts.txt' (source: http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE54695).
- Concentrations in ERCC mix. File 'cms_095046.txt'. The column names of this file have been modified to be readable from R (source: https://tools.thermofisher.com/content/sfs/manuals/cms_095046.txt)

Additionally, the following R libraries must be loaded before performing the analysis

```
library(BASiCS)
library(data.table) # For fast pre-processing of large datasets
```

Data pre-processing

Loading the data

```
data = fread(file.path(data.path,"GSE54695_data_transcript_counts.txt"))
Gene.Ids = data$GENENAME
Cell.Ids = names(data)[-1]
RawCounts = cbind(subset(data, select = Cell.Ids[grep("SC_2i", Cell.Ids)]),
                  subset(data, select = Cell.Ids[grep("RNA_2i", Cell.Ids)]))

Cell.Colour =  c( rep("lightpink3",length(grep("SC_2i", Cell.Ids))),
                  rep("darkolivegreen3",length(grep("RNA_2i", Cell.Ids))) )
```

The input data contains 12535 genes and 160 cells.

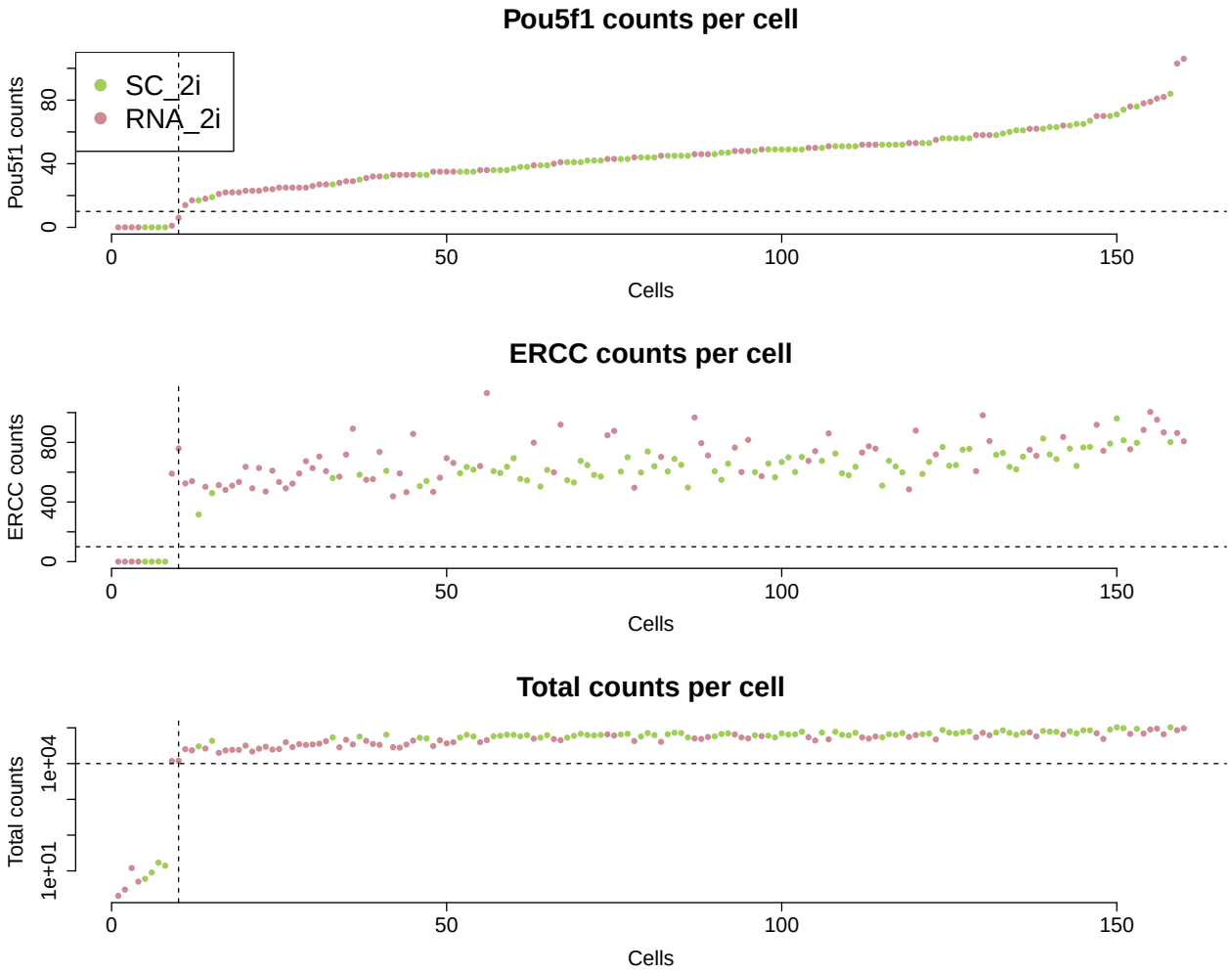
Transforming the data into UMI counts

```
# Function provided by Jong Kyoung Kim (EMBL-EBI)
UMICount <- function(MoleculeCount, UMILength)
{
  # MoleculeCount is the normalized count
  M = 4^UMILength
  UMICount = M*(1-exp(-MoleculeCount/M))
  return(UMICount)
}

CountsUMI = round(UMICount(RawCounts, 4))
```

