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

Ning Leng, John A. Dawson, Christina Kendziorski

October 9, 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 $I_g = k$ group, where $k = 1, 2, 3$, if the total number of isoforms from gene g is k (the $I_g = 3$ group contains all isoforms from genes having 3 or more isoforms). As shown in Figure 1, there is decreased variability in the $I_g = 1$ 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 $I_g = 1$ 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 $I_g = 2$ and $I_g = 3$ 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 I_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.

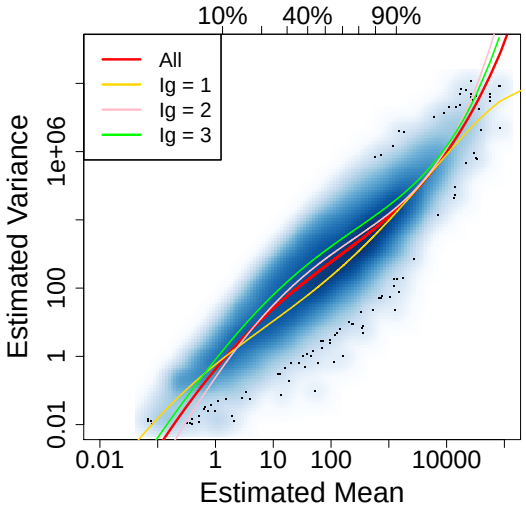

Figure 1: Empirical variance vs. mean for each isoform profiled in the ESCs vs iPSCs experiment detailed in the Case Study section of Leng *et al.*, 2012 [3]. A spline fit to all isoforms is shown in red with splines fit within the $I_g = 1$, $I_g = 2$, and $I_g = 3$ isoform groups shown in yellow, pink,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

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

We assume a prior distribution on $q_{g_i}^C$: $q_{g_i}^C | \alpha, \beta^{I_g} \sim Beta(\alpha, \beta^{I_g})$. The hyperparameter α is shared by all the isoforms and β^{I_g} is I_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):

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

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

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

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

$$f_1^{I_g}(X_{g_i}^{C1,C2}) = f_0^{I_g}(X_{g_i}^{C1}) f_0^{I_g}(X_{g_i}^{C2}) \tag{3}$$

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:

$$(1-p) f_0^{I_g}(X_{g_i}^{C1,C2}) + p f_1^{I_g}(X_{g_i}^{C1,C2}) \tag{4}$$

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

$$\frac{p f_1^{I_g}(X_{g_i}^{C1,C2})}{(1-p) f_0^{I_g}(X_{g_i}^{C1,C2}) + p f_1^{I_g}(X_{g_i}^{C1,C2})} \tag{5}$$

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:

P1: $q_{g_i}^{C1} = q_{g_i}^{C2} = q_{g_i}^{C3}$
P2: $q_{g_i}^{C1} = q_{g_i}^{C2} \neq q_{g_i}^{C3}$
P3: $q_{g_i}^{C1} = q_{g_i}^{C3} \neq q_{g_i}^{C2}$
P4: $q_{g_i}^{C1} \neq q_{g_i}^{C2} = q_{g_i}^{C3}$
P5: $q_{g_i}^{C1} \neq q_{g_i}^{C2} \neq q_{g_i}^{C3}$ and $q_{g_i}^{C1} \neq q_{g_i}^{C3}$

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

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

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

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

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

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

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 [6] 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. 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`. (e.g. `QuantileNorm(GeneMat,.75)` for Upper-Quantile Normalization in [2])

3.1.3 Running EBSeq on gene expression estimations

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 I_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], Up-Quantile Normalization [2], 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 25.035
iteration 2 done
time 14.094
iteration 3 done
time 13.651
iteration 4 done
time 13.198
iteration 5 done
time 11.716
```

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"]>=.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=MedianNorm(IsoMat)
```

