# EBSeq: An R package for differential expression analysis using RNA-seq data

Ning Leng, John A. Dawson, Christina Kendziorski

May 15, 2012

## Contents

# 1 Introduction

EBSeq may be used to identify differentially expressed (DE) genes and isoforms in an RNA-Seq experiment. As detailed in Leng *et al.*, 2012 [3], EBSeq is an empirical Bayesian approach that models a number of features observed in RNA-seq data. Importantly, for isoform level inference, EBSeq directly accommodates isoform expression estimation uncertainty by modeling the differential variability observed in distinct groups of isoforms. Consider Figure 1, where we have plotted variance against mean for all isoforms using RNA-Seq expression data from Leng *et al.*, 2012 [3]. Also shown is the fit within three sub-groups of isoforms defined by the number of constituent isoforms of the parent gene. An isoform of gene $g$ is assigned to the $N_g = k$ group, where $k = 1, 2, 3$, if the total number of isoforms from gene $g$ is $k$ (the $N_g = 3$ group contains all isoforms from genes having 3 or more isoforms). As shown in Figure 1, there is decreased variability in the $N_g = 1$ group, but increased variability in the others, due to the relative increase in uncertainty inherent in estimating isoform expression when multiple isoforms of a given gene are present. If this structure is not accommodated, there is reduced power for identifying isoforms in the $N_g = 1$ group (since the true variances in that group are lower, on average, than that derived from the full collection of isoforms) as well as increased false discoveries in the $N_g = 2$ and $N_g = 3$ groups (since the true variances are higher, on average, than those derived from the full collection). EBSeq directly models differential variability as a function of $N_g$ providing a powerful approach for isoform level inference. As shown in Leng *et al.*, 2012 [3], the model is also useful for identifying DE genes. We will briefly detail the model in Section 2 and then describe the flow of analysis in Section 3 for both isoform and gene-level inference.

# 2 The Model

## 2.1 Two conditions

We let $X_{g_i}^{C1} = X_{g_i,1}, X_{g_i,2}, ..., X_{g_i,S_1}$ denote data from condition 1 and $X_{g_i}^{C2} = X_{g_i,(S_1+1)}, X_{g_i,(S_1+2)}, ..., X_{g_i,S}$ data from condition 2. We assume that counts within condition $C$ are distributed as Negative Binomial: $X_{g_i,s}^{C} | r_{g_i,s}, q_{g_i}^{C} \sim NB(r_{g_i,s}, q_{g_i}^{C})$ where

$$P(X_{g_i,s}|r_{g_i,s}, q_{g_i}^{C}) = \binom{X_{g_i,s} + r_{g_i,s} - 1}{X_{g_i,s}} (1 - q_{g_i}^{C})^{X_{g_i,s}} (q_{g_i}^{C})^{r_{g_i,s}} \quad (1)$$

and $\mu_{g_i,s}^{C} = r_{g_i,s}(1 - q_{g_i}^{C})/q_{g_i}^{C}$; $(\sigma_{g_i,s}^{C})^2 = r_{g_i,s}(1 - q_{g_i}^{C})/(q_{g_i}^{C})^2$.

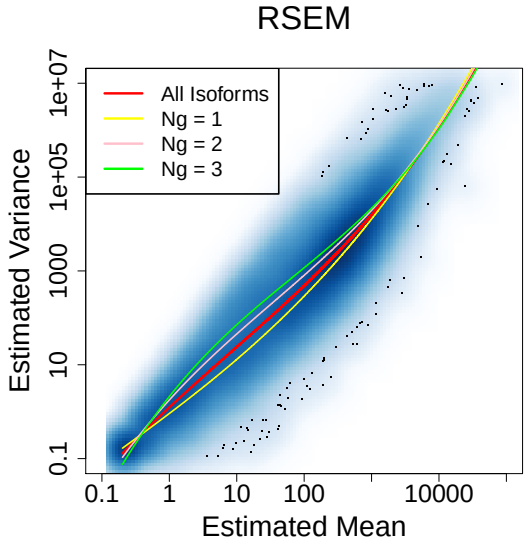


Figure 1: Empirical variance vs. mean for each isoform profiled in the mammary carcinoma experiment detailed in the Case Study section of Leng *et al.*, 2012 [3]. A spline fit to all isoforms is shown in red with splines fit within the $N_g = 1$, $N_g = 2$, and $N_g = 3$ isoform groups shown in yellow, pink, and green, respectively.

We assume a prior distribution on $q^C_{g_i}$: $q^C_{g_i}|\alpha, \beta^{N_g} \sim Beta(\alpha, \beta^{N_g})$. The hyperparameter $\alpha$ is shared by all the isoforms and $\beta^{N_g}$ is $N_g$ specific (note this is an index, not a power). We further assume that $r_{g_i,s} = r_{g_i,0} l_s$, where $r_{g_i,0}$ is an isoform specific parameter common across conditions and $r_{g_i,s}$ depends on it through the sample-specific normalization factor $l_s$. Of interest in this two group comparison is distinguishing between two cases, or what we will refer to subsequently as two patterns of expression, namely equivalent expression (EE) and differential expression (DE):

$$H_0 \text{ (EE)} : q^{C1}_{g_i} = q^{C2}_{g_i} \text{ vs } H_1 \text{ (DE)} : q^{C1}_{g_i} \neq q^{C2}_{g_i}.$$

Under the null hypothesis (EE), the data $X^{C1,C2}_{g_i} = X^{C1}_{g_i}, X^{C2}_{g_i}$ arises from the prior predictive distribution $f_0^{N_g}(X^{C1,C2}_{g_i})$:

$$f_0^{N_g}(X^{C1,C2}_{g_i}) = \left[\prod_{s=1}^{S} \binom{X_{g_i,s} + r_{g_i,s} - 1}{X_{g_i,s}}\right] \frac{Beta(\alpha + \sum_{s=1}^{S} r_{g_i,s}, \beta^{N_g} + \sum_{s=1}^{S} X_{g_i,s})}{Beta(\alpha, \beta^{N_g})} \tag{2}$$

Alternatively (in a DE scenario), $X^{C1,C2}_{g_i}$ follows the prior predictive distribution $f_1^{N_g}(X^{C1,C2}_{g_i})$:

$$f_1^{N_g}(X^{C1,C2}_{g_i}) = f_0^{N_g}(X^{C1}_{g_i}) f_0^{N_g}(X^{C2}_{g_i}) \tag{3}$$

