Learning the BASiCS: a Bayesian approach to single-cell RNA-seq data analysis

MLPM Summer School 2015

Catalina A. Vallejos, Nils Eling, John C. Marioni

22 September 2015

Setting up the R session

Data pre-processing

Running BASiCS

Post-processing of BASiCS results

And beyond . . .

Questions?

Setting up the R session

Before we start...

Open a new session of R Studio

- To clean the existig R environment use

```
rm(list = ls())
```

- Install `BiocGenerics` from BioConductor

```
source("http://bioconductor.org/biocLite.R")
biocLite("BiocGenerics")
```

- Install BASiCS from Github

```
library(devtools)
install_github('catavallejos/BASiCS')
```

Before we start...

Load the libraries that will be used througout the analysis

- To perform the analysis

```
library(BASiCS)
```

- For fast pre-processing of large datasets

```
# install.packages("data.table")
library(data.table)
```

Data pre-processing

Example dataset

To illustrate BASiCS, we analyse the mouse embryonic stem cell (ESC) dataset described in Islam et al (2014)[1]. In these data, expression counts are recorded in terms of Unique Molecular Identifiers (UMI), removing amplification biases.

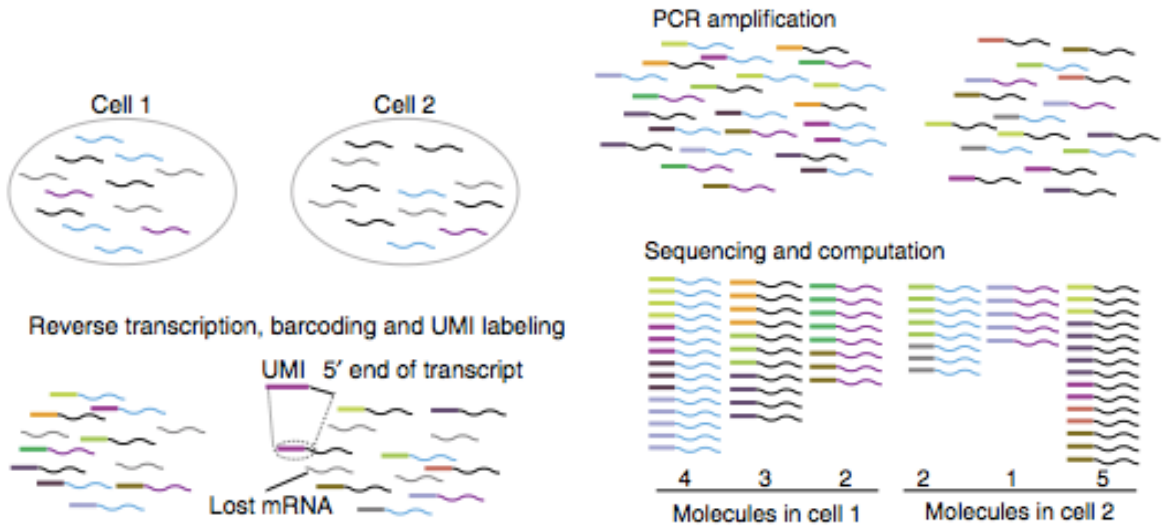


Figure 1: Illustration of the use of UMIs (source: Islam et al, 2014).

[1] Islam et al (2014). Quantitative single-cell RNA-seq with unique molecular identifiers. *Nature Methods*. doi:10.1038/nmeth.2772

Loading the example dataset

Let `data.path` be the directory where the data is stored, e.g.

```
data.path = "/Users/catalinavallejos/Documents/MRC/Projects/SCE/LaTeX/BASiCS/AnalysisMouseESC/"
```

The following files must be stored in the `data.path` directory:

- **Expression counts**: `GSE46980_CombinedMoleculeCounts.tab`
- **Quality control information**: `187_3lanes_CA.txt` (provided by Sten Linnarsson).
- **Input molecules of spike-in genes**: `SilverBulletCTRLConc.txt` (provided by Sten Linnarsson).

These files are provided within the materials of this tutorial.

Loading the data

To read the matrix of expression counts (excluding metadata) use:

```
Counts <- read.table(file.path(data.path,
                               "GSE46980_CombinedMoleculeCounts.tab"),
                     skip = 7, sep = "\t",
                     colClasses = c(rep("NULL", 7),
                                    rep("numeric", 96)))
dim(Counts)
```

This dataset contains 25914 genes (one at each row) and 96 cells (one at each column).

Metadata (gene names)

Gene names are given by

```
Genes <- read.table(file.path(data.path,
                              "GSE46980_CombinedMoleculeCounts.tab"),
                              skip = 7, sep = "\t",
                              colClasses = c("character",
                                             rep("NULL", 102)))[,1]
rownames(Counts) <- Genes
```

Metadata (cell identifiers)

Cell identifiers are given by

```
Cells <- read.table(file.path(data.path,
                              "GSE46980_CombinedMoleculeCounts.tab"),
                              skip = 5, nrows = 1, header = F)[-1]
Cells <- as.vector(t(Cells))
colnames(Counts) <- Cells
```

What does the data look like?

```
head(Counts[, 1:10], n = 10)
```

```
##                  A01 B01 C01 D01 E01 F01 G01 H01 A02 B02
## RNA_SPIKE_MC01     0   0   0   0   0   0   0   0   0   0
## RNA_SPIKE_MC02     0   7   2   8   4   1   3   0   0   1
## RNA_SPIKE_MC04     0   0   0   0   0   0   0   0   0   0
## RNA_SPIKE_MC07     0   0   0   0   0   0   0   0   0   0
## RNA_SPIKE_MC08     0   0   0   0   0   0   0   0   0   0
## RNA_SPIKE_MC09     0   0   0   0   0   0   0   0   0   0
## RNA_SPIKE_MC10     0   0   0   0   0   0   0   0   0   0
## RNA_SPIKE_MC14     4   2   3   5   1  10   2   1   3   1
## RNA_SPIKE_MC19     6   2   5   2   4   0   1   0   4   0
## RNA_SPIKE_MC20     6   4   1   1   2   0   0   3   0   1
```

...

⋮

Filtering cells

We adopt the same quality control criterion as in Islam et al (2014).

```
QC_Info <- read.table(file.path(data.path,
                                "187_3lanes_CA.txt"),
                      header = TRUE)
GoodCells <- QC_Info$Well[QC_Info$GoodCell==1]
```

We also discard 9 cells that are not ESCs (information provided by Sten Linnarsson)

```
NotESC <- c("D02", "E02", "A06", "H07", "D08",
            "A09", "G10", "F12", "G12")
GoodCells <- GoodCells[!(GoodCells %in% NotESC)]
```

Filtering cells

To remove cells that do not pass the inclusion criteria use

```
CountsQC <- subset(Counts,
                   select = Cells %in% GoodCells)
dim(CountsQC)
```

After this filter, 41 cells are left to be analysed.

Filtering of genes

To compute the total number of counts (over all cells) per gene use

```
TotCountsPerGene = rowSums(CountsQC)
sum(TotCountsPerGene == 0)
```

- We observe 10871 genes with zero counts
- Many other genes have just a few counts (in a few cells!)

Filtering of genes

Here, we only include those genes with (on average) at least 1 count per cell (i.e. when total counts are at least equal to the number of analysed cells).

```
GenesInclude = TotCountsPerGene >= 41
CountsQC = as.matrix(CountsQC[GenesInclude, ])
dim(CountsQC)
```

After this filter, 7941 genes are left to be analysed.

Running BASiCS

Building the BASiCS input dataset

As an input, BASiCS requires an object of class `BASiCS_Data`. To create this object, we need the following elements

- A matrix of expression counts, whose `rownames` contain the associated gene names
- A logical vector indicating whether or not each gene is a technical spike
- A `data.frame` whose first and second columns contain the gene names assigned to the spike-in genes and the associated input number of molecules, respectively.

Building the BASiCS input dataset

First, we create a variable indicating whether or not a gene is a technical spike (for every gene)

```
TechQC = grepl("SPIKE", rownames(CountsQC))
```

REMINDER: Gene names must be stored as `rownames(CountsQC)`

Building the BASiCS input dataset

Secondly, we need the input number of spike-in molecules per cell. This is provided in `SilverBulletCTRLConc.txt`.

```
SpikeInfo <- read.table(file.path(data.path,
                                  "SilverBulletCTRLConc.txt"),
                        sep = "\t", header = TRUE,
                        colClasses = c(rep("NULL", 5),
                                       "character",
                                       rep("NULL", 3),
                                       "numeric"))
head(SpikeInfo, n = 4)
```

```
##                     Name molecules_in_each_chamber
## 1         RNA_SPIKE_MC28                  4042.8912
## 2 RNA_SPIKE_MJ-1000-67                   252.6807
## 3  RNA_SPIKE_MJ-500-35                  2021.4456
## 4         RNA_SPIKE_TagJ                   252.6807
```

Building the BASiCS input dataset

NOTE: These values can be calculated using experimental information. For each spike-in gene i, we use

$$\mu_i = C_i \times 10^{-18} \times (6.022 \times 10^{23}) \times (9 \times 10^{-3}) \times D \quad \text{where,}$$

- C_i is the concentration of the spike i in the ERCC mix
- 10^{-18} is to convert att to mol
- 6.022×10^{23} is the Avogadro number (mol $\rightarrow$ molecule)
- 9×10^{-3} is the volume added into each chamber
- D is a dilution factor