3.2.3 The I_g vector

Since EBSeq fits rely on I_g, we need to obtain the number of isoforms the host gene contains (N_g) for each isoform therefore obtain the I_g groups. 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
```

More details could be found in section 4.2.

3.2.4 Running EBSeq on isoform expression estimations

The `EBTest` function is also used to run EBSeq for two condition comparisons 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 4.2.4 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 69.102
iteration 2 done
time 23.495
iteration 3 done
```

```
time 14.768
iteration 4 done
time 12.473
iteration 5 done
time 12.403
```

```
> 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"]>=.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 on gene expression estimations 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 14.325
iteration 2 done
time 5.876
iteration 3 done
time 6.345
iteration 4 done
```

```
time 3.03800000000001
iteration 5 done
time 2.798
```

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]

    Gene_1     Gene_2     Gene_3     Gene_4     Gene_5     Gene_6     Gene_7
"Pattern5" "Pattern2" "Pattern2" "Pattern2" "Pattern2" "Pattern2" "Pattern5"
    Gene_8     Gene_9    Gene_10
"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.

3.4 Working on isoform expression estimations with more than two conditions

Similar to `IsoList` The object `IsoMultiList` is a object containing the isoform expression estimations matrix, isoform names and gene names of isoforms' host genes. `IsoMultiList$IsoMultiMat` contains 1,000 simulated isoforms with 8 samples. the first two samples are from condition 1; the second and the third sample are from condition 2; the fifth and sixth sample are from condition 3; the last two samples are from condition 4. Similar as in section 3.2, function `MedianNorm` and `GetNg` could used for normalization and calculating N_g's.

```
> data(IsoMultiList)
> IsoMultiMat=IsoMultiList[[1]]
> IsoNames.Multi=IsoMultiList$IsoNames
> IsosGeneNames.Multi=IsoMultiList$IsosGeneNames
> IsoMultiSize=MedianNorm(IsoMultiMat)
> NgList.Multi=GetNg(IsoNames.Multi, IsosGeneNames.Multi)
> IsoNgTrun.Multi=NgList.Multi$IsoformNgTrun
> Conditions=c("C1","C1","C2","C2","C3","C3","C4","C4")
```

Here we have 4 conditions, there are 15 expression patterns to consider. The function `GetPatterns` allows the user to generate all possible patterns given the conditions. For example:

```
> PosParti.4Cond=GetPatterns(Conditions)
> PosParti.4Cond
```

```
          C1 C2 C3 C4
Pattern1   1  1  1  1
Pattern2   1  1  1  2
Pattern3   1  1  2  1
Pattern4   1  1  2  2
Pattern5   1  2  1  1
Pattern6   1  2  1  2
Pattern7   1  2  2  1
Pattern8   1  2  2  2
Pattern9   1  1  2  3
Pattern10  1  2  1  3
Pattern11  1  2  2  3
Pattern12  1  2  3  1
Pattern13  1  2  3  2
Pattern14  1  2  3  3
Pattern15  1  2  3  4
```

If we were interested in Patterns 1, 2, 3, 8 and 15 only, we'd define:

```
> Parti.4Cond=PosParti.4Cond[c(1,2,3,8,15),]
> Parti.4Cond
```

```
          C1 C2 C3 C4
Pattern1   1  1  1  1
Pattern2   1  1  1  2
Pattern3   1  1  2  1
Pattern8   1  2  2  2
Pattern15  1  2  3  4
```

Moving on to the analysis, `EBMultiTest` could be used to porferm the test:

```
> IsoMultiOut=EBMultiTest(IsoMultiMat,
+ NgVector=IsoNgTrun.Multi,Conditions=Conditions,
+ AllParti=Parti.4Cond, sizeFactors=IsoMultiSize,
+ maxround=5)

iteration 1 done
time 24.537
iteration 2 done
time 24.518
iteration 3 done
time 9.322
iteration 4 done
time 10.424
iteration 5 done
time 8.94400000000002
```