Let the latent variable $Z_{g_i}$ be defined so that $Z_{g_i} = 1$ indicates that isoform $g_i$ is DE and $Z_{g_i} = 0$ indicates isoform $g_i$ is EE, and $Z_{g_i} \sim Bernoulli(p)$. Then, the marginal distribution of $X_{g_i}^{C1,C2}$ and $Z_{g_i}$ is:

$$(1-p)f_0^{N_g}(X_{g_i}^{C1,C2}) + pf_1^{N_g}(X_{g_i}^{C1,C2}) \tag{4}$$

The posterior probability of being DE at isoform $g_i$ is obtained by Bayes' rule:

$$\frac{pf_1^{N_g}(X_{g_i}^{C1,C2})}{(1-p)f_0^{N_g}(X_{g_i}^{C1,C2}) + pf_1^{N_g}(X_{g_i}^{C1,C2})} \tag{5}$$

## 2.2 More than two conditions

EBSeq naturally accommodates multiple condition comparisons. For example, in a study with 3 conditions, there are K=5 possible expression patterns (P1,...,P5), or ways in which latent levels of expression may vary across conditions:

$$\begin{aligned}
&\text{P1: } q_{g_i}^{C1} = q_{g_i}^{C2} = q_{g_i}^{C3} \\
&\text{P2: } q_{g_i}^{C1} = q_{g_i}^{C2} \neq q_{g_i}^{C3} \\
&\text{P3: } q_{g_i}^{C1} = q_{g_i}^{C3} \neq q_{g_i}^{C2} \\
&\text{P4: } q_{g_i}^{C1} \neq q_{g_i}^{C2} = q_{g_i}^{C3} \\
&\text{P5: } q_{g_i}^{C1} \neq q_{g_i}^{C2} \neq q_{g_i}^{C3} \text{ and } q_{g_i}^{C1} \neq q_{g_i}^{C3}
\end{aligned}$$

The prior predictive distributions for these are given, respectively, by:

$$\begin{aligned}
g_1^{N_g}(X_{g_i}^{C1,C2,C3}) &= f_0^{N_g}(X_{g_i}^{C1,C2,C3}) \\
g_2^{N_g}(X_{g_i}^{C1,C2,C3}) &= f_0^{N_g}(X_{g_i}^{C1,C2})f_0^{N_g}(X_{g_i}^{C3}) \\
g_3^{N_g}(X_{g_i}^{C1,C2,C3}) &= f_0^{N_g}(X_{g_i}^{C1,C3})f_0^{N_g}(X_{g_i}^{C2}) \\
g_4^{N_g}(X_{g_i}^{C1,C2,C3}) &= f_0^{N_g}(X_{g_i}^{C1})f_0^{N_g}(X_{g_i}^{C2,C3}) \\
g_5^{N_g}(X_{g_i}^{C1,C2,C3}) &= f_0^{N_g}(X_{g_i}^{C1})f_0^{N_g}(X_{g_i}^{C2})f_0^{N_g}(X_{g_i}^{C3})
\end{aligned}$$

where $f_0^{N_g}$ is the same as in equation 2. Then the marginal distribution in equation 4 becomes:

$$\sum_{k=1}^{5} p_k g_k^{N_g}(X_{g_i}^{C1,C2,C3}) \tag{6}$$

where $\sum_{k=1}^{5} p_k = 1$. Thus, the posterior probability of isoform $g_i$ coming from pattern $K$ is readily obtained by:

$$\frac{p_K g_K^{N_g}(X_{g_i}^{C1,C2,C3})}{\sum_{k=1}^{5} p_k g_k^{N_g}(X_{g_i}^{C1,C2,C3})} \tag{7}$$

# 3 Quick Start

Before analysis can proceed, the EBSeq package must be loaded into the working space:

```
> library(EBSeq)
```

## 3.1 Gene Level DE Analysis (Two Conditions)

### 3.1.1 Required input

**Data**: The object `Data` should be a $G - by - S$ matrix containing the expression values for each gene and each lane (sample), where $G$ is the number of genes and $S$ is the number of lanes. These values should exhibit raw counts, without normalization across samples. Counts of this nature may be obtained from RSEM [4], Cufflinks [7] or other such pre-processing approaches.

**Conditions**: The object `Conditions` should be a Factor vector of length $S$ that indicates to which condition each sample belongs. For example, if there are two conditions and three samples in each, $S = 6$ and `Conditions` may be given by
`as.factor(c("C1","C1","C1","C2","C2","C2"))`

The object `GeneMat` is a simulated data matrix containing 10,000 rows of genes and 10 columns of samples (a function to simulate such a dataset is detailed in Section 4.1.1). The genes are named `Gene_1, Gene_2 ...`

```
> data(GeneMat)
> str(GeneMat)
```

```
 num [1:10000, 1:10] 1879 24 3291 97 485 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:10000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
  ..$ : NULL
```

### 3.1.2 Library size factor

As detailed in Section 2, EBSeq requires the library size factor $l_s$ for each sample $s$. Here, $l_s$ may be obtained via the function `MedianNorm`, which reproduces the median normalization approach in DESeq [1].

```
> Sizes = MedianNorm(GeneMat)
```

If quantile normalization is preferred, $l_s$ may be obtained via the function `QuantileNorm`.

### 3.1.3 Running EBSeq on gene counts

The function `EBTest` is used to detect DE genes. For gene-level data, we don't need to specify the parameter `NgVector` since there are no differences in $N_g$ structure among the different genes. Here, we simulated the first five lanes to be in condition 1 and the other five in condition 2, so define:

`Conditions=as.factor(rep(c("C1","C2"),each=5))`

`sizeFactors` is used to define the library size factor of each lane. It could be obtained by summing up the total number of reads within each lane, Median Normalization [1], scaling normalization [5], or some other such approach. These in hand, we run the EM algorithm, setting the number of iterations to five via `maxround=5` for demonstration purposes. However, we note that in practice, additional iterations may be required. Convergence should always be checked (see Section 4.1.3 for details). Please note this may take several minutes:

```
> EBOut = EBTest(Data = GeneMat, Conditions = as.factor(rep(c("C1",
+     "C2"), each = 5)), sizeFactors = Sizes, maxround = 5)

iteration 1 done
time 23.978
iteration 2 done
time 13.337
iteration 3 done
time 12.955
iteration 4 done
time 12.447
iteration 5 done
time 11.179
```

The posterior probabilities of being DE are obtained as follows, where `PP` is a matrix containing the posterior probabilities of being EE or DE for each of the 10,000 simulated genes:

```
> PP = GetPPMat(EBOut)
> str(PP)

 num [1:10000, 1:2] 0 0 0 0 0 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:10000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
  ..$ : chr [1:2] "PPEE" "PPDE"

> head(PP)

               PPEE PPDE
Gene_1 0.000000e+00    1
Gene_2 0.000000e+00    1
Gene_3 0.000000e+00    1
Gene_4 0.000000e+00    1
Gene_5 0.000000e+00    1
Gene_6 3.289292e-10    1
```

The matrix PP contains two columns PPEE and PPDE, corresponding to the posterior probabilities of being EE or DE for each gene. PP may be used to form an FDR-controlled list of DE genes with a target FDR of 0.05 as follows:

```
> DEfound = rownames(PP)[which(PP[, "PPDE"] >= 0.95)]
> str(DEfound)

 chr [1:991] "Gene_1" "Gene_2" "Gene_3" "Gene_4" "Gene_5" ...
```

EBSeq found 991 DE genes in total with target FDR 0.05.