Quality control: filtering cells

Before running the analysis it is important to filter out samples with poor quality.

```
Pou5f1.per.cell <- as.numeric(CountsUMI[which(Gene.Ids == "Pou5f1"),])
counts.per.cell <- colSums(CountsUMI)
genes.per.cell <- apply(CountsUMI, 2, function(x) sum( x>0 ))
ercc.per.cell <- colSums(CountsUMI[grep("ERCC", Gene.Ids),])
par(mfrow = c(3,1))
par(cex.lab = 1.5, cex.axis = 1.5, cex.main = 2)
plot(Pou5f1.per.cell[order(Pou5f1.per.cell)], pch = 16,
     col = Cell.Colour[order(Pou5f1.per.cell)],
     bty = "n", xlab = "Cells", ylab = "Pou5f1 counts",
     main = "Pou5f1 counts per cell")
abline(h = 10, lty = 2)
abline(v = 10, lty = 2)
legend('topleft', c("SC_2i", "RNA_2i"), col = unique(Cell.Colour)[2:1], pch = 16, cex = 2)
plot(ercc.per.cell[order(Pou5f1.per.cell)], pch = 16,
     col = Cell.Colour[order(Pou5f1.per.cell)],
     bty = "n", xlab = "Cells", ylab = "ERCC counts",
     main = "ERCC counts per cell")
abline(h = 100, lty = 2)
abline(v = 10, lty = 2)
plot(counts.per.cell[order(Pou5f1.per.cell)], pch = 16,
     col = Cell.Colour[order(Pou5f1.per.cell)], log = "y",
     bty = "n", xlab = "Cells", ylab = "Total counts",
     main = "Total counts per cell")
abline(h = 1e4, lty = 2)
abline(v = 10, lty = 2)
```




```
CountsUMI_1 <- CountsUMI[,Pou5f1.per.cell >= 10, with = FALSE]
Cell.Colour_1 <- Cell.Colour[Pou5f1.per.cell >= 10]
```

Following Grun et al (2014), we discarded those cells where less than 10 transcripts of *Pou5f1* were detected (because poor sequencing efficiency por potential undergoing differentiation). We also notice that most of these discarded cells also exhibit particularly low total counts (especially for the ERCCs). After this filter, **current processed data contains 12535 genes and 150 cells (74 and 76 single cells and pool-and-split samples, respectively).**

Filtering of transcripts (removing the low signal genes)

For the analysis, we only include transcripts with at least 50 counts in total across all cells.

```
CountsUMI_2 = CountsUMI_1[rowSums(CountsUMI_1) >= 50, ]
Gene.Ids_2 = Gene.Ids[rowSums(CountsUMI_1) >= 50]
```

After this filter, **current processed data contains 9378 (9343 and 35 intrinsic and spike-in genes, respectively).**

BASiCS analysis

The input dataset

To use perform comparisons between two groups of cells, the current implementation of `BASiCS` requires the creation of a `BASiCSDV_Data` object containing information regarding all cells and two separate `BASiCS_Data` objects to contain separate information regarding each group. This is a temporarily solution as the library will be updated shortly to allow a unique `BASiCS_Data` object containing all the required the information. In the meantime, we also provide a `BASiCSDV` library, where the functions related to the differential expression analysis (mean and over-dispersion) are implemented. A new official release of the `BASiCS` library, including all data analyses, will be uploaded to github during January.

Spike-in genes information

In addition to the matrix of counts, the creation of `BASiCS_Data` and `BASiCSDV_Data` objects requires information regarding the number of spike-in molecules that are theoretically added to each cell. These quantities can be calculated based on the concentrations of ERCC molecules in the mix and the dilution factors used for library preparation. These are calculated below.

```
SpikeInfo <- fread(file.path(data.path,"cms_095046.txt"))
SpikeInfo <- SpikeInfo[SpikeInfo$ERCC_ID %in% Gene.Ids_2[grep("ERCC", Gene.Ids_2)],]
SpikeInfo$MoleculesPerCell <- SpikeInfo$concentration_in_Mix_1 * (1e-18) * (6.022e23) * (1/2500000)

# To confirm with the 3.3% capture indicated by Grun et al
SpikeOut <- data.table("ERCC_ID" = Gene.Ids_2[grep("ERCC", Gene.Ids_2)],
                       "ERCC_MeanCount" = rowMeans(CountsUMI_2[grep("ERCC", Gene.Ids_2),]))
SpikeOut = merge(SpikeInfo, SpikeOut, by = "ERCC_ID")
SpikeInfoFilter = subset(SpikeInfo, select = c(ERCC_ID, MoleculesPerCell))
```

Overall, a 3.55 % of the added ERCC molecules where captured during the experiment.

Re-ordering of genes

Additionally, we reorder the matrix of counts to bring spike-in molecules to the bottom of the table. The creation of the `BASiCSDV_Data` and `BASiCS_Data` objects also requires, a logical vector to indicate whether a gene is intrinsic of a spike-in. This is stored in a `Tech` vector.

```
# Input number of molecules for spike-in genes
SpikeInput = SpikeInfo$MoleculesPerCell

# Creating indicator of technical genes
Tech = grepl("ERCC", Gene.Ids_2)

# Ordering the data so spike-in genes are at the bottom.
CountsUMI_2 = rbind(CountsUMI_2[!Tech,], CountsUMI_2[Tech,])
Gene.Ids_2 = c(Gene.Ids_2[!Tech], Gene.Ids_2[Tech])
Tech = c(Tech[!Tech],Tech[Tech])
```

Separating expression counts for each condition

A final step before creating the required `BASiCS_Data` objects is to separate the matrix of counts according to the grouping of cells.

```
CountsUMI_SC = as.matrix(CountsUMI_2)[, grep("SC_2i",colnames(CountsUMI_2))]
CountsUMI_RNA = as.matrix(CountsUMI_2)[, grep("RNA_2i",colnames(CountsUMI_2))]
rownames(CountsUMI_SC) <- Gene.Ids_2
rownames(CountsUMI_RNA) <- Gene.Ids_2
```