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

```
> IsoMultiPP=GetMultiPP(IsoMultiOut)
> names(MultiPP)

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

> IsoMultiPP$PP[1:10,]

              Pattern1    Pattern2      Pattern3      Pattern8    Pattern15
Iso_1_1   4.491089e-34 0.999067581  1.357994e-33  2.728660e-34 9.324189e-04
Iso_1_2   2.645937e-14 0.998132248  3.172235e-15  2.816801e-16 1.867752e-03
Iso_1_3   1.270054e-43 0.978252880  4.196709e-38  6.630693e-45 2.174712e-02
Iso_1_4   1.585237e-36 0.994599329  6.156303e-30  1.145819e-32 5.400671e-03
Iso_1_5   0.000000e+00 1.000000000  0.000000e+00  0.000000e+00 2.830424e-41
Iso_1_6  4.942113e-228 0.003241202 5.563985e-214 1.657836e-183 9.967588e-01
Iso_1_7  7.900232e-115 0.997756178 3.895311e-110 2.050441e-105 2.243822e-03
Iso_1_8  5.430600e-130 0.851933966  1.834973e-96 1.987346e-111 1.480660e-01
Iso_1_9   4.182136e-45 0.804797880  5.085895e-47  3.408432e-42 1.952021e-01
Iso_1_10  6.948051e-08 0.997979087  3.026767e-08  5.123101e-08 2.020762e-03

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

```
    Iso_1_1     Iso_1_2     Iso_1_3     Iso_1_4     Iso_1_5     Iso_1_6
 "Pattern2"  "Pattern2"  "Pattern2"  "Pattern2"  "Pattern2" "Pattern15"
    Iso_1_7     Iso_1_8     Iso_1_9    Iso_1_10
 "Pattern2"  "Pattern2"  "Pattern2"  "Pattern2"
```

```
> IsoMultiPP$Patterns
```

```
          C1 C2 C3 C4
Pattern1   1  1  1  1
Pattern2   1  1  1  2
Pattern3   1  1  2  1
Pattern8   1  2  2  2
Pattern15  1  2  3  4
```

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 Running EBSeq on simulated gene expression estimations

EBSeq is applied as described in Section 3.1.3.

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

iteration 1 done
time 24.734
iteration 2 done
time 13.912
iteration 3 done
time 13.477
iteration 4 done
time 12.976
iteration 5 done
time 11.546

> 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"]>=.95)]
> str(DEfound)

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

EBSeq found 991 DE genes for a target FDR of 0.05.

4.1.2 Calculating FC

The function `PostFC` could be used to calculate the Fold Change (FC) as well as the posterior FC on the normalization factor adjusted data. Figure 2 shows

```
> GeneFC=PostFC(EBOut)
> str(GeneFC)

List of 3
 $ PostFC   : Named num [1:10000] 0.239 0.244 4.152 4.238 3.927 ...
  ..- attr(*, "names")= chr [1:10000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
 $ RealFC   : Named num [1:10000] 0.239 0.24 4.154 4.305 3.942 ...
  ..- attr(*, "names")= chr [1:10000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
 $ Direction: chr "C1 Over C2"

> PlotPostVsRawFC(EBOut,GeneFC)
```

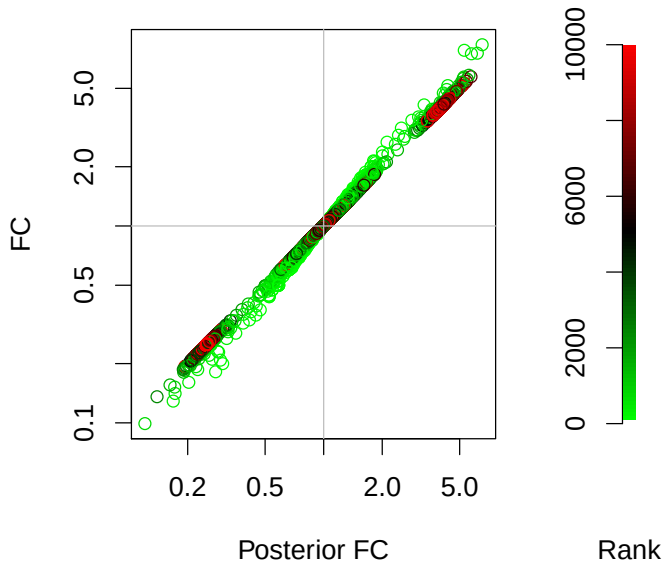

Figure 2: The Posterior FC vs FC on 10,000 gene expression estimations

Posterior FC vs FC on 10,000 gene expression estimations. The genes are ranked by their cross-condition mean (adjusted by the normalization factors). The posterior FC tends to be shrinkaged (be closer to 1) on the genes with low expressions (small rank).

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 α, β 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 Q-Q plot of the empirical q's vs the simulated q's from the Beta prior distribution with estimated hyper-parameters. Figure 3 shows that the data points lie on the $y = x$ line for both condition, which indicates the good fitting of our Beta prior distribution.