## 3.2 Isoform Level DE Analysis (Two Conditions)

### 3.2.1 Required inputs

**Data**: The object Data should be a $I - by - S$ matrix containing the expression values for each isoform and each lane, where $I$ is the number of isoforms and $S$ is the number of lanes. Again, these values should exhibit raw data, without normalization across samples.

**Conditions**: The object Conditions should be a vector with length $S$ to indicate the condition of each sample.

**IsoformNames**: The object IsoformNames should be a vector with length $I$ to indicate the isoform names.

**IsosGeneNames**: The object IsosGeneNames should be a vector with length $I$ to indicate the gene name of each isoform. (in the same order as IsoformNames.)

IsoList contains 6,000 simulated isoforms. In which IsoList$IsoMat is a data matrix containing 6,000 rows of isoforms and 10 columns of samples; IsoList$IsoNames contains the isoform names; IsoList$IsosGeneNames contains the names of the genes the isoforms belong to.

```
> data(IsoList)
> str(IsoList)

List of 3
 $ IsoMat       : num [1:6000, 1:10] 176 789 1300 474 1061 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:6000] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
  .. ..$ : NULL
 $ IsoNames     : chr [1:6000] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
 $ IsosGeneNames: chr [1:6000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...

> IsoMat = IsoList$IsoMat
> str(IsoMat)
```

```
 num [1:6000, 1:10] 176 789 1300 474 1061 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:6000] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
  ..$ : NULL
```

```
> IsoNames = IsoList$IsoNames
> IsosGeneNames = IsoList$IsosGeneNames
```

### 3.2.2 Library size factor

Similar to the gene-level analysis presented above, we may obtain the isoform-level library size factors via `MedianNorm`:

```
> IsoSizes = MmedianNorm(IsoMat)
```

### 3.2.3 The $N_g$ vector

Since EBSeq fits rely on $N_g$, we need to obtain the $N_g$ for each isoform. This can be done using the function `GetNg`. The required inputs of `GetNg` are the isoform names (`IsoformNames`) and their corresponding gene names (`IsosGeneNames`).

```
> NgList = GetNg(IsoNames, IsosGeneNames)
> IsoNgTrun = NgList$IsoformNgTrun
> IsoNgTrun[c(1:3, 1001:1003, 3001:3003)]
```

```
Iso_1_1 Iso_1_2 Iso_1_3 Iso_2_1 Iso_2_2 Iso_2_3 Iso_3_1 Iso_3_2 Iso_3_3
      1       1       1       2       2       2       3       3       3
```

### 3.2.4 Running EBSeq on isoform counts

The `EBTest` function is also used to run EBSeq on isoform-level data. Below we use 5 iterations to demonstrate. However, as in the gene level analysis, we advise that additional iterations may be required in practice (see Section 4.2.5 for details).

```
> IsoEBOut = EBTest(Data = IsoMat, NgVector = IsoNgTrun, Conditions = as.factor(rep(c("C1",
+     "C2"), each = 5)), sizeFactors = IsoSizes, maxround = 5)
```

```
iteration 1 done
time 65.385
iteration 2 done
time 22.121
iteration 3 done
time 13.899
iteration 4 done
time 11.692
iteration 5 done
time 11.704
```

```
> IsoPP = GetPPMat(IsoEBOut)
> str(IsoPP)

 num [1:6000, 1:2] 0 0 0 0 0 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:6000] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
  ..$ : chr [1:2] "PPEE" "PPDE"

> head(IsoPP)

        PPEE PPDE
Iso_1_1    0    1
Iso_1_2    0    1
Iso_1_3    0    1
Iso_1_4    0    1
Iso_1_5    0    1
Iso_1_6    0    1

> IsoDE = rownames(IsoPP)[which(IsoPP[, "PPDE"] >= 0.95)]
> str(IsoDE)

 chr [1:534] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" "Iso_1_5" ...
```

We see that EBSeq found 534 DE isoforms at the target FDR of 0.05.

### 3.3 Working with more than two conditions

The object `MultiGeneMat` is a matrix containing 1,000 simulated genes with 6 samples: the first two samples are from condition 1; the second and the third sample are from condition 2; the last two samples are from condition 3.

```
> data(MultiGeneMat)
> str(MultiGeneMat)

 num [1:1000, 1:6] 411 1652 268 1873 768 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:1000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
  ..$ : NULL
```

In analyses where the data are spread over more than two conditions, the set of possible patterns for each gene is more complicated than simply EE and DE. As noted in Section 2, when we have 3 conditions, there are 5 expression patterns to consider. In the simulated data, we have 6 samples, 2 in each of 3 conditions. The function `GetPatterns` allows the user to generate all possible patterns given the conditions. For example:

```
> Conditions = c("C1", "C1", "C2", "C2", "C3", "C3")
> PosParti = GetPatterns(Conditions)
> PosParti
```

```
         C1 C2 C3
Pattern1  1  1  1
Pattern2  1  1  2
Pattern3  1  2  1
Pattern4  1  2  2
Pattern5  1  2  3
```

where the first row means all three conditions have the same latent mean expression level; the second row means C1 and C2 have the same latent mean expression level but that of C3 is different; and the last row corresponds to the case where the three conditions all have different latent mean expression levels. The user may use all or only some of these possible patterns as an input to `EBMultiTest` (more on this function presently). For example, if we were interested in Patterns 1, 2, 4 and 5 only, we'd define:

```
> Parti = PosParti[-3, ]
> Parti
```

```
         C1 C2 C3
Pattern1  1  1  1
Pattern2  1  1  2
Pattern4  1  2  2
Pattern5  1  2  3
```

Moving on to the analysis, `MedianNorm` or one of its competitors should be used to determine the normalization factors. Once this is done, the formal test is performed by `EBMultiTest`.

```
> MultiSize = MedianNorm(MultiGeneMat)
> MultiOut = EBMultiTest(MultiGeneMat, NgVector = NULL, Conditions = Conditions,
+     AllParti = Parti, sizeFactors = MultiSize, maxround = 5)
```

```
iteration 1 done
time 15.027
iteration 2 done
time 6.131
iteration 3 done
time 6.565
iteration 4 done
time 3.18099999999998
iteration 5 done
time 2.91400000000002
```