Creating the input object

```
Data = newBASiCS_DV_Data(CountsTest = CountsUMI_SC,
                         CountsRef = CountsUMI_RNA,
                         Tech = Tech,
                         SpikeInputTest = SpikeInput,
                         SpikeInputRef = SpikeInput)
```

```
## An object of class BASiCS_DV_Data
##  Dataset contains 9378 genes (9343 biological and 35 technical) and 150 cells.
##         - 74 cells in the test sample
##         - 76 cells in the reference sample
##  Elements (slots): CountsTest, CountsRef, Tech, SpikeInputTest and SpikeInputRef.
##
## NOTICE: BASiCS_D requires a pre-filtered dataset
##     - You must remove poor quality cells before creating the BASiCS data object
##     - We recommend to pre-filter very lowly expressed transcripts before creating the object.
##       Inclusion criteria may vary for each data. For example, remove transcripts
##           - with very low total counts across of all cells
##           - that are only expressed in few cells
##             (by default genes expressed in only 1 cell are not accepted)
##           - with very low total counts across the cells where the transcript is expressed
##
##  BASiCS_DV_Filter can be used for this purpose
```

```
Data.SC = newBASiCS_Data(Counts = CountsUMI_SC,
                         Tech = Tech,
                         SpikeInfo = SpikeInfoFilter)
```

```
## An object of class BASiCS_Data
##  Dataset contains 9378 genes (9343 biological and 35 technical) and 74 cells.
##  Elements (slots): Counts, Tech, SpikeInput, GeneNames and BatchInfo.
##  The data contains 1 batch.
##
## NOTICE: BASiCS requires a pre-filtered dataset
##     - You must remove poor quality cells before creating the BASiCS data object
##     - We recommend to pre-filter very lowly expressed transcripts before creating the object.
##       Inclusion criteria may vary for each data. For example, remove transcripts
##           - with very low total counts across of all of the samples
##           - that are only expressed in a few cells
##             (by default genes expressed in only 1 cell are not accepted)
##           - with very low total counts across the samples where the transcript is expressed
##
##  BASiCS_Filter can be used for this purpose
```

```
Data.RNA = newBASiCS_Data(Counts = CountsUMI_RNA,
                          Tech = Tech,
                          SpikeInfo = SpikeInfoFilter)
```

```
## An object of class BASiCS_Data
##  Dataset contains 9378 genes (9343 biological and 35 technical) and 76 cells.
##  Elements (slots): Counts, Tech, SpikeInput, GeneNames and BatchInfo.
##  The data contains 1 batch.
##
## NOTICE: BASiCS requires a pre-filtered dataset
##     - You must remove poor quality cells before creating the BASiCS data object
##     - We recommend to pre-filter very lowly expressed transcripts before creating the object.
##       Inclusion criteria may vary for each data. For example, remove transcripts
##           - with very low total counts across of all of the samples
##           - that are only expressed in a few cells
##             (by default genes expressed in only 1 cell are not accepted)
##           - with very low total counts across the samples where the transcript is expressed
##
##  BASiCS_Filter can be used for this purpose
```

Fitting the BASiCS model

To run the MCMC algorithm, we use the function `BASiCS_MCMC`. As a default, we used `a2.mu = 0.5` and `a2.delta = 0.5`.

```
N = 40000; Thin = 20; Burn = 20000
RunName.SC = paste0("Grun_SplitFilter2i_SC_",N)
RunName.RNA = paste0("Grun_SplitFilter2i_RNA_",N)

MCMC_Output.SC <- BASiCS_MCMC(Data.SC, N = N, Thin = Thin, Burn = Burn,
                              PrintProgress = TRUE, StoreChains = TRUE,
                              StoreDir = chains.path, RunName = RunName.SC)

MCMC_Output.RNA <- BASiCS_MCMC(Data.RNA, N = N, Thin = Thin, Burn = Burn,
                              PrintProgress = TRUE, StoreChains = TRUE,
                              StoreDir = chains.path, RunName = RunName.RNA)
```

Loading pre-computed chains

This report shows the results of pre-computed chains as defined by the previous chunk of code.

```
ChainMuTest = as.matrix(fread(paste0(chains.path, "chain_mu_Grun_Split2i_SC_40000.txt")))
ChainMuRef = as.matrix(fread(paste0(chains.path, "chain_mu_Grun_Split2i_RNA_40000.txt")))
ChainDeltaTest = as.matrix(fread(paste0(chains.path, "chain_delta_Grun_Split2i_SC_40000.txt")))
ChainDeltaRef = as.matrix(fread(paste0(chains.path, "chain_delta_Grun_Split2i_RNA_40000.txt")))
ChainPhiTest = as.matrix(fread(paste0(chains.path, "chain_phi_Grun_Split2i_SC_40000.txt")))
ChainPhiRef = as.matrix(fread(paste0(chains.path, "chain_phi_Grun_Split2i_RNA_40000.txt")))
ChainSTest = as.matrix(fread(paste0(chains.path, "chain_s_Grun_Split2i_SC_40000.txt")))
ChainSRef = as.matrix(fread(paste0(chains.path, "chain_s_Grun_Split2i_RNA_40000.txt")))
ChainNuTest = as.matrix(fread(paste0(chains.path, "chain_nu_Grun_Split2i_SC_40000.txt")))
ChainNuRef = as.matrix(fread(paste0(chains.path, "chain_nu_Grun_Split2i_RNA_40000.txt")))
```