```
> par(mfrow=c(2,2))
> QQP(EBOut)
```

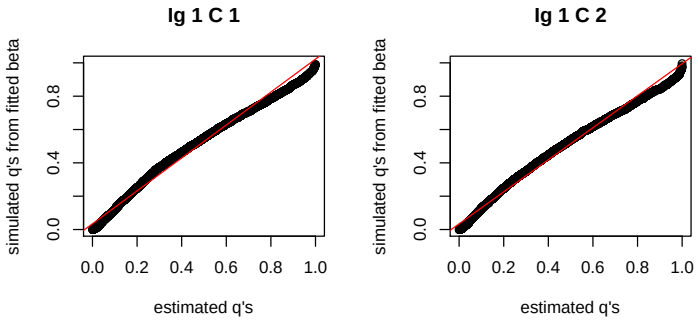

Figure 3: The QQ-plot for checking the assumption of a Beta prior as well as the model fit using data from condition 1 and condition 2

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. Figure 4 also shows our estimated distribution fits the data very well.

```
> par(mfrow=c(2,2))
> DenNHist(EBOut)
```

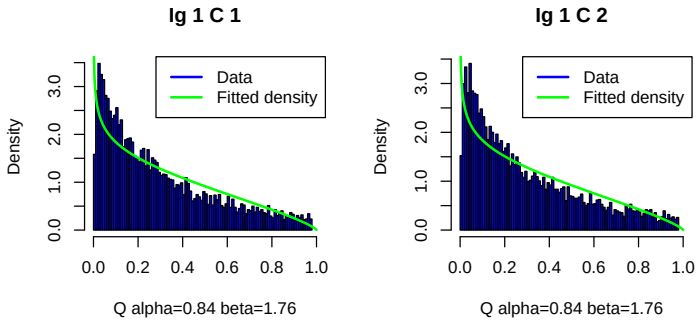

Figure 4: The density plot for checking the model fit using data from condition 1 and condition 2

4.2 Isoform Level DE Analysis (Two Conditions)

4.2.1 The I_g vector

Since EBSeq fits rely on I_g, we need to obtain the I_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 $I_g = 1$ group belong to genes `Gene_1, ... , Gene_1000`; The isoforms in the $I_g = 2$ group belong to genes `Gene_1001, ..., Gene_2000`; and isoforms in the $I_g = 3$ group belong to `Gene_2001, ..., Gene_3000`.

```
> 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$IsoMat
> IsoNames=IsoList$IsoNames
> IsosGeneNames=IsoList$IsosGeneNames
> 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 N_g within each gene (within each isoform's host gene). `GeneNgTrun` (`IsoformNgTrun`) provides the truncated N_g values (I_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.

4.2.2 Using mappability ambiguity clusters instead of the I_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. The mapping ambiguity clusters are partitioned via k-means algorithm on the unmapability scores by RSEM. 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`.

Other unmappability scores and other cluster methods (e.g. Gaussian Mixed Model) could also be used to form the uncertainty clusters.

4.2.3 Running EBSeq on simulated isoform expression estimations

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 67.788
iteration 2 done
time 22.982
iteration 3 done
time 14.4580000000001
iteration 4 done
time 12.183
iteration 5 done
time 12.1990000000001

> 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"]>=.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 a target FDR of 0.05.