Building the BASiCS input dataset

This file include spike-in genes that did not pass the inclusion criteria. To remove these use

```
SpikeInfoQC <- SpikeInfo[SpikeInfo$Name %in% rownames(CountsQC)[TechQC],]
```

Building the BASiCS input dataset

Finally, to create a `BASiCS_Data` object use

```
Data = newBASiCS_Data(Counts = CountsQC,
                      Tech = TechQC,
                      SpikeInfo = SpikeInfoQC)
```

```
## An object of class BASiCS_Data
##  Dataset contains 7941 genes (7895 biological and 46 technical) and 41 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
```

The BASiCS model

For each gene i and cell j the model implemented in BASiCS is

$$X_{ij}|\mu_i, \phi_j, \nu_j, \rho_{ij} \sim \begin{cases} \text{Poisson}(\phi_j \nu_j \mu_i \rho_{ij}), & \text{if gene } i \text{ is biological;} \\ \text{Poisson}(\nu_j \mu_i), & \text{if gene } i \text{ is a technical spike, with} \end{cases}$$

$$\nu_j|s_j, \theta \sim \text{Gamma}(1/\theta, 1/(s_j\theta)) \qquad \text{and} \qquad \rho_{ij}|\delta_i \sim \text{Gamma}(1/\delta_i, 1/\delta_i),$$

Parameters' interpretation

- ϕ_j and s_j are cell-specific normalising constants
- ν_j and θ capture technical noise
- μ_i is the overall expression rate of a gene i
- δ_i controls cell-to-cell biological variability of a gene i

The BASiCS model

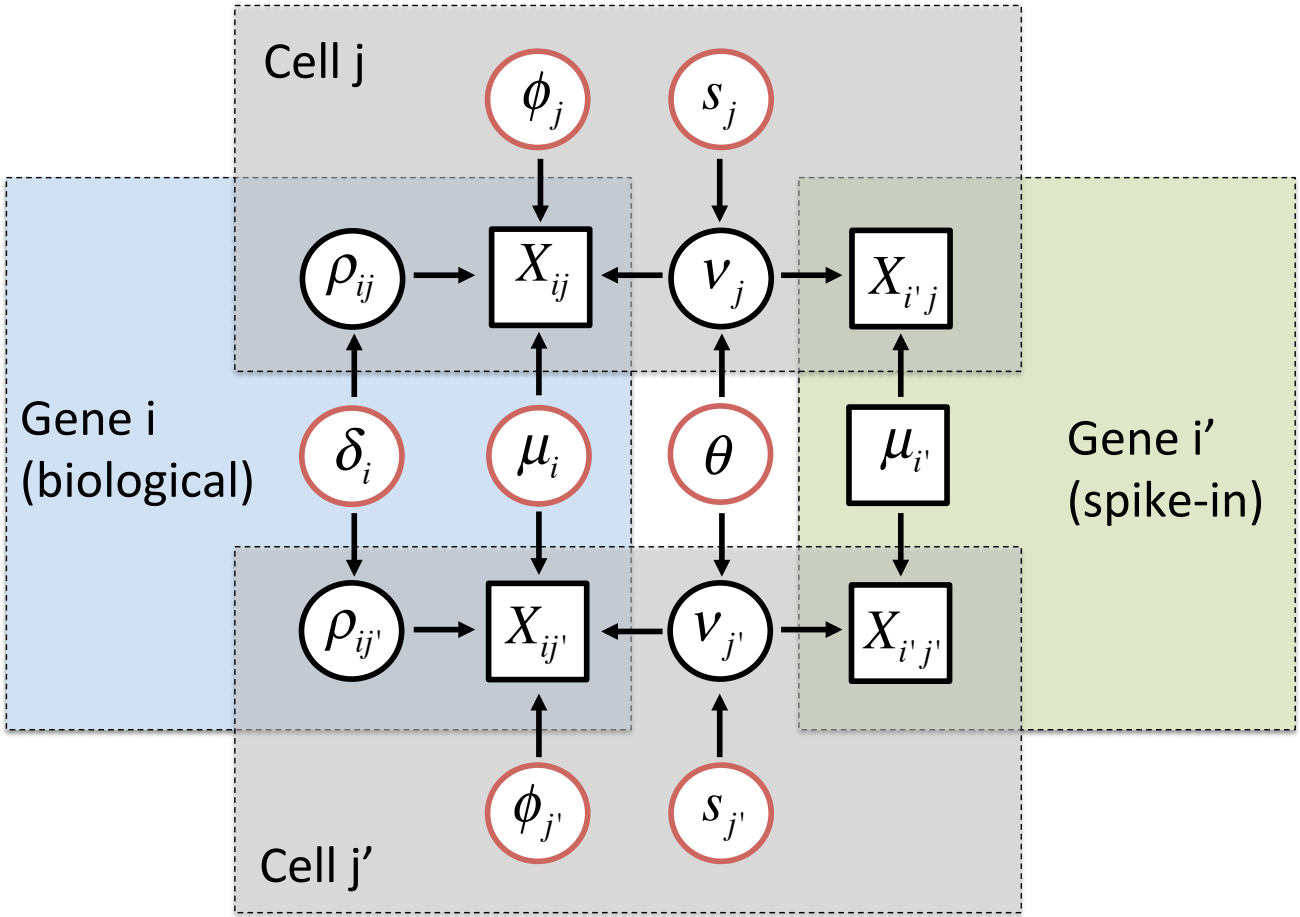


Figure 2: Graphical representation of the model implemented on BASiCS.

Running BASiCS

The `BASiCS_MCMC` function runs the MCMC sampler. It receives multiple arguments, however most of them are optional. There are 4 arguments that are required to run this function:

- `Data`: an object of class `BASiCS_Data`.
- `N`: Total number of iterations for the MCMC sampler.
- `Thin`: Thining period for the MCMC sampler.
- `Burn`: Burn-in period for the MCMC sampler.

For more information about the `BASiCS_MCMC` function use

```
help("BASiCS_MCMC", package = "BASiCS")
```

Running BASiCS

For a short run of the model use

```
MCMC_Output <- BASiCS_MCMC(Data,
                           N = 100,
                           Thin = 2,
                           Burn = 50)
```

What information is printed on the console?

Running BASiCS - Console output

- MCMC loop start message

```
-------------------------------------------------------------------
MCMC sampler has been started: 100 iterations to go.
-------------------------------------------------------------------
```

- Progress report (every 2*`Thin` iterations)

```
-------------------------------------------------------------------
MCMC iteration 80 out of 100 has been completed.
-------------------------------------------------------------------
Current draws of some selected parameters are displayed below.
mu (gene 1): 113.983
delta (gene 1): 0.201707
phi (cell 1): 1.17592
s (cell 1): 0.352644
nu (cell 1): 0.264805
theta: 0.41559
-------------------------------------------------------------------
Current proposal variances for Metropolis Hastings updates (log-scale).
LSmu (gene 1): -3.99
LSdelta (gene 1): -2.99
LSphi: 11.01
LSnu (cell 1): -7.24627
LStheta: -5.99
```

Running BASiCS - Console output

- MCMC loop end message

```
------------------------------------------------------------------------
------------------------------------------------------------------------
All 100 MCMC iterations have been completed.
------------------------------------------------------------------------
------------------------------------------------------------------------
```

- Summary of acceptace rates for Metropolis-Hastings updates

```
------------------------------------------------------------------------
Please see below a summary of the overall acceptance rates.
------------------------------------------------------------------------

Minimum acceptance rate among mu[i]'s: 0.14
Average acceptance rate among mu[i]'s: 0.709282
Maximum acceptance rate among mu[i]'s: 0.98

Minimum acceptance rate among delta[i]'s: 0.48
Average acceptance rate among delta[i]'s: 0.782353
Maximum acceptance rate among delta[i]'s: 1

Acceptance rate for phi (joint): 0.36

Minimum acceptance rate among nu[j]'s: 0.18
Average acceptance rate among nu[j]'s: 0.320976
Maximum acceptance rate among nu[j]'s: 0.52

Acceptance rate for theta: 0.94
------------------------------------------------------------------------
```

Running BASiCS - Console output

- Running time

```
------------------------------------------------------------------------
MCMC running time
------------------------------------------------------------------------
   user  system elapsed
 14.829   1.268  15.854

------------------------------------------------------------------------
```

- Output summary

```
------------------------------------------------------------------------
Output
------------------------------------------------------------------------
An object of class BASiCS_Chain
 25 MCMC samples.
 Dataset contains 7895 biological genes and 41 cells (1 batch).
 Elements (slots): mu, delta, phi, s, nu and theta.
```

Running BASiCS - Optional parameters

`StoreChains`:

If `TRUE`, the slots of the generated `BASiCS_Chain` object are stored in separate `.txt` files (one iteration at each row) using `RunName` argument to index file names (default: `StoreChains = FALSE`)

`StoreDir`:

Directory where output files are stored (default: `StoreDir = getwd()`)

Running BASiCS - Optional parameters

`StoreChains`:

If `TRUE`, the slots of the generated `BASiCS_Chain` object are stored in separate `.txt` files (one iteration at each row) using `RunName` argument to index file names (default: `StoreChains = FALSE`)

`StoreDir`:

Directory where output files are stored (default: `StoreDir = getwd()`)