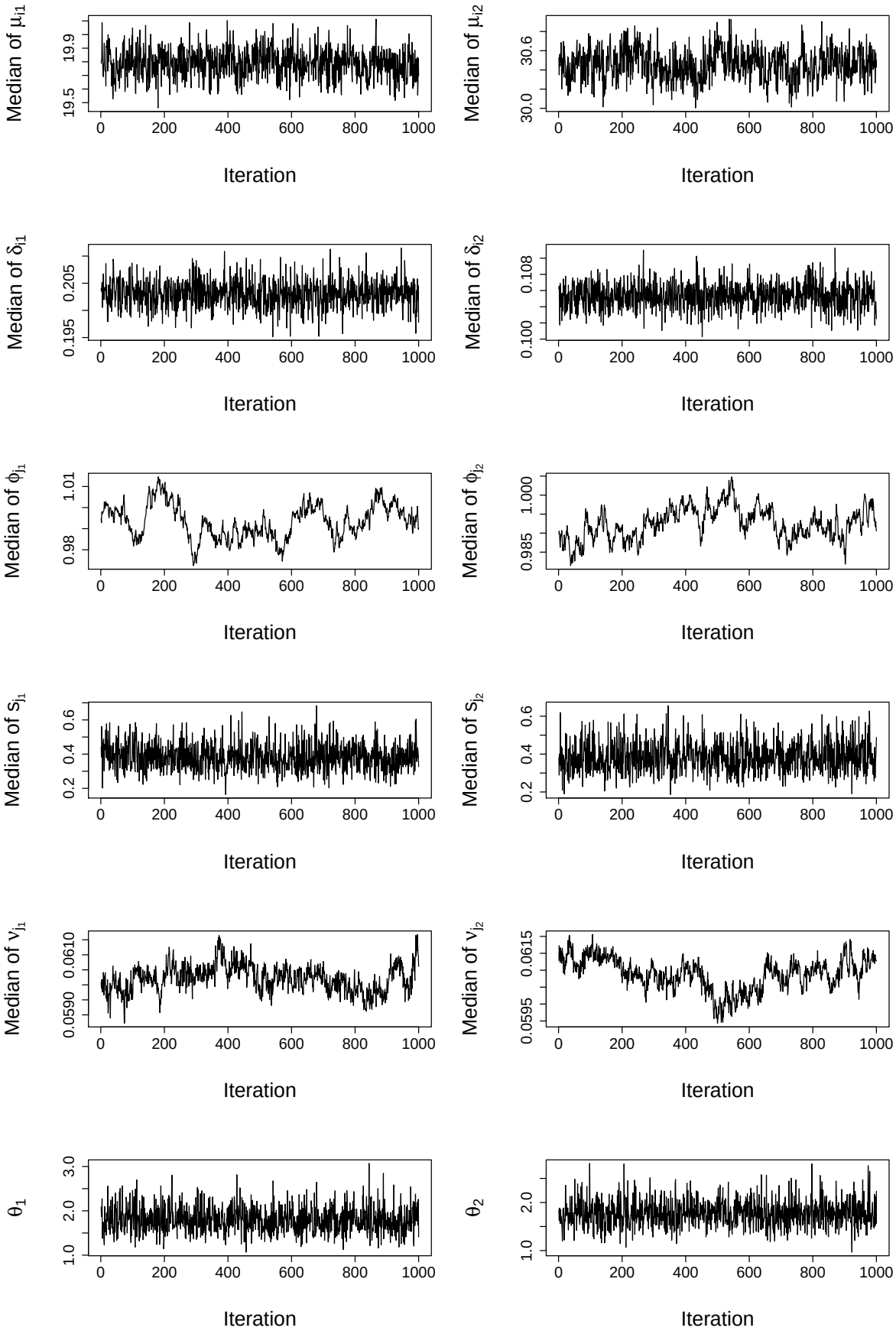
```
ChainThetaTest = fread(paste0(chains.path, "chain_theta_Grun_Split2i_SC_40000.txt"))$Batch1
ChainThetaRef = fread(paste0(chains.path, "chain_theta_Grun_Split2i_RNA_40000.txt"))$Batch1

MCMC_Output <- newBASiCS_DV_Chain(muTest = ChainMuTest, muRef = ChainMuRef,
                                  deltaTest = ChainDeltaTest, delta = ChainDeltaRef,
                                  phi = cbind(ChainPhiTest, ChainPhiRef),
                                  s = cbind(ChainSTest, ChainSRef),
                                  nu = cbind(ChainNuTest, ChainNuRef),
                                  thetaTest = ChainThetaTest,
                                  thetaRef = ChainThetaRef)
```

```
## An object of class BASiCS_DV_Chain
##  1000 MCMC samples.
##  Dataset contains 9343 biological genes and 150 cells (in total across both samples).
##  Elements (slots): muTest, muRef, deltaTest, omegaRef, phi, s, nu, thetaTest and thetaRef.
```