The function `PostFC` could also be used here to calculate the Fold Change (FC) as well as the posterior FC on the normalization factor adjusted data.

```
> IsoFC=PostFC(IsoEBOut)
> str(IsoFC)

List of 3
 $ PostFC   : Named num [1:6000] 0.285 0.28 3.54 0.304 3.743 ...
  ..- attr(*, "names")= chr [1:6000] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
 $ RealFC   : Named num [1:6000] 0.284 0.28 3.544 0.304 3.749 ...
  ..- attr(*, "names")= chr [1:6000] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
 $ Direction: chr "C1 Over C2"
```

4.2.4 Checking convergence

For isoform level data, we assume the prior distribution of q_{gi}^C is $Beta(\alpha, \beta^{I_g})$. 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 β's in each iteration corresponding to $\beta^{I_g=1}, \beta^{I_g=2}, \beta^{I_g=3}$. We see that parameters are fixed within 10^{-2} or 10^{-3}. In practice, we require changes less than 10^{-3} to declare convergence.

4.2.5 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 I_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 5 (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)
```

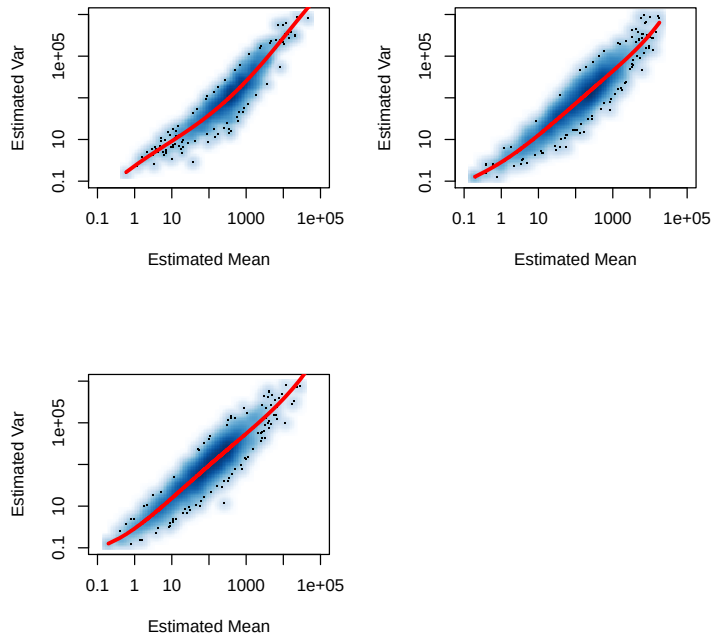

Figure 5: The mean-variance fitting plot for each Ng group

Superimposing all I_g groups using the code below will generate the figure (shown here in Figure 6), 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]]$fit[PolyFitValue[[1]]$sort],col="yellow",lwd=2)
> lines(log10(IsoEBOut$C1Mean[[2]][PolyFitValue[[2]]$sort]),
+ PolyFitValue[[2]]$fit[PolyFitValue[[2]]$sort],col="pink",lwd=2)
> lines(log10(IsoEBOut$C1Mean[[3]][PolyFitValue[[3]]$sort]),
+ PolyFitValue[[3]]$fit[PolyFitValue[[3]]$sort],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)
```