For other optional parameters refer to the documentation

Running BASiCS - Convergence of MCMC chains

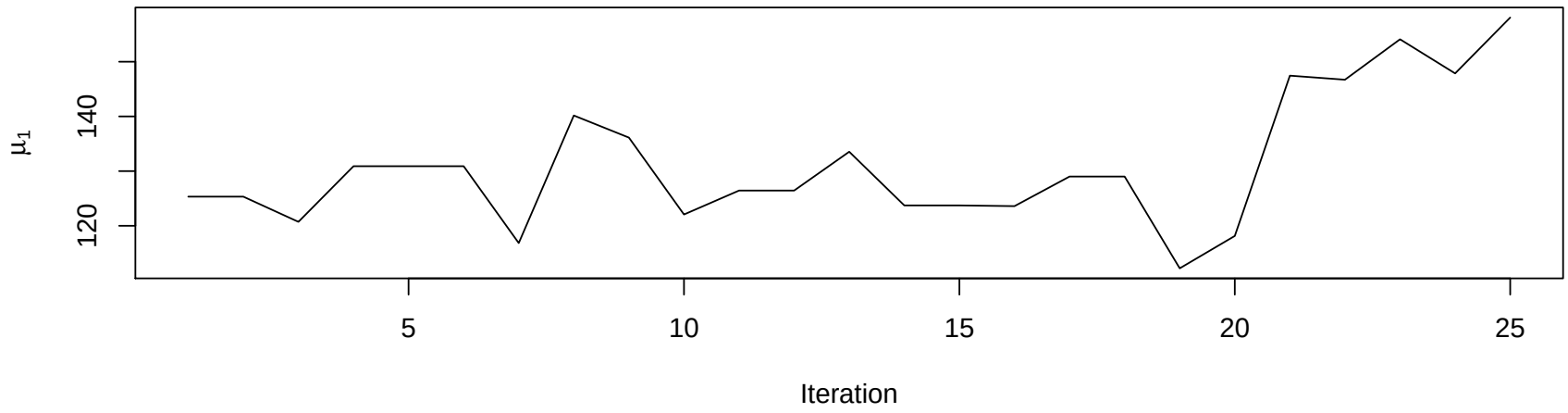
Once the MCMC algorithm has run, we need to assess whether the chains converged to their stationary distribution.

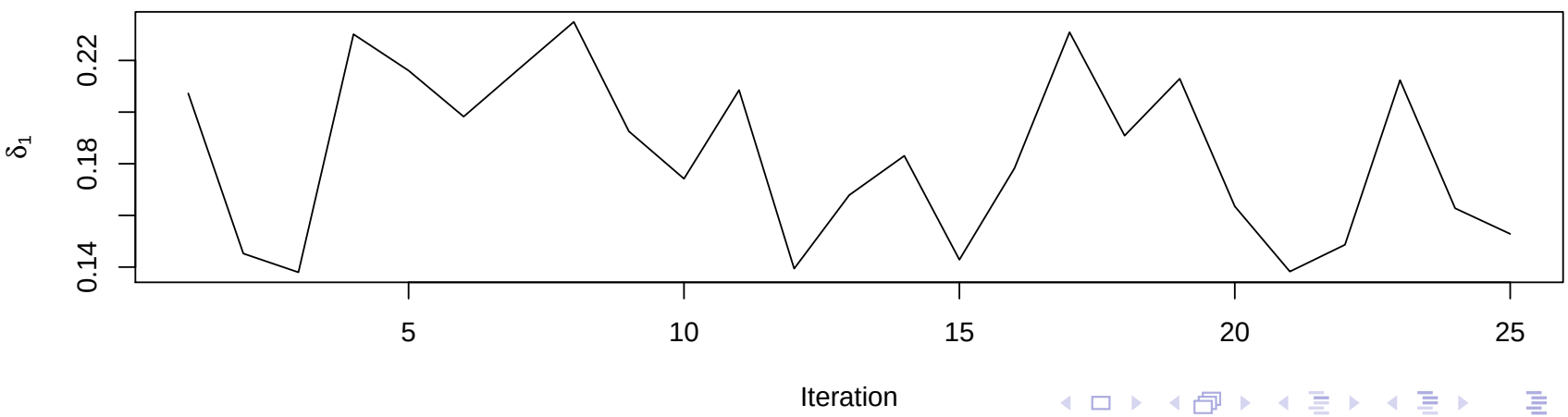
- Combination of visual inspections and test-based diagnostics
- The `R` library `coda` includes several convergence diagnostics

More details in separate slides . . .

Running BASiCS - Convergence of MCMC chains

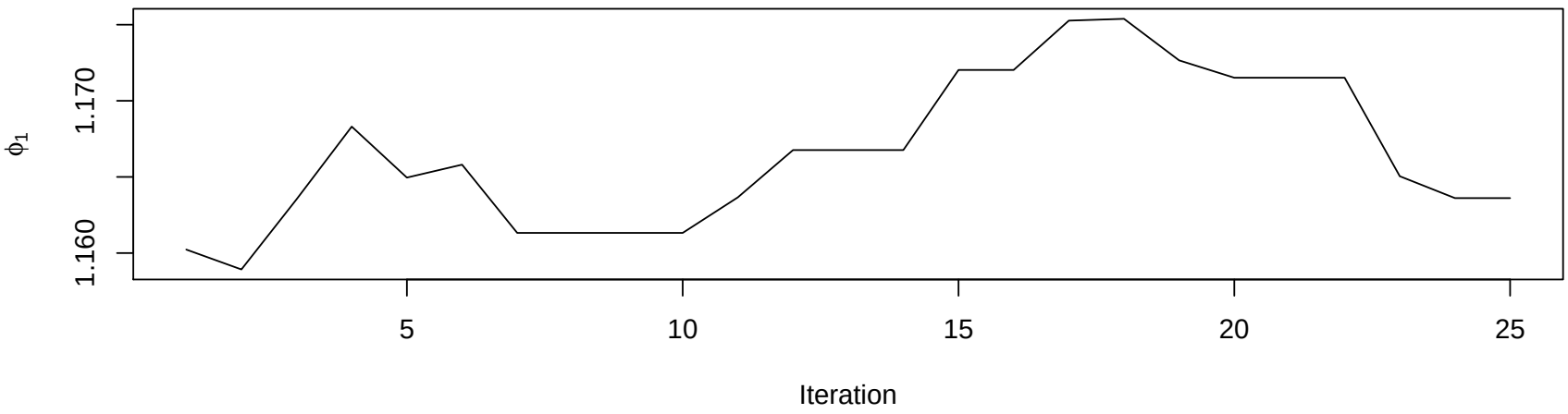
```
par(mfrow = c(2,1))
plot(MCMC_Output, Param = "mu", Gene = 1)
plot(MCMC_Output, Param = "delta", Gene = 1)
```

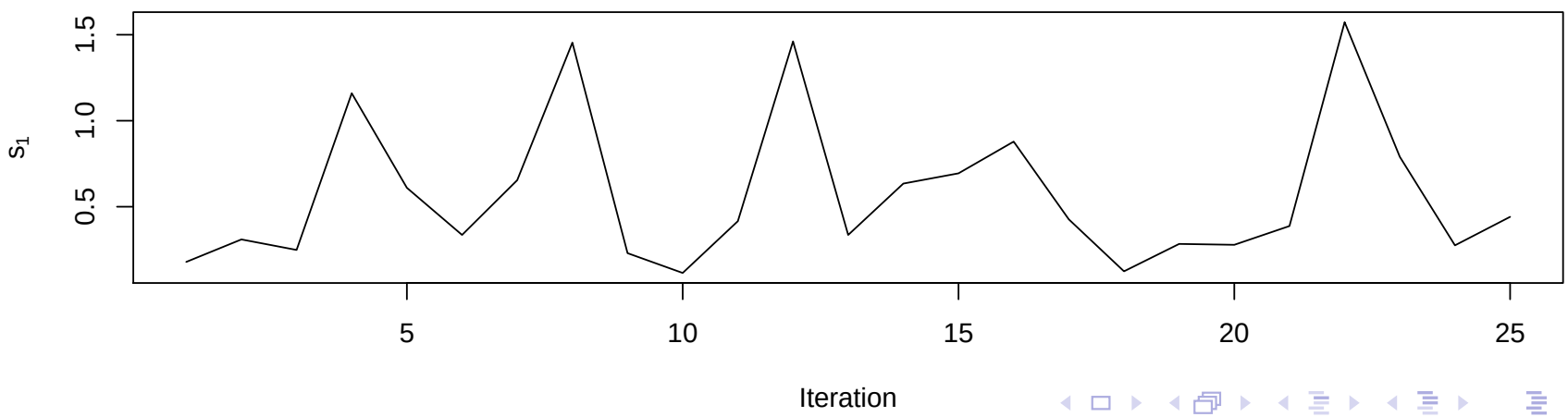




Running BASiCS - Convergence of MCMC chains

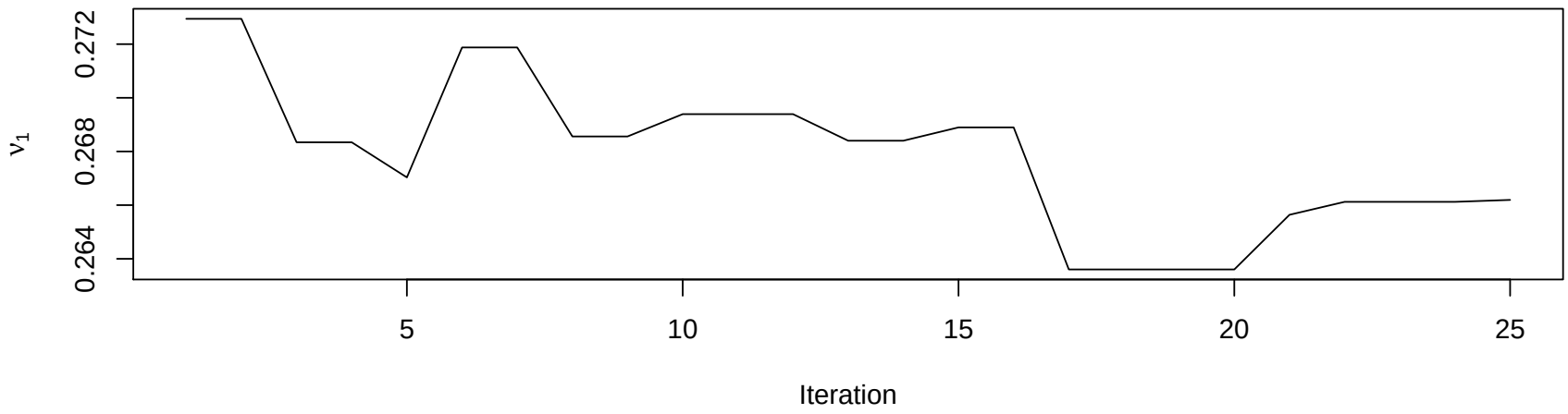
```
par(mfrow = c(2,1))
plot(MCMC_Output, Param = "phi", Cell = 1)
plot(MCMC_Output, Param = "s", Cell = 1)
```