Convergence diagnostics

To assess convergence of the chain, the convergence diagnostics provided by the package `coda` can be used. Additionally, a visual inspection can be performed using traceplot. As an illustration, here we display traceplots for medians across groups of parameters (e.g. the median across all $\mu[i1]$'s). It can be seen that the mixing of the chains related to ϕ_j's and ν_j's mixes less well. Nonetheless, their values are still concentrated on a very small range.

```
nTest = ncol(Data@CountsTest)
nRef = ncol(Data@CountsRef)
n = nTest + nRef
par(mgp = c(5,1,0)); par(mar = c(7,9,4,0.5)); par(mfrow = c(6,2))
par(cex.lab = 2, cex.axis = 1.5)
plot(apply(MCMC_Output@muTest, 1, median), type = "l",
     ylab = expression(paste("Median of ",mu[i1])), xlab = "Iteration")
plot(apply(MCMC_Output@muRef, 1, median), type = "l",
     ylab = expression(paste("Median of ",mu[i2])), xlab = "Iteration")
plot(apply(MCMC_Output@deltaTest, 1, median), type = "l",
     ylab = expression(paste("Median of ",delta[i1])), xlab = "Iteration")
plot(apply(MCMC_Output@deltaRef, 1, median), type = "l",
     ylab = expression(paste("Median of ",delta[i2])), xlab = "Iteration")
plot(apply(MCMC_Output@phi[,1:nTest], 1, median), type = "l",
     ylab = expression(paste("Median of ",phi[j[1]])), xlab = "Iteration")
plot(apply(MCMC_Output@phi[,(nTest+1):n], 1, median), type = "l",
     ylab = expression(paste("Median of ",phi[j[2]])), xlab = "Iteration")
plot(apply(MCMC_Output@s[,1:nTest], 1, median), type = "l",
     ylab = expression(paste("Median of ",s[j[1]])), xlab = "Iteration")
plot(apply(MCMC_Output@s[,(nTest+1):n], 1, median), type = "l",
     ylab = expression(paste("Median of ",s[j[2]])), xlab = "Iteration")
plot(apply(MCMC_Output@nu[,1:nTest], 1, median), type = "l",
     ylab = expression(paste("Median of ",nu[j[1]])), xlab = "Iteration")
plot(apply(MCMC_Output@nu[,(nTest+1):n], 1, median), type = "l",
     ylab = expression(paste("Median of ",nu[j[2]])), xlab = "Iteration")
plot(MCMC_Output@thetaTest, type = "l",
     ylab = expression(theta[1]), xlab = "Iteration")
plot(MCMC_Output@thetaRef, type = "l",
     ylab = expression(theta[2]), xlab = "Iteration")
```


Median of μ_{i1}
19.5
19.9
Median of μ_{i2}
30.0
30.6
Median of δ_{i1}
0.195
0.205
Median of δ_{i2}
0.100
0.108
Median of ϕ_{i1}
0.98
1.01
Median of ϕ_{i2}
0.985
1.000
Median of s_{i1}
0.2
0.4
0.6
Median of s_{i2}
0.2
0.4
0.6
Median of ν_{i1}
0.0590
0.0610
Median of ν_{i2}
0.0595
0.0615
θ_1
1.0
2.0
3.0
θ_2
1.0
2.0
0
200
400
600
800
1000
Iteration

Offset correction

Once the model has been fitted, possible offset effects are corrected using the following function.

```
OffSetCorrection <- function(MCMC_Output)
{
  median(rowSums(MCMC_Output@muRef)/rowSums(MCMC_Output@muTest))
}
```

```
OffSet = OffSetCorrection(MCMC_Output)
OffSet
```

```
## [1] 1.47201
```

Hereafter, results are displayed based on offset corrected chains.

```
MCMC_Output2 <- newBASiCS_DV_Chain(muTest = ChainMuTest,
                                   muRef = ChainMuRef / OffSet,
                                   deltaTest = ChainDeltaTest,
                                   delta = ChainDeltaRef,
                                   phi = cbind(ChainPhiTest, ChainPhiRef * OffSet),
                                   s = cbind(ChainSTest, ChainSRef),
                                   nu = cbind(ChainNuTest, ChainNuRef),
                                   thetaTest = ChainThetaTest,
                                   thetaRef = ChainThetaRef)
```

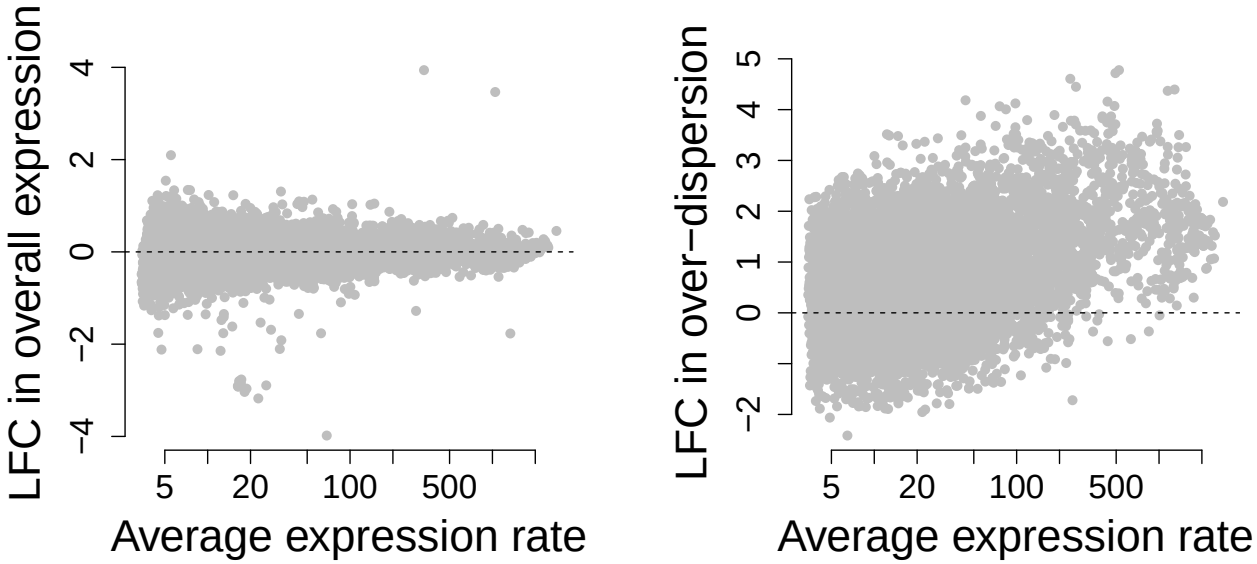
```
## An object of class BASiCS_DV_Chain
##  1000 MCMC samples.
##  Dataset contains 9343 biological genes and 150 cells (in total across both samples).
##  Elements (slots): muTest, muRef, deltaTest, omegaRef, phi, s, nu, thetaTest and thetaRef.
```