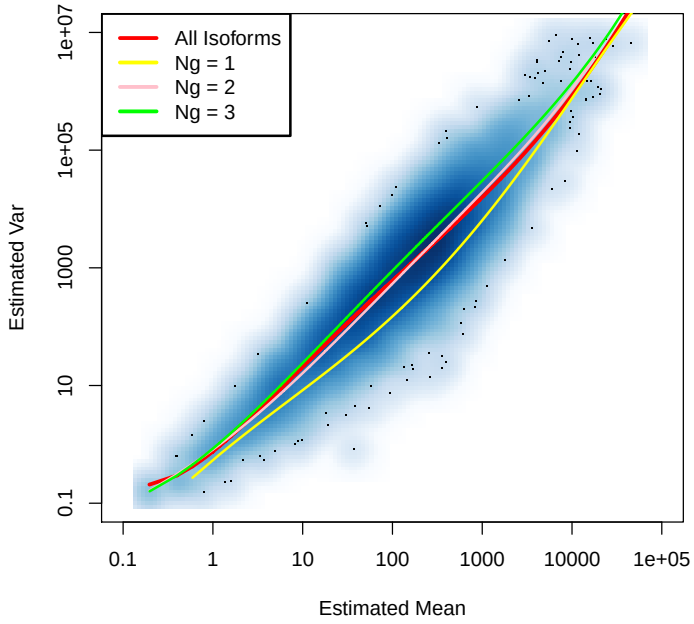

Figure 6: The mean-variance plot for each Ng group

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 7 panels (as in the gene-level analysis):

```
> par(mfrow=c(2,3))
> QQP(IsoEBOut)
```

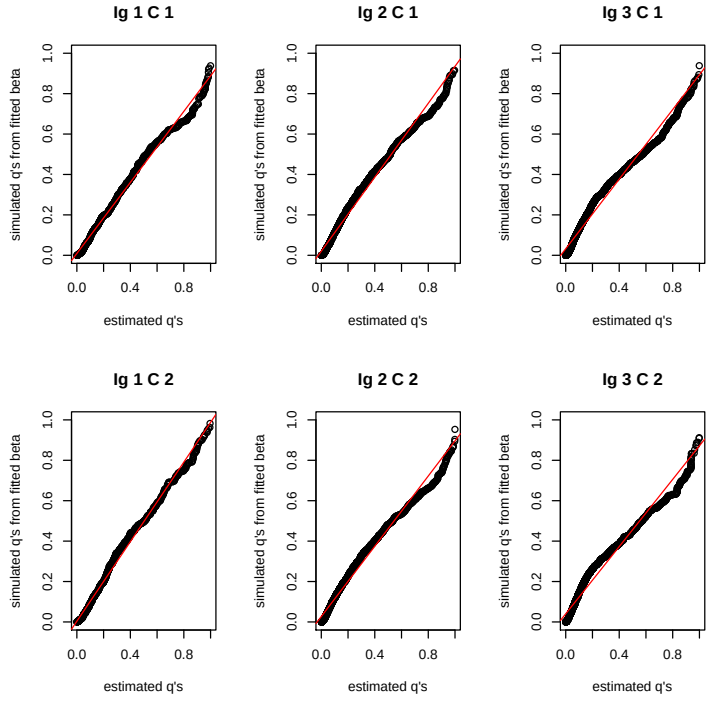

Figure 7: The QQ-plot of the fitted prior distribution within each Ig group

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 8):

```
> par(mfrow=c(2,3))
> DenNHist(IsoEBOut)
```