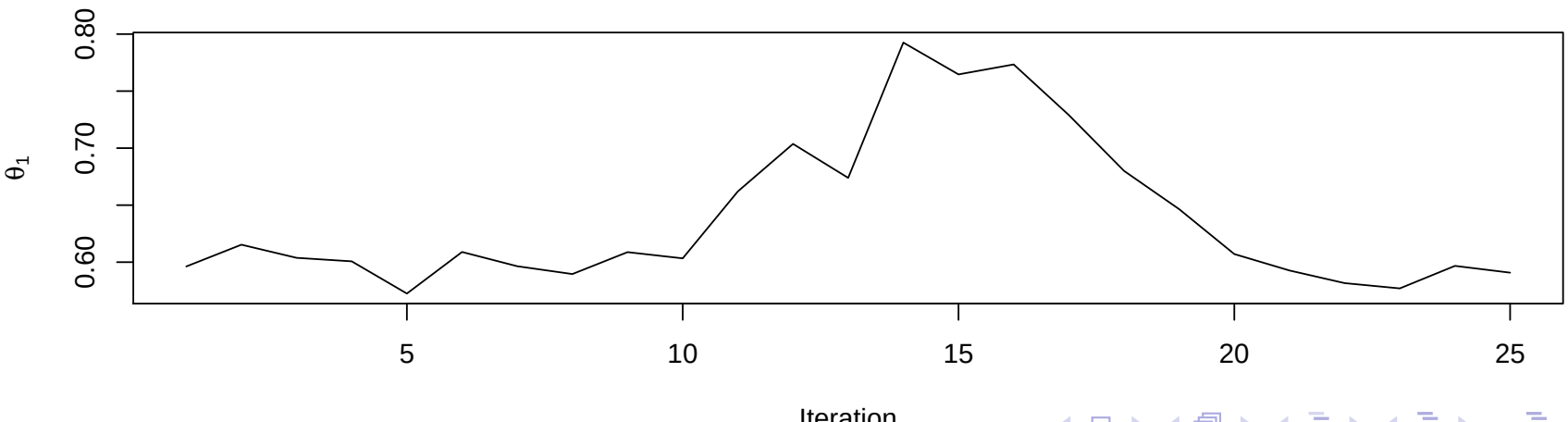




Running BASiCS - Convergence of MCMC chains

```
par(mfrow = c(2,1))
plot(MCMC_Output, Param = "nu", Cell = 1)
plot(MCMC_Output, Param = "theta")
```





Running BASiCS - Convergence of MCMC chains

Clearly, `N = 100` iterations is not enough. Before continuing, we load the provided pre-computed MCMC chains `N = 20000` (~50 min run time)

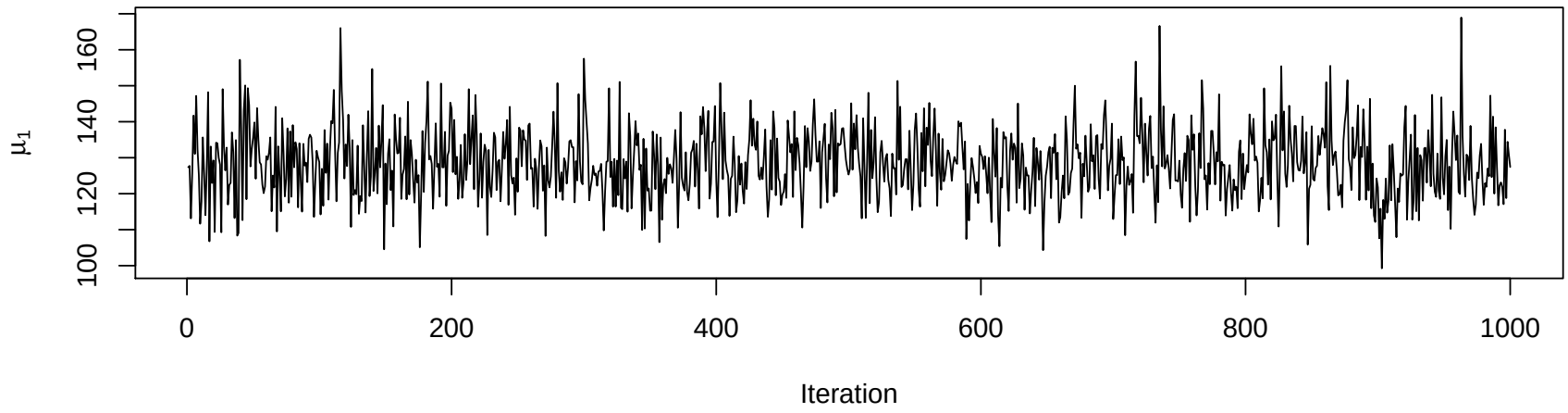
```
chains.path = "/Users/catalinavallejos/Documents/MRC/Teaching/MLPM2015/Tutorial/"
```

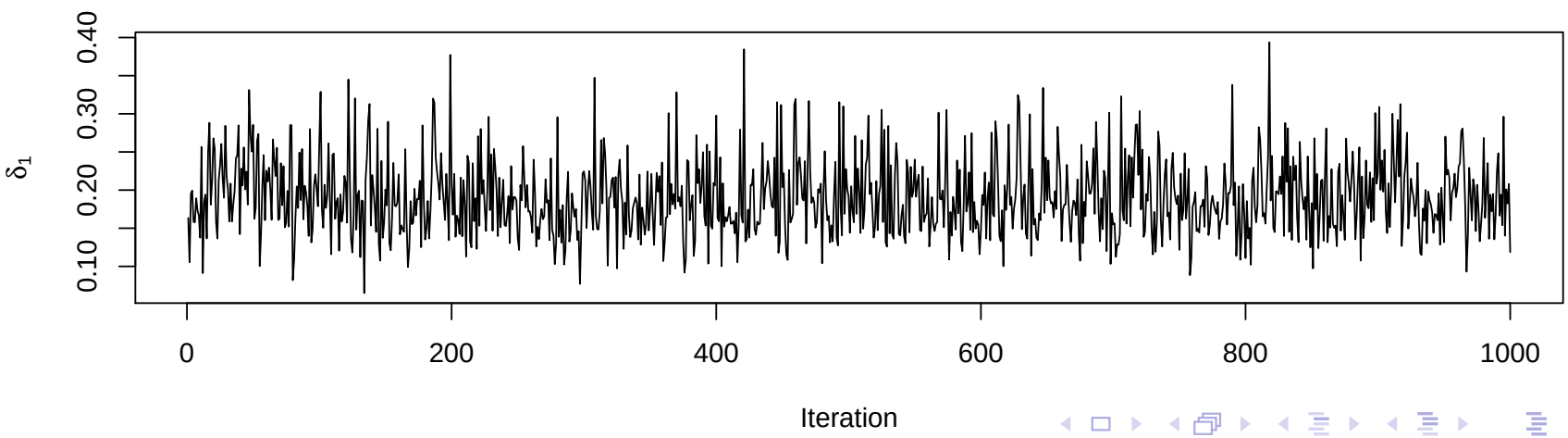
```
ChainMu=fread(file.path(chains.path,"chain_mu_BASiCS.txt"), header = TRUE)
ChainDelta=fread(file.path(chains.path,"chain_delta_BASiCS.txt"), header = TRUE)
ChainPhi=fread(file.path(chains.path,"chain_phi_BASiCS.txt"), header = TRUE)
ChainS=fread(file.path(chains.path,"chain_s_BASiCS.txt"), header = TRUE)
ChainNu=fread(file.path(chains.path,"chain_nu_BASiCS.txt"), header = TRUE)
ChainTheta=fread(file.path(chains.path,"chain_theta_BASiCS.txt"), header = TRUE)

MCMC_Output = newBASiCS_Chain(mu = as.matrix(ChainMu),
                              delta = as.matrix(ChainDelta),
                              phi = as.matrix(ChainPhi),
                              s = as.matrix(ChainS),
                              nu = as.matrix(ChainNu),
                              theta = as.matrix(ChainTheta))
```

```
## An object of class BASiCS_Chain
##  1000 MCMC samples.
##  Dataset contains 7895 biological genes and 41 cells (1 batch).
##  Elements (slots): mu, delta, phi, s, nu and theta.
```