The posterior probobility of being in each pattern for every gene is obtained by using the function `GetMultiPP`:

```
> MultiPP = GetMultiPP(MultiOut)
> names(MultiPP)
```

```
[1] "PP"       "MAP"      "Patterns"
```

```
> MultiPP$PP[1:10, ]
```

```
             Pattern1  Pattern2      Pattern4   Pattern5
Gene_1   1.667196e-95 0.4474787  5.035574e-73 0.55252128
Gene_2   3.110967e-20 0.9697363  2.313379e-19 0.03026369
Gene_3  2.082947e-159 0.9679974 1.603355e-105 0.03200255
Gene_4  1.207559e-252 0.5357814 3.144834e-185 0.46421856
Gene_5   6.347984e-27 0.9371062  1.550532e-20 0.06289380
Gene_6   2.586284e-18 0.8383464  1.363798e-19 0.16165359
Gene_7   0.000000e+00 0.4467407  0.000000e+00 0.55325926
Gene_8   2.192617e-16 0.9850391  5.905463e-17 0.01496091
Gene_9   1.180902e-15 0.9453487  3.636282e-15 0.05465126
Gene_10  8.208183e-72 0.9219273  1.217895e-67 0.07807272
```

```
> MultiPP$MAP[1:10]
```

```
 [1] "Pattern5" "Pattern2" "Pattern2" "Pattern2" "Pattern2" "Pattern2"
 [7] "Pattern5" "Pattern2" "Pattern2" "Pattern2"
```

```
> MultiPP$Patterns
```

```
         C1 C2 C3
Pattern1  1  1  1
Pattern2  1  1  2
Pattern4  1  2  2
Pattern5  1  2  3
```

where `MultiPP$PP` provides the posterior probobility of being in each pattern for every gene. `MultiPP$MAP` provides the most likely pattern of each gene based on the posterior probabilities. `MultiPP$Patterns` provides the details of the patterns.

# 4 More detailed examples

## 4.1 Gene Level DE Analysis (Two Conditions)

### 4.1.1 Simulating gene-level counts

The function `GeneSimu` may be used to simulate gene-level count data. As in [6] and [2], the function assumes counts are distributed as Negative Binomial with gene-specific mean in sample $s$ and condition $C$ given by $l_s\mu_g^C$ and variance $l_s\mu_g^C(1 + l_s\mu_g^C\phi_g)$. We first generate 10,000 genes and 5 samples for each of two conditions. Here we use `DEGeneProp = 0.1` to define the DE gene percentage, so that 10% of the genes will be generated as DE. The EE genes are simulated as $\mu_g^{C1} = \mu_g^{C2}$. The DE genes are simualted as half $\mu_g^{C1} = \delta_g\mu_g^{C2}$ and half $\mu_g^{C2} = \delta_g\mu_g^{C1}$. We could define the DE genes to have constant $\delta_g$ equal to 4, say, by setting `DVDconstant = 4`. This simulation scenario was considered in [6] and [2]. Otherwise, the user may specify the `DVDqt1` and `DVDqt2` parameters to get non-constant $\delta_g$ from the corresponding lower and upper quantiles of the empirical $\delta_g$ values obtained from the data under investigation. (For example, let `DVDqt1=0.95` and `DVDqt2=0.97`. Then the $\delta_g$ will be randomly sampled from 95%-97% quantile of the empirical $\delta_g$ values.) To simulate the exact dataset in Section 3.1, we set a specific seed.

```
> set.seed(13)
> GeneGenerate = GeneSimu(DVDconstant = 4, DVDqt1 = NULL, DVDqt2 = NULL,
+     Conditions = rep(c("C1", "C2"), each = 5), NumofSample = 10,
+     NumofGene = 10000, DEGeneProp = 0.1, Phiconstant = NULL,
+     Phi.qt1 = 0.1, Phi.qt2 = 0.9, Meanconstant = NULL, OnlyData = T)
> GeneData = GeneGenerate$data
> GeneTrueDENames = GeneGenerate$TrueDE
```

The `GeneSimu` function is used to simulate a data matrix containing 10,000 rows of genes and 10 columns of samples. The genes are named `Gene_1, Gene_2 ...`

```
> str(GeneData)

 num [1:10000, 1:10] 1879 24 3291 97 485 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:10000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
  ..$ : NULL
```

where the first 10% of genes are simulated so that they are DE:

```
> str(GeneTrueDENames)

 chr [1:1000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" "Gene_5" ...
```

### 4.1.2 Running EBSeq on simulated gene counts

EBSeq is applied as described in Section 3.1.3.

```
> Sizes = MedianNorm(GeneData)
> EBOut = EBTest(Data = GeneData, Conditions = as.factor(rep(c("C1",
+     "C2"), each = 5)), sizeFactors = Sizes, maxround = 5)

iteration 1 done
time 23.007
iteration 2 done
time 13.113
iteration 3 done
time 12.728
iteration 4 done
time 12.232
iteration 5 done
time 10.945

> PP = GetPPMat(EBOut)
> str(PP)

 num [1:10000, 1:2] 0 0 0 0 0 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:10000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
  ..$ : chr [1:2] "PPEE" "PPDE"

> head(PP)

               PPEE PPDE
Gene_1 0.000000e+00    1
Gene_2 0.000000e+00    1
Gene_3 0.000000e+00    1
Gene_4 0.000000e+00    1
Gene_5 0.000000e+00    1
Gene_6 3.289292e-10    1

> DEfound = rownames(PP)[which(PP[, "PPDE"] >= 0.95)]
> str(DEfound)

 chr [1:991] "Gene_1" "Gene_2" "Gene_3" "Gene_4" "Gene_5" ...

> sum(DEfound %in% GeneTrueDENames)

[1] 959
```

EBSeq found 991 DE genes for a target FDR of 0.05, and we see that 959 of them were true positives.

### 4.1.3 Checking convergence

As detailed in Section 2, we assume the prior distribution of $q_g^C$ is $Beta(\alpha, \beta)$. The EM algorithm is used for estimating the hyper-parameters $\alpha, \beta$ and the mixture parameter $p$. The optimized parameters at each iteration may be obtained as follows (recall we are using 5 iterations for demonstration purposes):

```
> EBOut$Alpha
```

```
           [,1]
iter1 0.8442208
iter2 0.8451571
iter3 0.8440494
iter4 0.8441588
iter5 0.8441456
```

```
> EBOut$Beta
```

```
           Ng1
iter1 1.735640
iter2 1.762926
iter3 1.760836
iter4 1.760182
iter5 1.759134
```

```
> EBOut$P
```

```
           [,1]
iter1 0.1804475
iter2 0.1409131
iter3 0.1348617
iter4 0.1340951
iter5 0.1338376
```

In our case the differences between the 4th and 5th iteration are always less than 0.001.