Differential expression analysis (mean and over-dispersion)

To visualise gene-wide behaviour of log-fold change estimates, we create the following plots.

```
MedianMuRef = apply(MCMC_Output2@muRef, 2, median)[!Tech]
MedianMuTest = apply(MCMC_Output2@muTest, 2, median)[!Tech]
MuBase=(MedianMuRef * nRef + MedianMuTest * nTest)/n

ChainTau = log(MCMC_Output2@muTest / MCMC_Output2@muRef)
ChainOmega = log(MCMC_Output2@deltaTest / MCMC_Output2@deltaRef)
MedianTau = apply(ChainTau, 2, median)
MedianOmega = apply(ChainOmega, 2, median)

par(mfrow = c(1,2))
par(mar = c(5, 6, 5, 2) + 0.1)
```

```
par(cex.axis = 1.5)

plot(MuBase, MedianTau, pch = 16, col = "grey", bty = "n", log = "x",
     xlab = "Average expression rate", cex.lab = 2,
     ylab = "LFC in overall expression")
abline(h = 0, lty = 2)

plot(MuBase, MedianOmega, pch = 16, col = "grey", bty = "n", log = "x",
     xlab = "Average expression rate", cex.lab = 2,
     ylab = "LFC in over-dispersion")
abline(h = 0, lty = 2)
```




```
Test40 <- BASiCS_DV_TestDE(Data, MCMC_Output2, GeneNames = Gene.Ids_2[!Tech],
                           EpsilonM = 0.4, EpsilonD = 0.4,
                           EFDR_M = 0.05, EFDR_D = 0.05,
                           OrderVariable = "GeneIndex",
                           GroupLabelRef = "P&S", GroupLabelTest = "SC")
```

```
## ------------------------------------------------------------------------
## 557  genes with a change on the overall expression:
## - Higher expression in  SC group: 250
## - Higher expression in  P&S group: 307
## - Fold change tolerance =  40 %
## - Evidence threshold =  0.85425
## - EFDR =  5 %
## - EFNR =  18.65 %
## ------------------------------------------------------------------------
##
## ------------------------------------------------------------------------
## 2355  genes with a change on the cell-to-cell biological over dispersion:
## - Higher over dispersion in  SC group: 2291
## - Higher over dispersion in  P&S group: 64
## - Fold change tolerance =  40 %
## - Evidence threshold =  0.87925
```