Running BASiCS - Convergence of MCMC chains

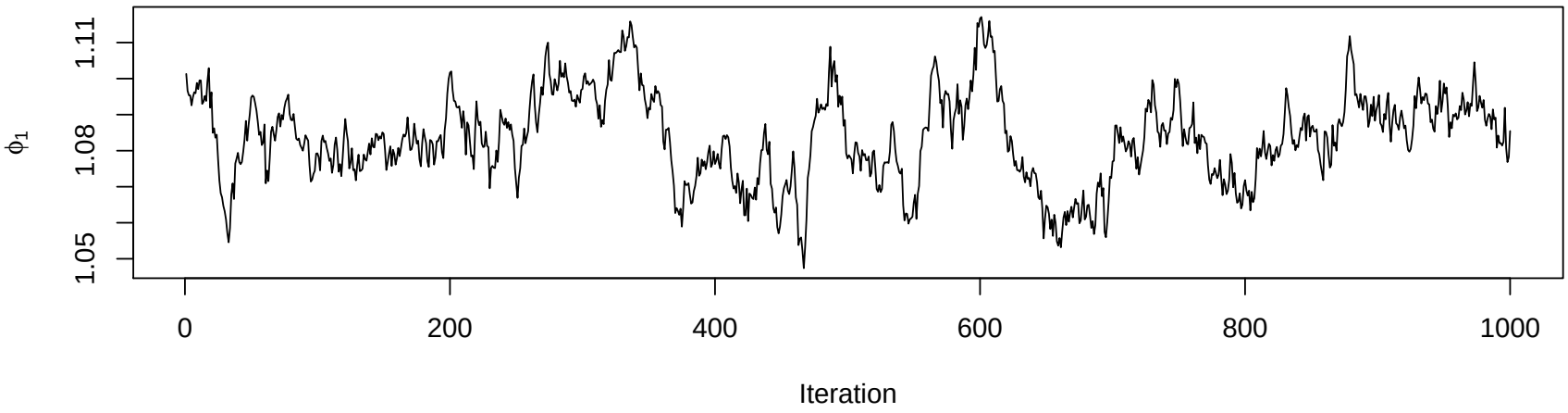
```
par(mfrow = c(2,1))
plot(MCMC_Output, Param = "mu", Gene = 1)
plot(MCMC_Output, Param = "delta", Gene = 1)
```

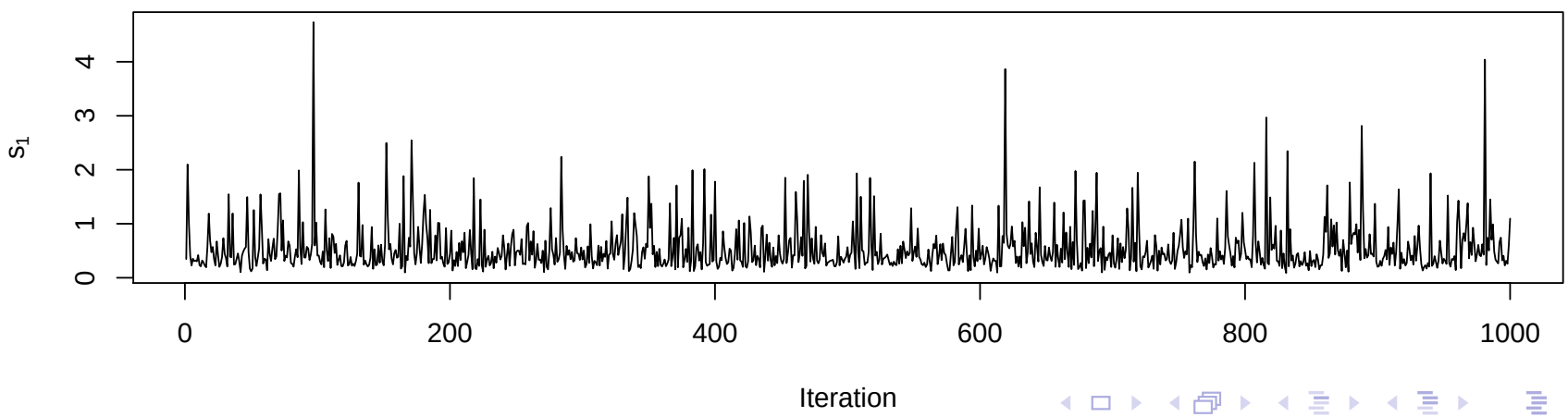




Running BASiCS - Convergence of MCMC chains

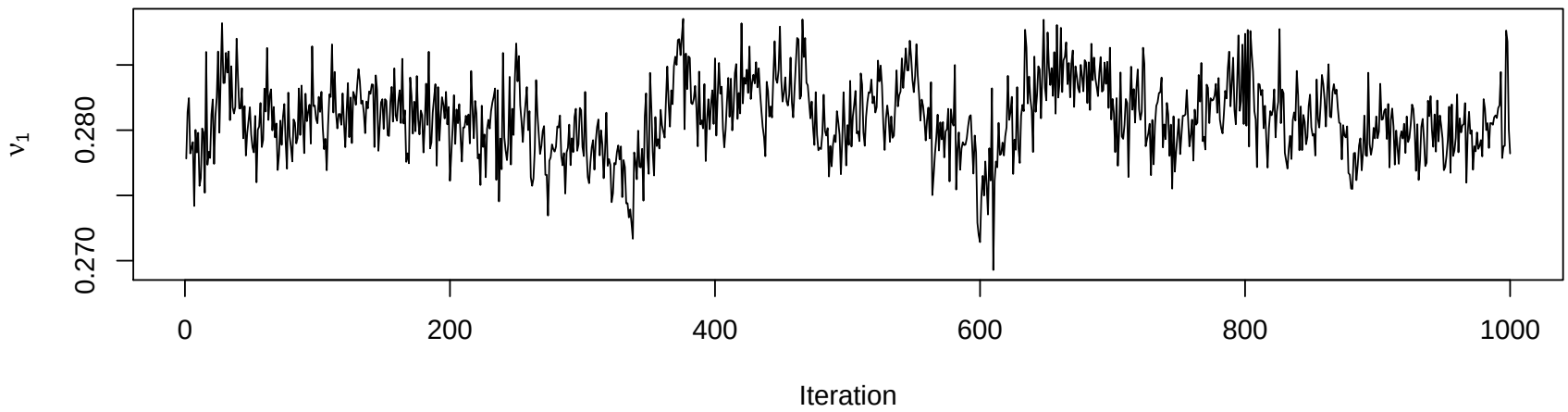
```
par(mfrow = c(2,1))
plot(MCMC_Output, Param = "phi", Cell = 1)
plot(MCMC_Output, Param = "s", Cell = 1)
```

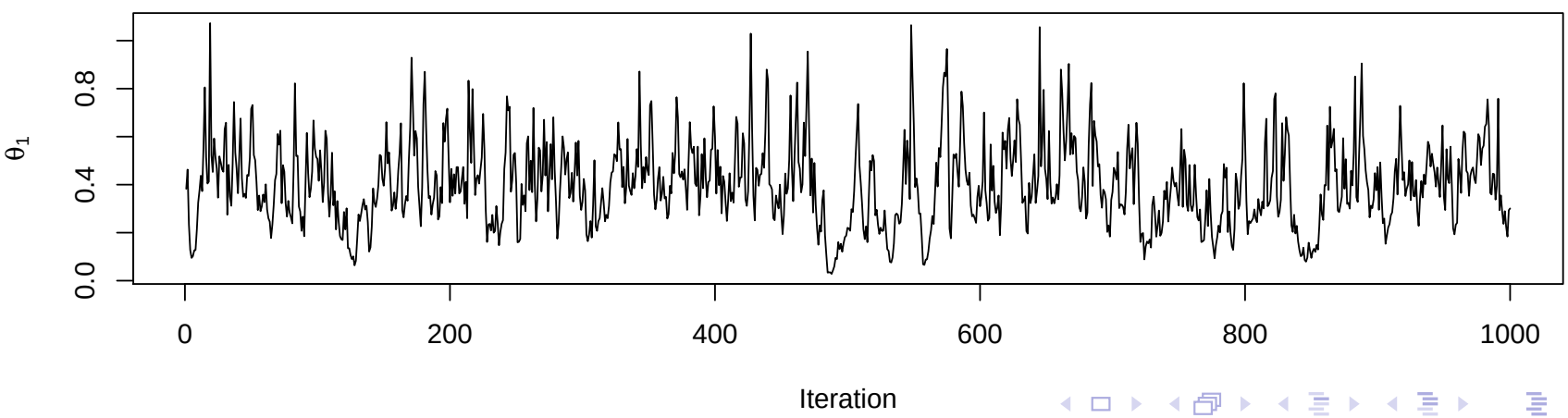




Running BASiCS - Convergence of MCMC chains

```
par(mfrow = c(2,1))
plot(MCMC_Output, Param = "nu", Cell = 1)
plot(MCMC_Output, Param = "theta")
```





Running BASiCS - Convergence of MCMC chains

DIY: Apply other convergence diagnostics to

- Randomly selected parameters
- Average values across groups of parameters

HINT: To return the MCMC chain related to the parameter indicated by `Param` (1 column per param, 1 row per iteration) use

```
displayChainBASiCS(MCMC_Output, Param = "mu")
```

Running BASiCS - Convergence of MCMC chains

DIY: Apply other convergence diagnostics to

- Randomly selected parameters
- Average values across groups of parameters

HINT: To return the MCMC chain related to the parameter indicated by `Param` (1 column per param, 1 row per iteration) use

```
displayChainBASiCS(MCMC_Output, Param = "mu")
```

Solution available in the provided R script

Post-processing of BASiCS results

Posterior summary

The MCMC chains generated by the `BASiCS_MCMC` function contain samples from the posterior distribuion of all model parameters.