### 4.1.4 Checking the model fit and other diagnostics

As noted in Leng *et al.*, 2012 [3], EBSeq relies on parametric assumptions that should be checked following each analysis. The `QQP` function may be used to assess prior assumptions. In practice, `QQP` generates the QQ plot of the empirical $q$'s vs the simulated $q$'s from the Beta prior distribution with estimated hyper-parameters (see Figure 2).

```
> par(mfrow = c(2, 2))
> QQP(EBOut, name = "Gene Simulation")
```

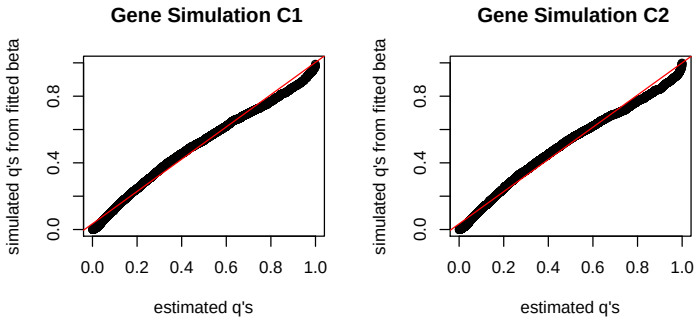


Figure 2: The QQ-plot for checking the assumption of a Beta prior as well as the model fit using data from condition 1 and condition 2

Here we see no violation of the Beta assumption. Likewise, the `DenNHist` function may be used to check the density plot of empirical $q$'s vs the simulated $q$'s from the fitted Beta prior distribution (see Figure 3).

```
> par(mfrow = c(2, 2))
> DenNHist(EBOut, name = "Gene Simulation")
```

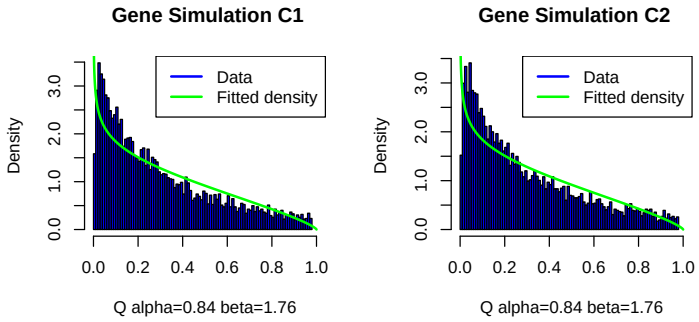


Figure 3: The density plot for checking the model fit using data from condition 1 and condition 2

## 4.2 Isoform Level DE Analysis (Two Conditions)

### 4.2.1 Simulating isoform-level counts

In order to simulate isoform-level data, the function `IsoSimu` may be used. As in Section 4.1.1, the function assumes counts are distributed as Negative Binomial with isoform-specific mean in sample $s$ and condition $C$ given by $l_s\mu_g^C$ and variance $l_s\mu_{gi}^C(1+l_s\mu_{gi}^C\phi_{gi})$. We first generate 6,000 isoforms and 5 samples for each of two conditions. Here we use `DEIsoProp = 0.1` to define the DE isoform percentage, so that 10% of the isoforms will be generated as DE. The EE isoforms are simulated as $\mu_{gi}^{C1} = \mu_{gi}^{C2}$. The DE isoforms are simualted as half $\mu_{gi}^{C1} = \delta_{gi}\mu_{gi}^{C2}$ and half $\mu_{gi}^{C2} = \delta_{gi}\mu_{gi}^{C1}$. Here we use `DVDqt1=0.95` and `DVDqt2=0.97` instead of using constant $\delta_{gi}$. Then the $\delta_{gi}$'s are randomly sampled from the 95%-97% quantile of the empirical $\delta_{gi}$ values.

And we use `NumofIso` to define the number of isoforms in each $N_g$ group: Here, we simulated 6,000 isoforms in total. The number of isoforms in $N_g$ =

$1, 2, 3$ groups are 1,000, 2,000 and 3,000, respectively.

```
> set.seed(13)
> IsoGenerate = IsoSimu(DVDconstant = NULL, DVDqt1 = 0.97, DVDqt2 = 0.98,
+     Conditions = as.factor(rep(c("C1", "C2"), each = 5)), NumofSample = 10,
+     NumofIso = c(1000, 2000, 3000), DEIsoProp = 0.1, Phiconstant = NULL,
+     Phi.qt1 = 0.25, Phi.qt2 = 0.75, OnlyData = T)
> str(IsoGenerate)
```

```
List of 2
 $ data  :List of 3
  ..$ : num [1:1000, 1:10] 176 789 1300 474 1061 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:1000] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
  .. .. ..$ : NULL
  ..$ : num [1:2000, 1:10] 669 11 494 11 251 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2000] "Iso_2_1" "Iso_2_2" "Iso_2_3" "Iso_2_4" ...
  .. .. ..$ : NULL
  ..$ : num [1:3000, 1:10] 384 84 0 66 44 132 23 9 668 54 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:3000] "Iso_3_1" "Iso_3_2" "Iso_3_3" "Iso_3_4" ...
  .. .. ..$ : NULL
 $ TrueDE: chr [1:600] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
```

`TrueDENames` is a vector containing all the isoforms that are truly DE. Since `EBTest` requires a matrix that contains all of the isoform expressions, we need to convert the list `IsoGenerate$data` into a matrix:

```
> IsoMat = do.call(rbind, IsoGenerate$data)
> str(IsoMat)
```

```
 num [1:6000, 1:10] 176 789 1300 474 1061 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:6000] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
  ..$ : NULL
```

### 4.2.2 The $N_g$ vector

Since EBSeq fits rely on $N_g$, we need to obtain the $N_g$ for each isoform. This can be done using the function `GetNg`. The required inputs of `GetNg` are the isoform names (`IsoformNames`) and their corresponding gene names (`IsosGeneNames`), described above. In the simulated data, we assume that the isoforms in the $N_g = 1$ group belong to genes `Gene_1, ... , Gene_1000`; The isoforms in the $N_g = 2$ group belong to genes `Gene_1001, ..., Gene_2000`; and isoforms in the $N_g = 3$ group belong to `Gene_2001, ..., Gene_3000`.

```
> IsoNames = rownames(IsoMat)
> str(IsoNames)
```

```
 chr [1:6000] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" "Iso_1_5" ...

> GeneNames = paste("Gene", c(1:3000), sep = "_")
> IsosGeneNames = c(GeneNames[1:1000], rep(GeneNames[1001:2000],
+     each = 2), rep(GeneNames[2001:3000], each = 3))
> NgList = GetNg(IsoNames, IsosGeneNames, TrunThre = 3)
> names(NgList)

[1] "GeneNg"        "GeneNgTrun"    "IsoformNg"     "IsoformNgTrun"

> IsoNgTrun = NgList$IsoformNgTrun
> IsoNgTrun[c(1:3, 1001:1003, 3001:3003)]

Iso_1_1 Iso_1_2 Iso_1_3 Iso_2_1 Iso_2_2 Iso_2_3 Iso_3_1 Iso_3_2 Iso_3_3
      1       1       1       2       2       2       3       3       3
```

`GetNg` contains 4 vectors. `GeneNg` (`IsoformNg`) provides the number of isoforms within each gene (within each isoform's host gene). `GeneNgTrun` (`IsoformNgTrun`) provides the truncated $N_g$ values. The defalt truncation threshold is 3, which means the values in `GeneNg` (`IsoformNg`) who are greater than 3 will be changed to 3 in `GeneNgTrun` (`IsoformNgTrun`). We use 3 in the case studies since the number of isoforms with $N_g$ larger than 3 is relatively small and the small sample size may induce poor parameter fitting if we treat them as separate groups. In practice, if there is evidence that the $N_g = 4, 5, 6...$ groups should be treated as separate groups, a user can change `TrunThre` to define a different truncation threshold.