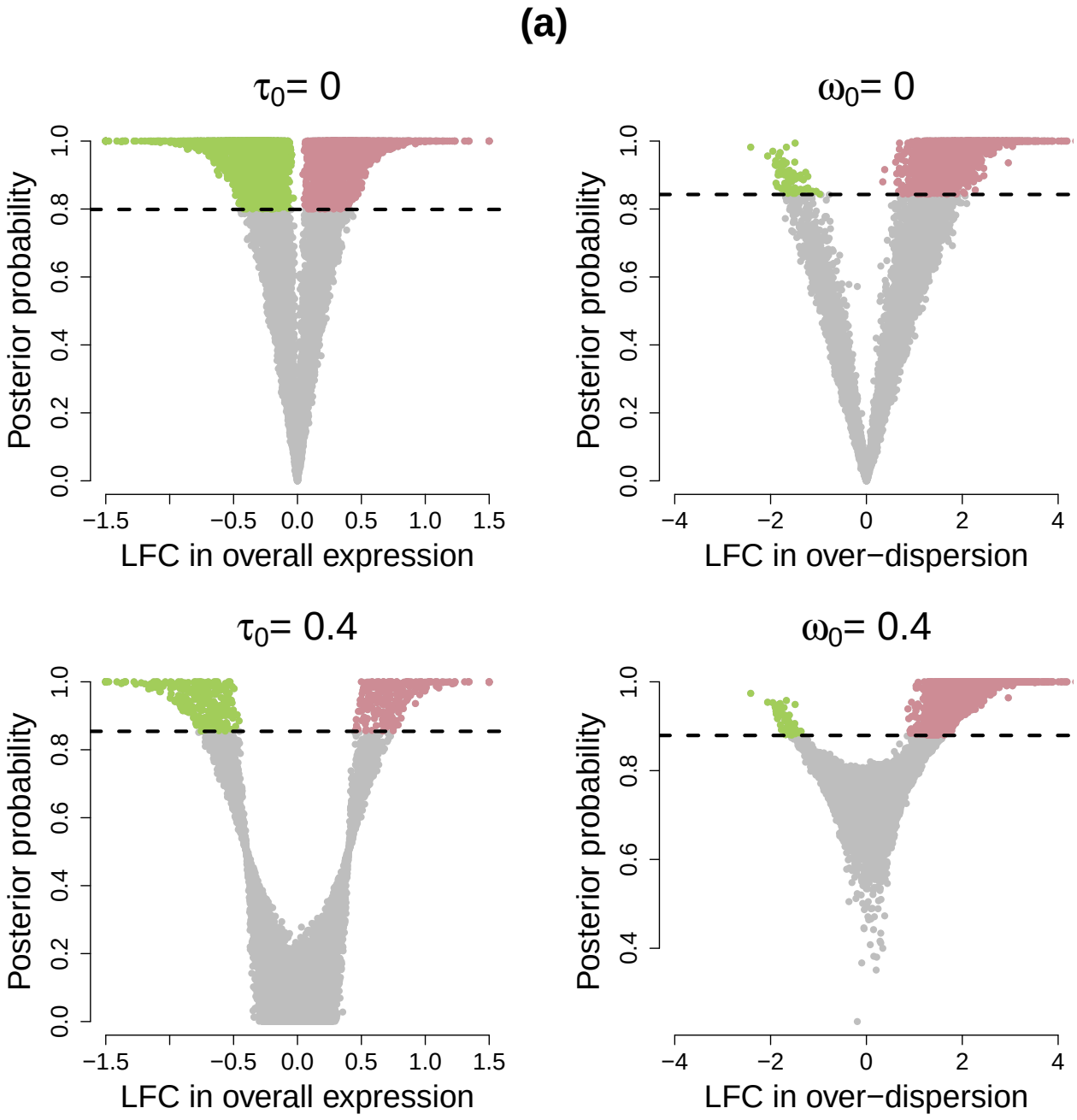
```
## - EFDR =  4.97 %
## - EFNR =  76.28 %
## ----------------------------------------------------------------------
```

```
Test0 <- BASiCS_DV_TestDE(Data, MCMC_Output2, GeneNames = Gene.Ids_2[!Tech],
                          EpsilonM = 0, EpsilonD = 0,
                          EFDR_M = 0.05, EFDR_D = 0.05,
                          OrderVariable = "GeneIndex",
                          GroupLabelRef = "P&S", GroupLabelTest = "SC")
```

```
## ----------------------------------------------------------------------
## 4688  genes with a change on the overall expression:
## - Higher expression in  SC group: 1912
## - Higher expression in  P&S group: 2776
## - Fold change tolerance =  0 %
## - Evidence threshold =  0.79875
## - EFDR =  4.97 %
## - EFNR =  43.39 %
## ----------------------------------------------------------------------
##
## ----------------------------------------------------------------------
## 2277  genes with a change on the cell-to-cell biological over dispersion:
## - Higher over dispersion in  SC group: 2212
## - Higher over dispersion in  P&S group: 65
## - Fold change tolerance =  0 %
## - Evidence threshold =  0.84275
## - EFDR =  5.03 %
## - EFNR =  41.78 %
## ----------------------------------------------------------------------
```

```
par(mfrow = c(2,2))
par(mar = c(5, 6, 4, 2) + 0.1, oma = c(0,0,3,0))
par(cex.axis = 1.5, cex.lab = 2)
MedianTauAux0 = ifelse(abs(Test0$Table$ExpLogFC) > 1.5,
                       1.5 * sign(Test0$Table$ExpLogFC), Test0$Table$ExpLogFC)
pchAux = ifelse(abs(Test0$Table$ExpLogFC) > 1.5, 4, 16)
plot(MedianTauAux0, Test0$Table$ProbDiffExp, bty = "n",
     main = expression(paste(tau[0], "= 0")), cex.main = 2.5,
     xlab = "LFC in overall expression",
     ylab = "Posterior probability",
     pch = 16,
     col = "grey", xlim = c(-1.5, 1.5))
points(MedianTauAux0[Test0$Table$ResultDiffExp == "SC+"],
       Test0$Table$ProbDiffExp[Test0$Table$ResultDiffExp == "SC+"],
       pch = 16,
       col = "lightpink3")
points(MedianTauAux0[Test0$Table$ResultDiffExp == "P&S+"],
       Test0$Table$ProbDiffExp[Test0$Table$ResultDiffExp == "P&S+"],
       pch = 16,
       col = "darkolivegreen3")
abline(h = Test0$DiffExpSummary$EviThreshold, lty = 2, lwd = 3)

plot(Test0$Table$OverDispLogFC, Test0$Table$ProbDiffOverDisp, bty = "n",
```

```
     main = expression(paste(omega[0], "= 0")), cex.main = 2.5,
     xlab = "LFC in over-dispersion",
     ylab = "Posterior probability",
     pch = 16, col = "grey", xlim = c(-4, 4))
points(Test0$Table$OverDispLogFC[Test0$Table$ResultDiffOverDisp == "SC+"],
       Test0$Table$ProbDiffOverDisp[Test0$Table$ResultDiffOverDisp == "SC+"],
       pch = 16, col = "lightpink3")
points(Test0$Table$OverDispLogFC[Test0$Table$ResultDiffOverDisp == "P&S+"],
       Test0$Table$ProbDiffOverDisp[Test0$Table$ResultDiffOverDisp == "P&S+"],
       pch = 16, col = "darkolivegreen3")
abline(h = Test0$DiffOverDispSummary$EviThreshold, lty = 2, lwd = 3)

MedianTauAux40 = ifelse(abs(Test40$Table$ExpLogFC) > 1.5,
                        1.5 * sign(Test40$Table$ExpLogFC), Test40$Table$ExpLogFC)
plot(MedianTauAux40, Test40$Table$ProbDiffExp, bty = "n",
     main = expression(paste(tau[0], "= 0.4")), cex.main = 2.5,
     xlab = "LFC in overall expression",
     ylab = "Posterior probability",
     pch = 16,
     col = "grey", xlim = c(-1.5, 1.5))
points(MedianTauAux40[Test40$Table$ResultDiffExp == "SC+"],
       Test40$Table$ProbDiffExp[Test40$Table$ResultDiffExp == "SC+"],
       pch = 16,
       col = "lightpink3")
points(MedianTauAux40[Test40$Table$ResultDiffExp == "P&S+"],
       Test40$Table$ProbDiffExp[Test40$Table$ResultDiffExp == "P&S+"],
       pch = 16,
       col = "darkolivegreen3")
abline(h = Test40$DiffExpSummary$EviThreshold, lty = 2, lwd = 3)

plot(Test40$Table$OverDispLogFC, Test40$Table$ProbDiffOverDisp, bty = "n",
     main = expression(paste(omega[0], "= 0.4")), cex.main = 2.5,
     xlab = "LFC in over-dispersion",
     ylab = "Posterior probability",
     pch = 16, col = "grey", xlim = c(-4, 4))
points(Test40$Table$OverDispLogFC[Test40$Table$ResultDiffOverDisp == "SC+"],
       Test40$Table$ProbDiffOverDisp[Test40$Table$ResultDiffOverDisp == "SC+"],
       pch = 16, col = "lightpink3")
points(Test40$Table$OverDispLogFC[Test40$Table$ResultDiffOverDisp == "P&S+"],
       Test40$Table$ProbDiffOverDisp[Test40$Table$ResultDiffOverDisp == "P&S+"],
       pch = 16, col = "darkolivegreen3")
abline(h = Test40$DiffOverDispSummary$EviThreshold, lty = 2, lwd = 3)

title("(a)", outer=TRUE, cex.main = 2.5)
```