How can we summarise this information?

Some commonly used summaries are:

- Posterior means, medians, modes
- Highest Posterior Density (HPD) intervals

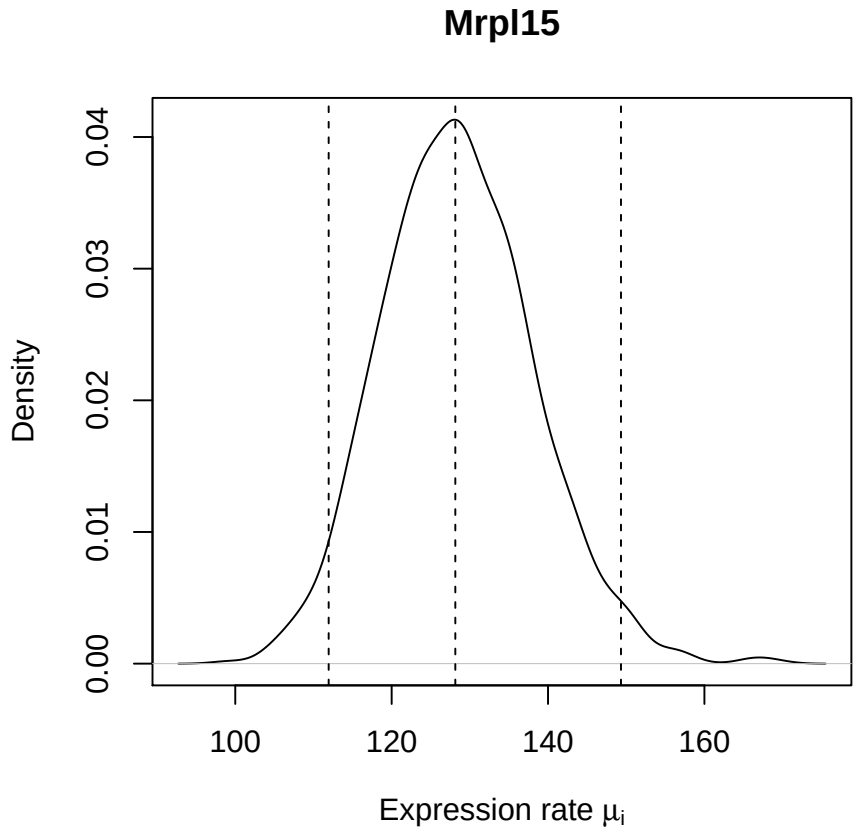
Posterior summary - Medians and 95% HPD intervals

```
MCMC_Summary <- Summary(MCMC_Output)
head(displaySummaryBASiCS(MCMC_Summary, Param = "mu"))
```

```
##                  Mu      lower     upper
## Mrpl15  128.144573 111.945012 149.33075
## Lypla1   17.550838  13.376970  22.72566
## Tcea1    70.689610  62.108196  81.29168
## Atp6v1h  11.920986   9.458517  14.46194
## Rb1cc1    7.170405   4.628787  10.01856
## Pcmtd1   17.853392  14.812363  22.06449
```

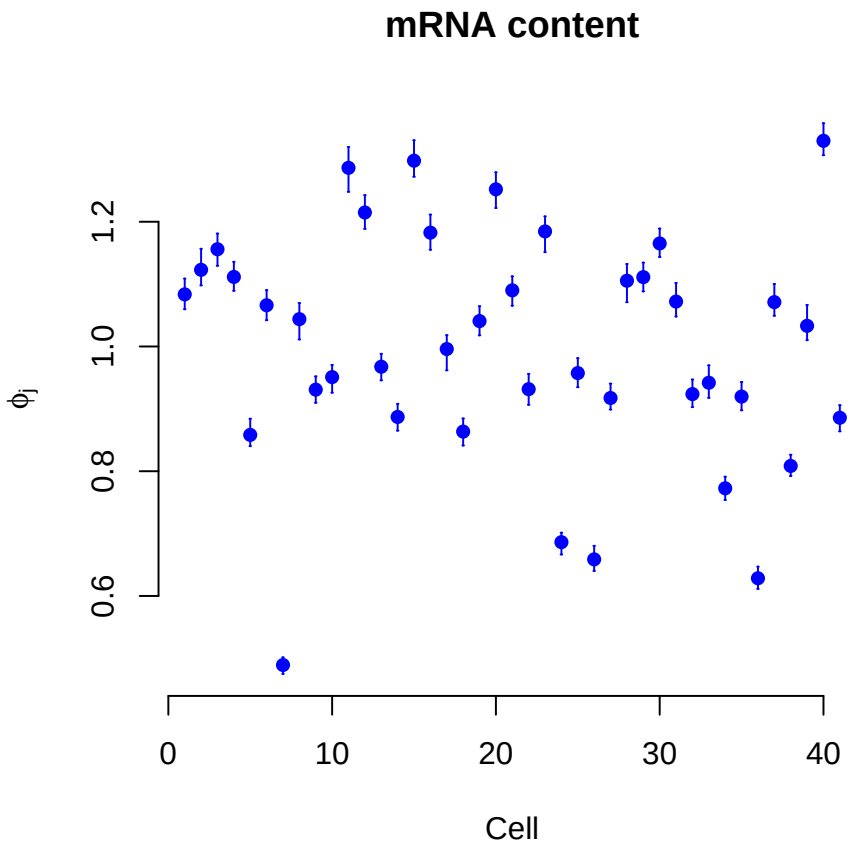
Posterior summary - Medians and 95% HPD intervals

```
plot(density(displayChainBASiCS(MCMC_Output, Param = "mu")[,1]),
     main = displayGeneNames(Data)[1],
     xlab = expression(paste("Expression rate ",mu[i])))
abline(v = displaySummaryBASiCS(MCMC_Summary, Param = "mu")[1,], lty = 2)
```



Posterior summary - Normalisation I

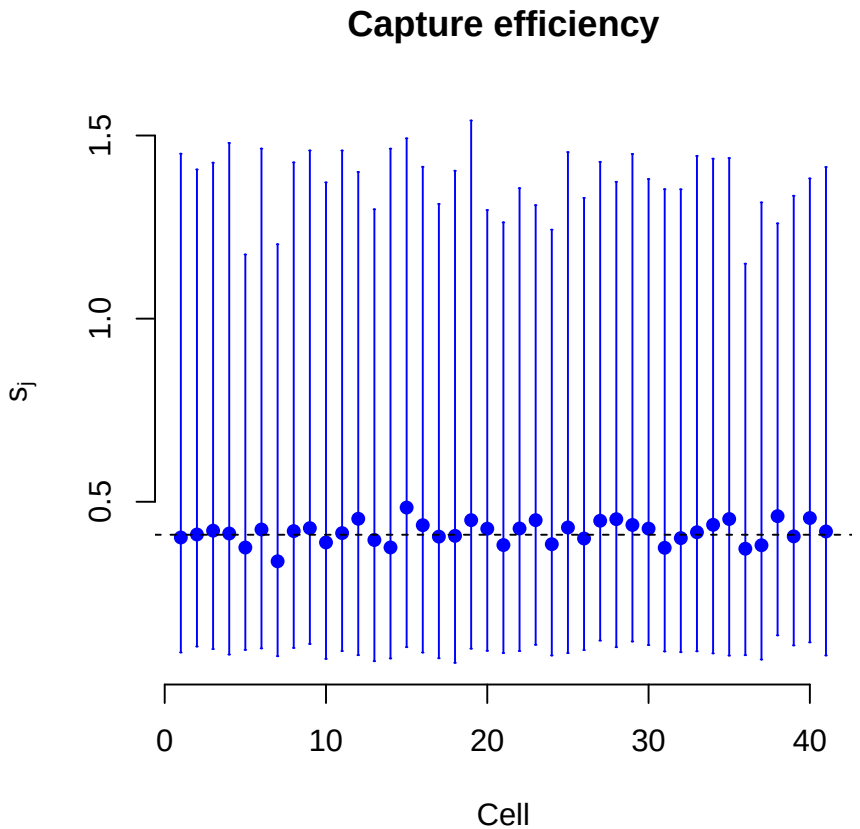
```
plot(MCMC_Summary, Param = "phi",
     main = "mRNA content")
```



- There is some heterogeneity in the total mRNA content per cell
- Yet, this is still a very homogeneous population of cells