#### 4.2.3 Using mappability ambiguity clusters instead of the $N_g$ vector when the gene-isoform relationship is unknown

While working with a de-novo assembled transcriptome, in which case the gene-isoform relationship is unknown, a user can use read mapping ambiguity cluster information instead of Ng, as provided by RSEM [4] in the output file `output_name.ngvec`. The file contains a vector with the same length as the total number of transcripts. Each transcript has been assigned to one of 3 levels (1, 2, or 3) to indicate the mapping uncertainty level of that transcript. A user can read in the mapping ambiguity cluster information using:

```
> IsoNgTrun = scan(file = "output_name.ngvec", what = 0, sep = "\n")
```

More details on using the RSEM-EBSeq pipeline on de novo assembled transcriptomes can be found at `http://deweylab.biostat.wisc.edu/rsem/README.html#de`.

#### 4.2.4 Running EBSeq on simulated isoform counts

EBSeq can be applied as described in Section 3.2.4.

```
> IsoSizes = MedianNorm(IsoMat)
> IsoEBOut = EBTest(Data = IsoMat, NgVector = IsoNgTrun, Conditions = as.factor(rep(c("C1",
+     "C2"), each = 5)), sizeFactors = IsoSizes, maxround = 5)

iteration 1 done
time 64.404
iteration 2 done
time 21.936
iteration 3 done
time 13.845
iteration 4 done
time 11.87
iteration 5 done
time 11.873

> IsoPP = GetPPMat(IsoEBOut)
> str(IsoPP)

 num [1:6000, 1:2] 0 0 0 0 0 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:6000] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
  ..$ : chr [1:2] "PPEE" "PPDE"

> IsoDE = rownames(IsoPP)[which(IsoPP[, "PPDE"] >= 0.95)]
> str(IsoDE)

 chr [1:534] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" "Iso_1_5" ...

> sum(IsoDE %in% IsoGenerate$TrueDE)

[1] 507
```

We see that EBSeq found 534 DE isoforms at a target FDR of 0.05; 507 of those were true positives.

### 4.2.5 Checking convergence

For isoform level data, we assume the prior distribution of $q_{gi}^C$ is $Beta(\alpha, \beta^{N_g})$. As in Section 4.1.3, the optimized parameters at each iteration may be obtained as follows (recall we are using 5 iterations for demonstration purposes):

```
> IsoEBOut$Alpha

           [,1]
iter1 0.6964958
iter2 0.7056876
iter3 0.7036317
iter4 0.7040022
iter5 0.7032276
```

```
> IsoEBOut$Beta

           Ng1      Ng2      Ng3
iter1 1.642912 2.261051 2.654820
iter2 1.695865 2.376032 2.834243
iter3 1.693549 2.367979 2.827887
iter4 1.692093 2.370663 2.835483
iter5 1.692036 2.363139 2.826236

> IsoEBOut$P

           [,1]
iter1 0.2072478
iter2 0.1590444
iter3 0.1473788
iter4 0.1442685
iter5 0.1430039
```

Here we have 3 $\beta$'s in each iteration corresponding to $\beta^{N_g=1}, \beta^{N_g=2}, \beta^{N_g=3}$. We see that parameters are fixed within $10^{-2}$ or $10^{-3}$. In practice, we require changes less than $10^{-3}$ to declare convergence.

#### 4.2.6 Checking the model fit and other diagnostics

In Leng *et al.*, 2012[3], we showed the mean-variance differences across different isoform groups on multiple data sets. In practice, if it is of interest to check differences among isoform groups defined by truncated $N_g$ (such as those shown here in Figure 1), the function `PolyFitValue` may be used. The following code generates the three panels shown in Figure 4 (if condition 2 is of interest, a user could change each `C1` to `C2`.):

```
> par(mfrow = c(2, 2))
> PolyFitValue = vector("list", 3)
> for (i in 1:3) PolyFitValue[[i]] = PolyFitPlot(IsoEBOut$C1Mean[[i]],
+     IsoEBOut$C1EstVar[[i]], 5)
```

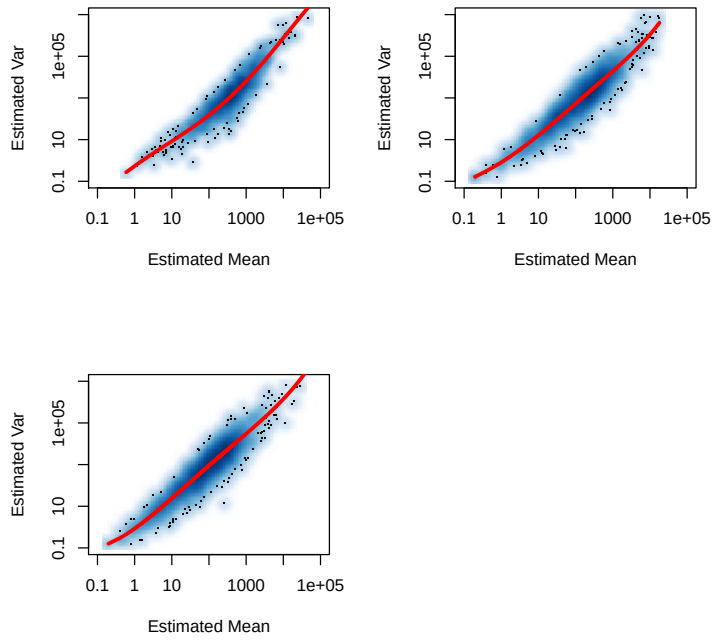


Figure 4: The mean-variance fitting plot for each Ng group

Superimposing all $N_g$ groups using the code below will generate the figure (shown here in Figure 5), which is similar in structure to Figure 1:

```
> PolyAll = PolyFitPlot(unlist(IsoEBOut$C1Mean), unlist(IsoEBOut$C1EstVar),
+     5)
> lines(log10(IsoEBOut$C1Mean[[1]][PolyFitValue[[1]]$sort]), PolyFitValue[[1]]$fit[PolyFitVa
+     col = "yellow", lwd = 2)
> lines(log10(IsoEBOut$C1Mean[[2]][PolyFitValue[[2]]$sort]), PolyFitValue[[2]]$fit[PolyFitVa
+     col = "pink", lwd = 2)
> lines(log10(IsoEBOut$C1Mean[[3]][PolyFitValue[[3]]$sort]), PolyFitValue[[3]]$fit[PolyFitVa
+     col = "green", lwd = 2)
> legend("topleft", c("All Isoforms", "Ng = 1", "Ng = 2", "Ng = 3"),
+     col = c("red", "yellow", "pink", "green"), lty = 1, lwd = 3,
+     box.lwd = 2)
```