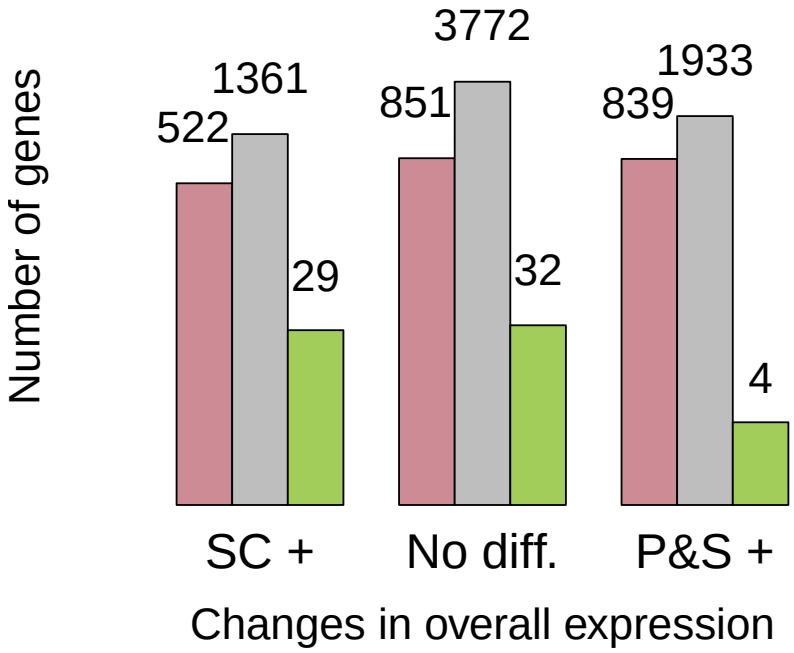



```
Result0 = paste0(Test0$Table$ResultDiffExp,Test0$Table$ResultDiffOverDisp)

NamesBarplot = c(names(table(Result0)))
ColourBarplot = c(rep(unique(Cell.Colour_1)[1], 3),
                  rep("grey", 3),
                  rep(unique(Cell.Colour_1)[2], 3))

par(mfrow = c(2,1))
par(mar = c(4, 4, 4, 2) + 0.1, oma = c(0,0,3,0))
#par(mgp = c(4, 1, 0))
par(cex.main = 2, cex.axis = 1.5, cex.lab = 1.5)
```

```
barplot(table(Test0$Table$ResultDiffOverDisp,
              Test0$Table$ResultDiffExp)[c(3,1,2),c(3,1,2)] + 1,
        col = unique(ColourBarplot), beside = TRUE, cex.names = 1.7,
        ylim = c(1, 35000), xlab = "Changes in overall expression",
        main = expression(paste(tau[0], "= 0, ", omega[0], "= 0")),
        ylab = "Number of genes", axes = FALSE, log = "y",
        names.arg = c("SC +", "No diff.", "P&S +"))
text(x = c(1.3, 2.5, 3.5, 5.3, 6.5, 7.5, 9.3, 10.5, 11.5),
     y = 3 * as.vector(table(Test0$Table$ResultDiffOverDisp,
                             Test0$Table$ResultDiffExp)[c(3,1,2),c(3,1,2)]),
     as.vector(table(Test0$Table$ResultDiffOverDisp,
                             Test0$Table$ResultDiffExp)[c(3,1,2),c(3,1,2)]),
     cex = 1.5, col = "black")


barplot(table(Test40$Table$ResultDiffOverDisp,
              Test40$Table$ResultDiffExp)[c(3,1,2),c(3,1,2)] + 1,
        col = unique(ColourBarplot), beside = TRUE, cex.names = 1.7,
        ylim = c(1, 35000), xlab = "Changes in overall expression",
        main = expression(paste(tau[0], "= 0.4, ", omega[0], "= 0.4")),
        ylab = "Number of genes", axes = FALSE, log = "y",
        names.arg = c("SC +", "No diff.", "P&S +"))
text(x = c(1.5, 2.5, 3.5, 5.2, 6.5, 7.5, 9.3, 10.5, 11.5),
     y = pmax(5, 3 * as.vector(table(Test40$Table$ResultDiffOverDisp,
                             Test40$Table$ResultDiffExp)[c(3,1,2),c(3,1,2)])),
     as.vector(table(Test40$Table$ResultDiffOverDisp,
                             Test40$Table$ResultDiffExp)[c(3,1,2),c(3,1,2)]),
     cex = 1.5, col = "black")

title("(b)", outer=TRUE, cex.main = 2)
```

(b)

$\tau_0 = 0$, $\omega_0 = 0$



$\tau_0 = 0.4$, $\omega_0 = 0.4$