Posterior summary - Normalisation II

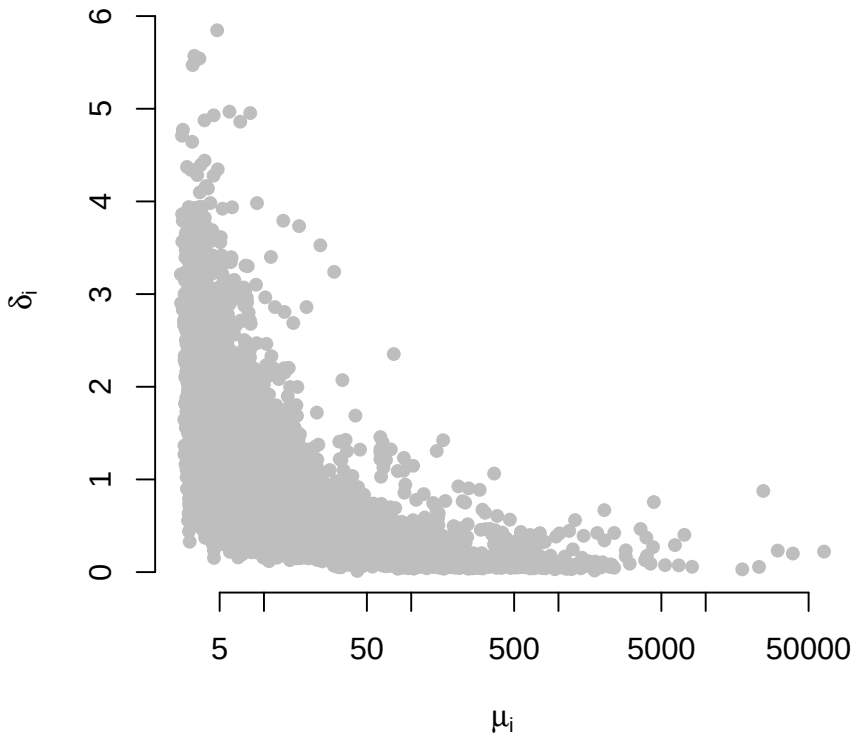
```
plot(MCMC_Summary, Param = "s",
     main = "Capture efficiency")
abline(h = 0.41, lty = 2)
```



- No amplification biases, as expected for UMI-based counts

Posterior summary - Expression vs variability

```
plot(MCMC_Summary,
     Param = "mu", Param2 = "delta",
     log = "x", col = 8)
```



- Highly expressed genes are more stable (core celullar processes)
- More variable genes concentrated on the lower end of expression

This figure changes when analysing more heterogeneous cell populations

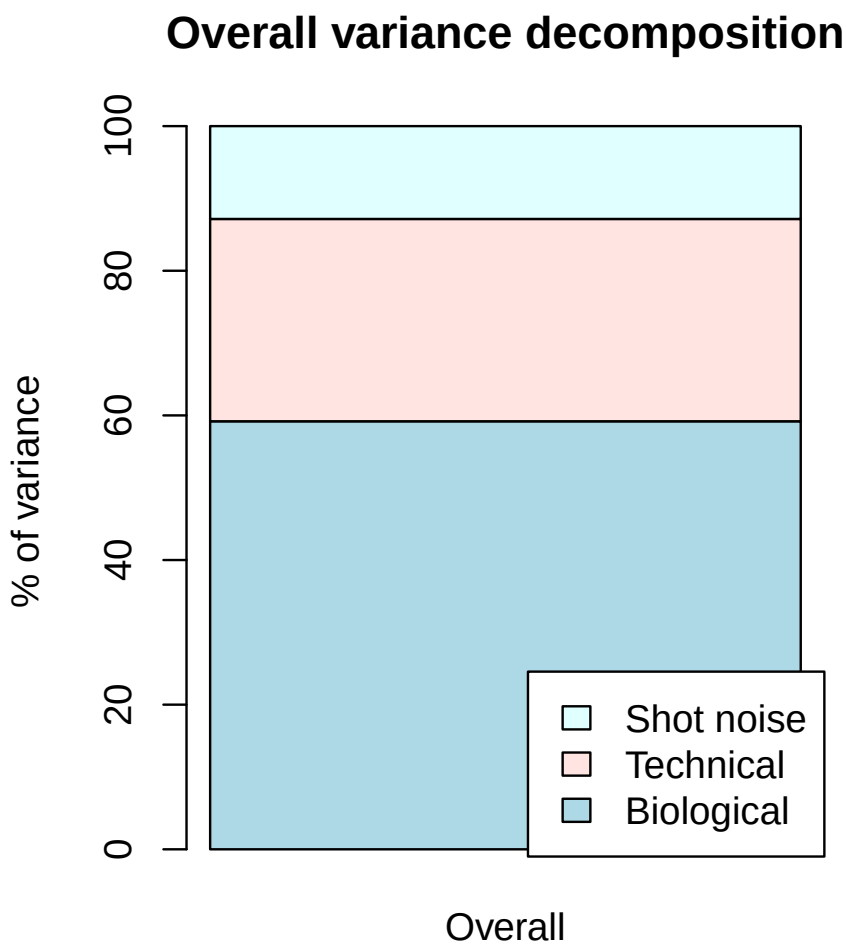
Variance decomposition

BASiCS also provides a variance decomposition for the total variability of gene expression

$$\mathbf{Var}(X_{i,j}) = \underbrace{\phi_j s_j \mu_i}_{\textbf{Shot noise}} + \underbrace{\theta(\phi_j s_j \mu_i)^2}_{\textbf{Technical}} + \underbrace{\delta_i(\theta+1)(\phi_j s_j \mu_i)^2}_{\textbf{Biological heterogeneity}}$$

Variance decomposition

```
VD = BASiCS_VarianceDecomp(Data,
                           MCMC_Output)
```



- Overall, more than 20% of the total variability is technical

Variance decomposition

We also define

$$\sigma_i = \frac{\delta_i(\theta+1)}{[(\phi s)^* \mu_i]^{-1} + \theta + \delta_i(\theta+1)}, \qquad (\phi s)^* = \underset{j \in \{1,\ldots,n\}}{\text{median}} \{\phi_j s_j\}$$

as the proportion of variability related to biological cell-to-cell heterogeneity in a typical cell

Similar expressions can be defined for the components related to technical variability and shot noise

Variance decomposition

```
head(VD)
```

```
##      GeneIndex      GeneNames BioVarGlobal TechVarGlobal ShotNoiseGlobal
## 7874      7874     r_RLTR4_Mm    0.9106224    0.07061425      0.01876333
## 4217      4217          Sprr2b    0.9086214    0.04961512      0.04176345
## 7610      7610 r_IAPEY4_I-int    0.9084450    0.06526786      0.02628719
## 5458      5458            Dqx1    0.9072444    0.07753175      0.01522389
## 5488      5488          Ccdc48    0.9042831    0.06370885      0.03200801
## 7682      7682      r_RLTR13A    0.9011824    0.04987446      0.04894314
```

```
tail(VD)
```

```
##      GeneIndex      GeneNames BioVarGlobal TechVarGlobal ShotNoiseGlobal
## 4111      4111           Naa15   0.11090273     0.8242483    0.0648490052
## 5753      5753           Rps16   0.10780109     0.8873862    0.0048127036
## 7255      7255      r_L1_Mur2    0.10329357     0.8902666    0.0064398243
## 7860      7860 r_SSU-rRNA_Hsa   0.09785355     0.9018658    0.0002806542
## 2891      2891        Hsp90ab1   0.05800949     0.9383036    0.0036868725
## 3475      3475         Mir466d   0.03972668     0.8275039    0.1327694289
```

Detection of highly and lowly variable genes

Highly Variable Genes (HVG)

For a given variance threshold γ_H, and evidence threshold α_H, BASiCS labels a gene as HVG if:

$$\pi_i^H(\gamma_H) = \mathrm{P}\left(\sigma_i > \gamma_H | \ \{\text{Data}\}\right) > \alpha_H$$

Lowly Variable Genes (LVG)

Similarly, for a given variance threshold γ_L, and evidence threshold α_L, we classify as LVG those for which:

$$\pi_i^H(\gamma_L) = \mathrm{P}\left(\sigma_i < \gamma_L | \ \{\text{Data}\}\right) > \alpha_L$$

Detection of highly and lowly variable genes

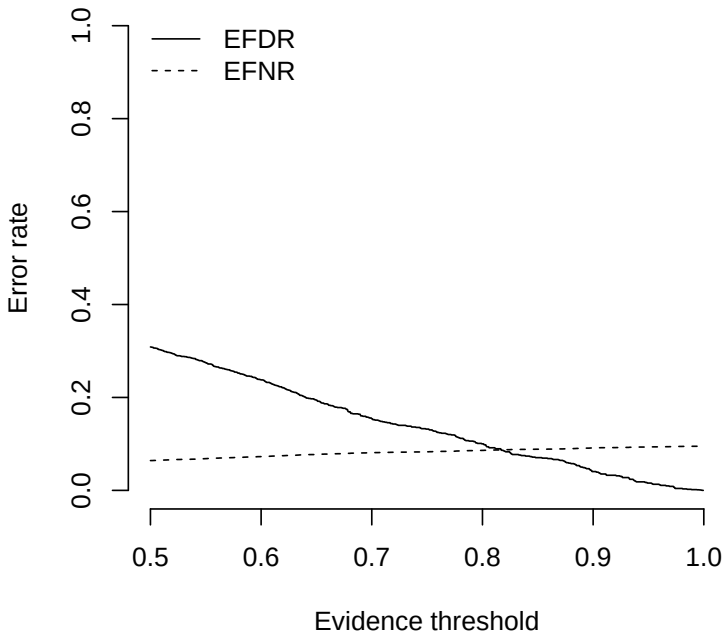
How to determine these thresholds?