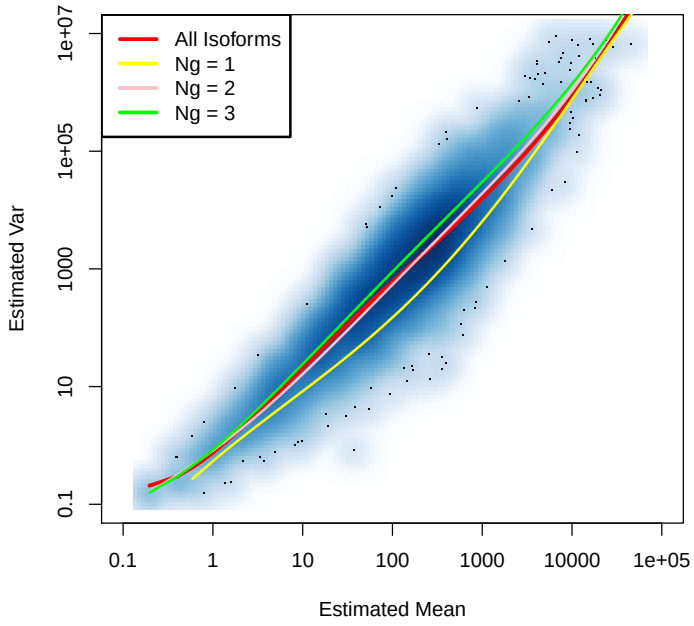


Figure 5: The mean-variance plot for each Ng group

To generate a QQ-plot of the fitted Beta prior distribution and the $\hat{q^C}$'s within condition, a user may use the following code to generate 6 panels (as in the gene-level analysis):

```
> par(mfrow = c(2, 3))
> QQP(IsoEBOut, name = "Isoforms", GroupName = paste("Ng = ", c(1:3),
+     sep = ""))
```

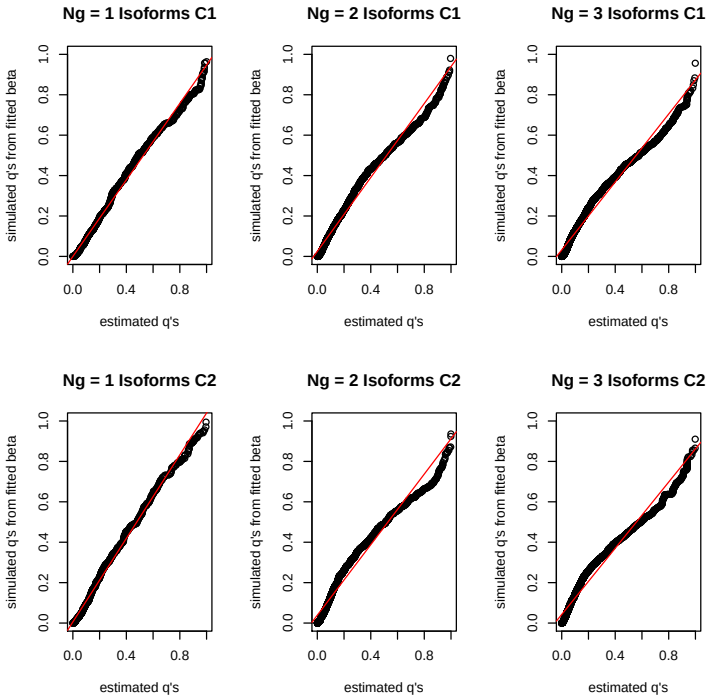


Figure 6: The QQ-plot of the fitted prior distribution within each Ng group

And in order to produce the plot of the fitted Beta prior densities and the histograms of $\hat{q^C}$'s within each condition, the following may be used (it generates Figure 7):

```
> par(mfrow = c(2, 3))
> DenNHist(IsoEBOut, name = "Isoforms", GroupName = paste("Ng = ",
+     c(1:3), sep = ""))
```

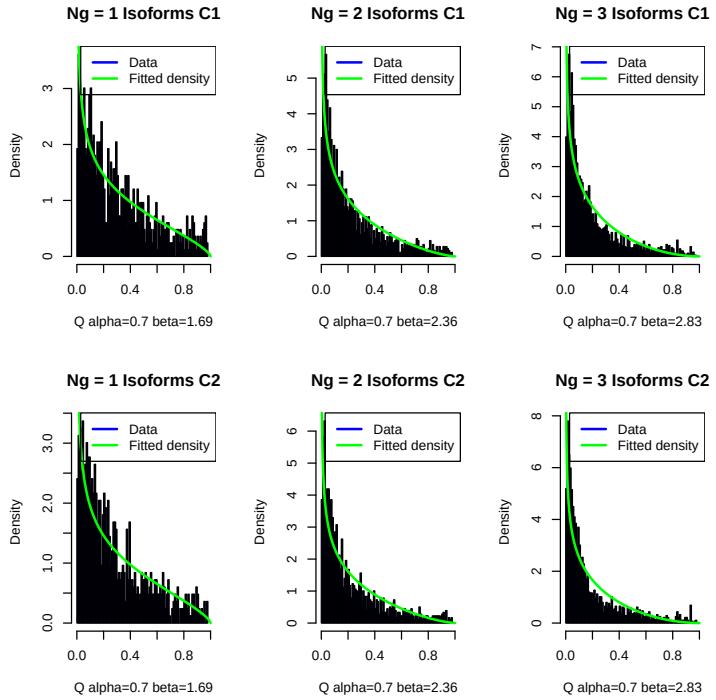


Figure 7: The prior distribution fitting within each Ng group

## 4.3 Working with more than two conditions

As described in Section 3.3, the function `GetPatterns` allows the user to generate all possible patterns given the conditions. To visualize the patterns, the function `PlotPattern` may be used.
As described, if we were interested in Patterns 1, 2, 4 and 5 only, we'd define:

```
> Parti = PosParti[-3, ]
> Parti
```

```
         C1 C2 C3
Pattern1  1  1  1
Pattern2  1  1  2
Pattern4  1  2  2
Pattern5  1  2  3
```