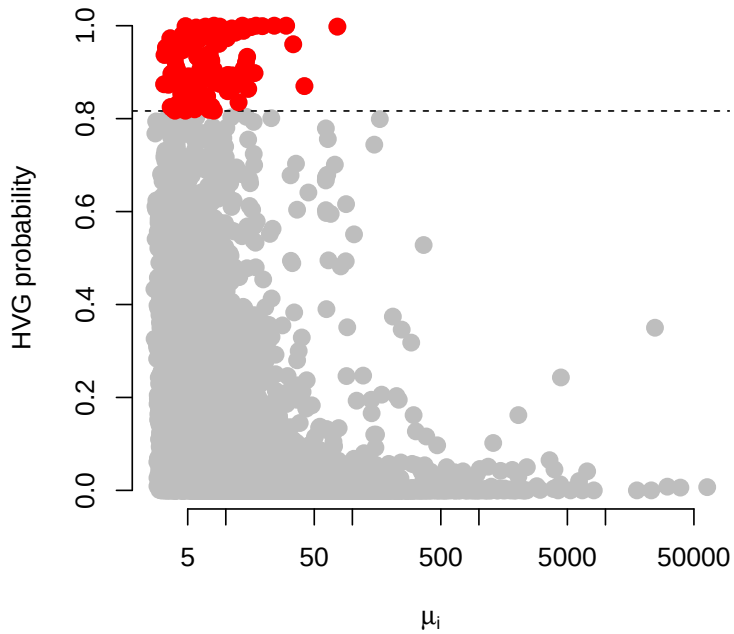
- γ_H and γ_L have a biological meaning and therefore can be chosen by an expert
- α_H and α_L are evidence thresholds and can be chosen by controlling the trade-off between EFDR and EFNR.

In the absence of pre-defined variance contribution thresholds, we can use a grid search by e.g. fixing EFDR $=$ EFNR $= 10\%$

Detection of highly variable genes

```
par(mfrow = c(1,2))
DetectHVG <- BASiCS_DetectHVG(Data, MCMC_Output,
                              VarThreshold = 0.79, Plot = TRUE)
```






```
## 90  genes classified as highly variable using:
## - Variance contribution threshold =  79 %
## - Evidence threshold =  0.8165
## - EFDR =  8.96 %
## - EFNR =  8.7 %
```

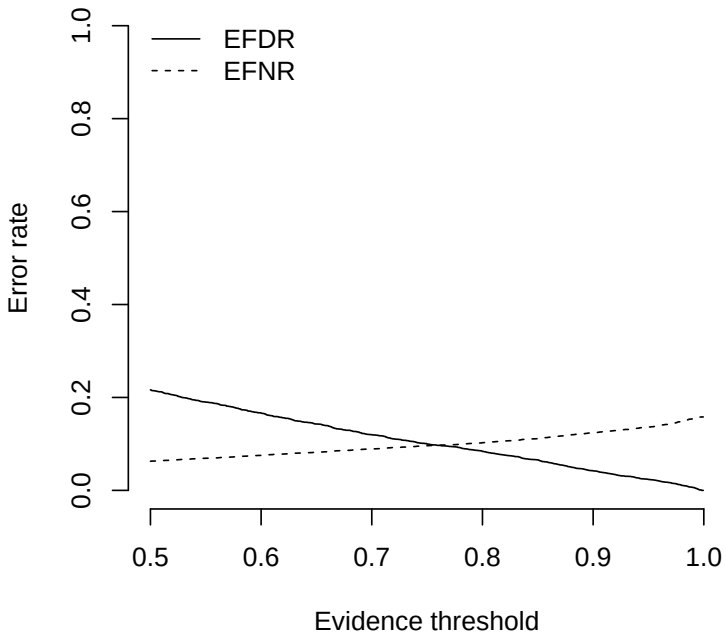
Detection of highly variable genes

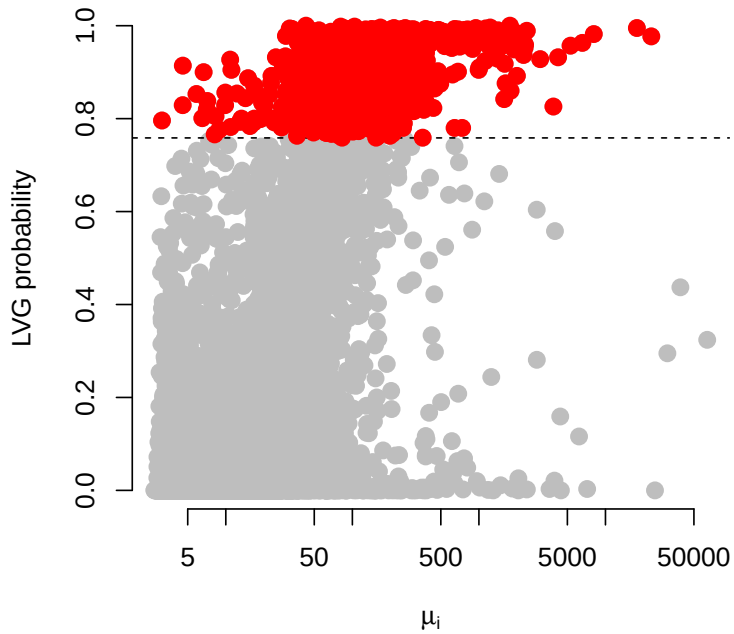
```
head(DetectHVG$Table)
```

```
##      GeneIndex      GeneNames        mu    delta     Sigma  Prob  HVG
## 4217      4217         Sprr2b  8.068237 4.953578 0.9086214 1.000 TRUE
## 5458      5458           Dqx1 29.974355 3.240290 0.9072444 1.000 TRUE
## 5488      5488         Ccdc48 13.521266 3.791891 0.9042831 1.000 TRUE
## 7610      7610 r_IAPEY4_I-int 17.314630 3.733755 0.9084450 1.000 TRUE
## 7874      7874     r_RLTR4_Mm 24.152601 3.526805 0.9106224 1.000 TRUE
## 118        118           Mreg 19.449927 2.860096 0.8859377 0.999 TRUE
```

Detection of lowly variable genes

```
par(mfrow = c(1,2))
DetectLVG <- BASiCS_DetectLVG(Data, MCMC_Output,
                              VarThreshold = 0.41, Plot = TRUE)
```






```
## 601  genes classified as lowly variable using:
## - Variance contribution threshold =  41 %
## - Evidence threshold =  0.7585
## - EFDR =  9.76 %
## - EFNR =  9.69 %
```

Detection of lowly variable genes

```
head(DetectLVG$Table)
```

```
##       GeneIndex GeneNames         mu      delta      Sigma  Prob  LVG
## 2891       2891  Hsp90ab1 1759.51170 0.01695337 0.05800949 1.000 TRUE
## 3475       3475   Mir466d   43.03774 0.01276304 0.03972668 1.000 TRUE
## 420         420    Gm6251  104.06384 0.03790647 0.11636563 0.998 TRUE
## 4111       4111     Naa15   80.96550 0.03664674 0.11090273 0.998 TRUE
## 5753       5753     Rps16 1227.70143 0.03211328 0.10780109 0.996 TRUE
## 7173       7173    Rpl36a  166.65200 0.03641563 0.11472605 0.996 TRUE
```

And beyond . . .

And beyond . . .

There are many aspects of scRNA-seq and downstream analysis that were not covered by this tutorial

And beyond . . . batch effects

The current implementation of BASiCS also allows batch-effect correction using batch-specific values of θ. To use this feature, the `BASiCS_Data` object requires an additional element. For example

```
Data = newBASiCS_Data(Counts = CountsQC,
                      Tech = TechQC,
                      SpikeInfo = SpikeInfoQC,
                      BatchInfo = c(rep(1, times = 20), rep(2, times = 21)))
```

```
## An object of class BASiCS_Data
##  Dataset contains 7941 genes (7895 biological and 46 technical) and 41 cells.
##  Elements (slots): Counts, Tech, SpikeInput, GeneNames and BatchInfo.
##  The data contains 2 batches.
##
## 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
```

And beyond . . . clustering

Clustering cells according to their expression profile is widely applied to scRNA-seq datasets

BASiCS will soon include clustering as a built-in analysis tool

In the meantime, you can extract **normalised and denoised** expression rates to be used for the clustering (and other downstream analyses). For each gene i and cell j, these are defined as

$$\text{DR}_{ij} = \mu_i \rho_{ij}$$

And beyond . . . clustering

Posterior estimates for these quantities can be obtained using

```
DR = BASiCS_DenoisedRates(Data, MCMC_Output)
```

```
## [1] "This calculation requires a loop across the 1000 MCMC iterations"
## [1] "Please be patient ... "
##
## [1] "To see a progress report use PrintProgress = TRUE"
```

And beyond . . . clustering

```
head(round(DR[, 1:10],1), n = 10)
```

```
##           B01   C01   D01   C02  A03   A04  B04   G04   H04   A05
## Mrpl15  180.3 195.9 103.1 163.1 86.8 175.7 82.3 130.7 116.6 125.2
## Lypla1   19.2   8.4  17.6   6.6 32.6   6.7 13.2  32.6  17.1   7.5
## Tcea1    46.6  86.5  40.4  74.9 95.3 123.1 57.0  66.6 109.6  67.1
## Atp6v1h  13.9  19.8  10.4   9.2 12.4   7.9 11.3  13.6  20.1  13.1
## Rb1cc1    4.6   4.1   4.2   2.2  8.2   4.4 12.1   2.2   2.4  24.4
## Pcmtd1   20.5  17.4  17.7  21.3 19.6  11.7 33.8  21.6  19.1  12.6  ...
## Rrs1     18.5  34.8  14.5  12.5  3.7   2.9  5.9  30.1  13.6   6.1
## Vcpip1    9.6  13.1  31.9   4.3  2.6  19.0 18.8   2.0   2.1   5.0
## Snhg6    46.4  80.3   9.1 101.2 62.4  76.9 40.3  52.4  30.3  78.4
## Cops5    26.9  41.9  24.6  11.1 20.8  30.8 13.1  23.4  17.2  29.2
```

Questions?