This established, we simulate 1,000 genes with 6 samples. The proportions of

```
> Conditions = c("C1", "C1", "C2", "C2", "C3", "C3")
> PosParti = GetPatterns(Conditions)
> PosParti

         C1 C2 C3
Pattern1  1  1  1
Pattern2  1  1  2
Pattern3  1  2  1
Pattern4  1  2  2
Pattern5  1  2  3

> PlotPattern(PosParti)
```

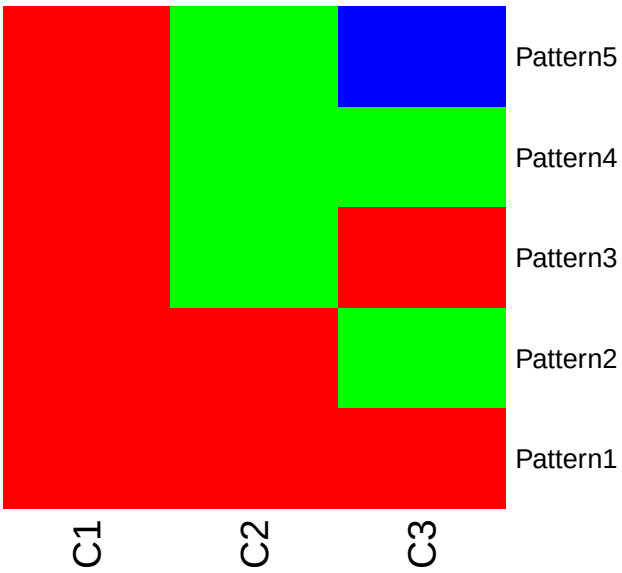


Figure 8: All possible patterns

genes in each of our four patterns are $(0.7, 0.1, 0.1, 0.1)$:

```
> set.seed(13)
> MultiData = GeneMultiSimu(Conditions = Conditions, AllParti = Parti,
+     NumofSample = 6, NumofGene = 1000, DEGeneProp = c(0.7, 0.1,
+         0.1, 0.1), DVDqt1 = 0.98, DVDqt2 = 0.99, Phi.qt1 = 0.25,
+     Phi.qt2 = 0.75)
```

```
> str(MultiData)

List of 2
 $ data    : num [1:1000, 1:6] 411 1652 268 1873 768 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:1000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
  .. ..$ : NULL
 $ Patterns: Named chr [1:1000] "Pattern2" "Pattern2" "Pattern2" "Pattern2" ...
  ..- attr(*, "names")= chr [1:1000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
```

`MultiData$data` provides the expression matrix. `MultiData$Patterns` provides the true pattern to which each gene belongs.

Moving on to the analysis, `MedianNorm` or one of its competitors should be used to determine the normalization factors. Once this is done, the formal test is performed by `EBMultiTest`.

```
> MultiSize = MedianNorm(MultiData$data)
> MultiOut = EBMultiTest(MultiData$data, NgVector = NULL, Conditions = Conditions,
+     AllParti = Parti, sizeFactors = MultiSize, maxround = 5)

iteration 1 done
time 15.117
iteration 2 done
time 6.08600000000001
iteration 3 done
time 6.50399999999996
iteration 4 done
time 3.15100000000001
iteration 5 done
time 2.89600000000002
```

The posterior probability of being in each pattern for every gene is obtained using the function `GetMultiPP`:

```
> MultiPP = GetMultiPP(MultiOut)
> names(MultiPP)

[1] "PP"       "MAP"      "Patterns"

> MultiPP$PP[1:10, ]

             Pattern1  Pattern2      Pattern4   Pattern5
Gene_1   1.667196e-95 0.4474787  5.035574e-73 0.55252128
Gene_2   3.110967e-20 0.9697363  2.313379e-19 0.03026369
Gene_3  2.082947e-159 0.9679974 1.603355e-105 0.03200255
Gene_4  1.207559e-252 0.5357814 3.144834e-185 0.46421856
Gene_5   6.347984e-27 0.9371062  1.550532e-20 0.06289380
Gene_6   2.586284e-18 0.8383464  1.363798e-19 0.16165359
```

```
Gene_7   0.000000e+00 0.4467407  0.000000e+00 0.55325926
Gene_8   2.192617e-16 0.9850391  5.905463e-17 0.01496091
Gene_9   1.180902e-15 0.9453487  3.636282e-15 0.05465126
Gene_10  8.208183e-72 0.9219273  1.217895e-67 0.07807272
```

```
> MultiPP$MAP[1:10]
```

```
 [1] "Pattern5" "Pattern2" "Pattern2" "Pattern2" "Pattern2" "Pattern2"
 [7] "Pattern5" "Pattern2" "Pattern2" "Pattern2"
```

```
> MultiPP$Patterns
```

```
         C1 C2 C3
Pattern1  1  1  1
Pattern2  1  1  2
Pattern4  1  2  2
Pattern5  1  2  3
```

where `MultiPP$PP` provides the posterior probobility of being in each pattern for every gene. `MultiPP$MAP` provides the most likely pattern of each gene based on the posterior probabilities. `MultiPP$Patterns` provides the details of the patterns.

```
> sum(MultiPP$MAP == MultiData$Patterns)
```

```
[1] 918
```

For this simulated data set with 3 conditions, EBSeq identified the correct expression pattern 91.8% of the time.

# References


[1] S Anders and W Huber. Differential expression analysis for sequence count data. *Genome Biology*, 11:R106, 2010.

[2] T J Hardcastle and K A Kelly. bayseq: empirical bayesian methods for identifying differential expression in sequence count data. *BMC Bioinformatics*, 11:422, 2010.

[3] N. Leng, J.A. Dawson, J.A Thomson, V Ruotti, R. A. Rissman, B.M.G Smits, J.D. Hagg, M.N. Gould, R.M. Stwart, and C. Kendziorski. Ebseq: An empirical bayes hierarchical model for inference in rna-seq experiments. *BMI technical report, University of Wisconsin Madison*, 226, 2012.

[4] B Li and C N Dewey. Rsem: accurate transcript quantification from rna-seq data with or without a reference genome. *BMC Bioinformatics*, 12:323, 2011.

[5] M D Robinson and Oshlack A. A scaling normalization method for differential expression analysis of rna-seq data. *Genome Biology*, 11:R25, 2010.

[6] M D Robinson and G K Smyth. Moderated statistical tests for assessing differences in tag abundance. *Bioinformatics*, 23(21):2881–2887, 2007.

[7] C Trapnell, A Roberts, L Goff, G Pertea, D Kim, D R Kelley, H Pimentel, S L Salzberg, J L Rinn, and L Pachter. Differential gene and transcript expression analysis of rna-seq experiments with tophat and cufflinks. *Nature Protocols*, 7(3):562–578, 2012.