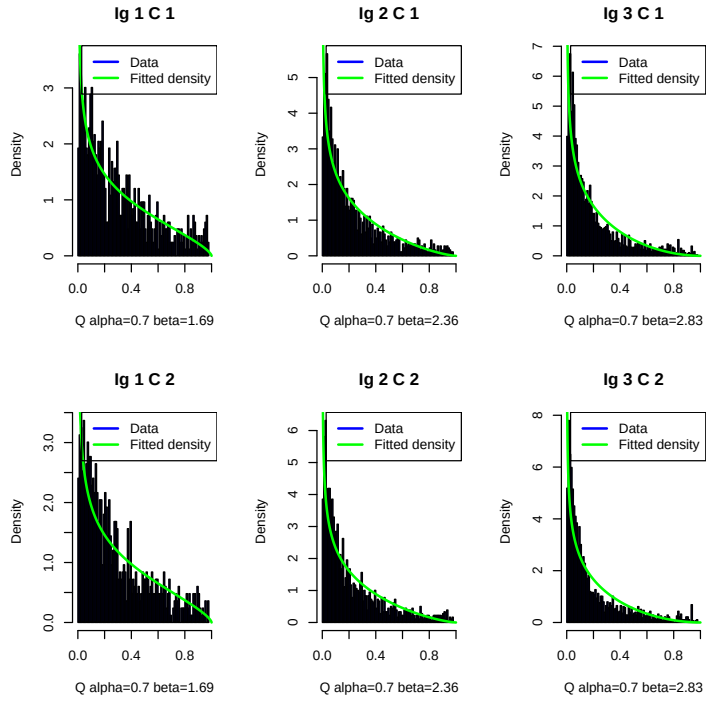

Figure 8: The prior distribution fitting within each Ig group

4.3 Working on gene expression estimations 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
```

Moving on to the analysis, `MedianNorm` or one of its competitors should be

```
> 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)
```

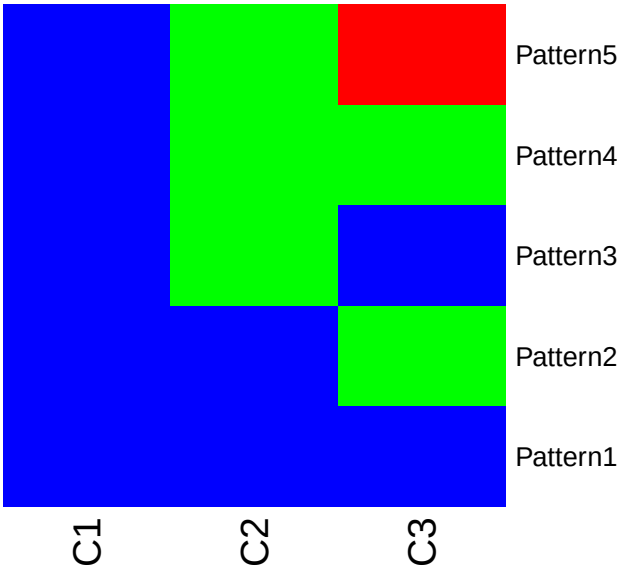

Figure 9: All possible patterns

used to determine the normalization factors. Once this is done, the formal test is performed by `EBMultiTest`.

```
> data(MultiGeneMat)
> MultiSize=MedianNorm(MultiGeneMat)
> MultiOut=EBMultiTest(MultiGeneMat,
+ NgVector=NULL,Conditions=Conditions,
```

```
+ AllParti=Parti, sizeFactors=MultiSize,
+ maxround=5)

iteration 1 done
time 14.434
iteration 2 done
time 5.91600000000005
iteration 3 done
time 6.28700000000003
iteration 4 done
time 3.03699999999992
iteration 5 done
time 2.80200000000002
```

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]

    Gene_1     Gene_2     Gene_3     Gene_4     Gene_5     Gene_6     Gene_7
"Pattern5" "Pattern2" "Pattern2" "Pattern2" "Pattern2" "Pattern2" "Pattern5"
    Gene_8     Gene_9    Gene_10
"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.

The FC and posterior FC for multiple condition data could be obtained by function `GetMultiFC`:

```
> MultiFC=GetMultiFC(MultiOut)
> str(MultiFC)

List of 3
 $ FCMat    : num [1:1000, 1:3] 1.211 1.013 0.947 1.09 1.065 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:1000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
  .. ..$ : chr [1:3] "C1OverC2" "C1OverC3" "C2OverC3"
 $ Log2FCMat: num [1:1000, 1:3] 0.2761 0.019 -0.079 0.1242 0.0902 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:1000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
  .. ..$ : chr [1:3] "C1OverC2" "C1OverC3" "C2OverC3"
 $ CondMeans: num [1:1000, 1:3] 496 1841 252 1787 808 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:1000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
  .. ..$ : Named chr [1:3] "C1" "C2" "C3"
  .. .. ..- attr(*, "names")= chr [1:3] "Condition1" "Condition2" "Condition3"
```

To generate a QQ-plot of the fitted Beta prior distribution and the $\hat{q}^C$'s within condition, a user could also use function `DenNHist` and `QQP`.

```
> par(mfrow=c(2,2))
> QQP(MultiOut)
```

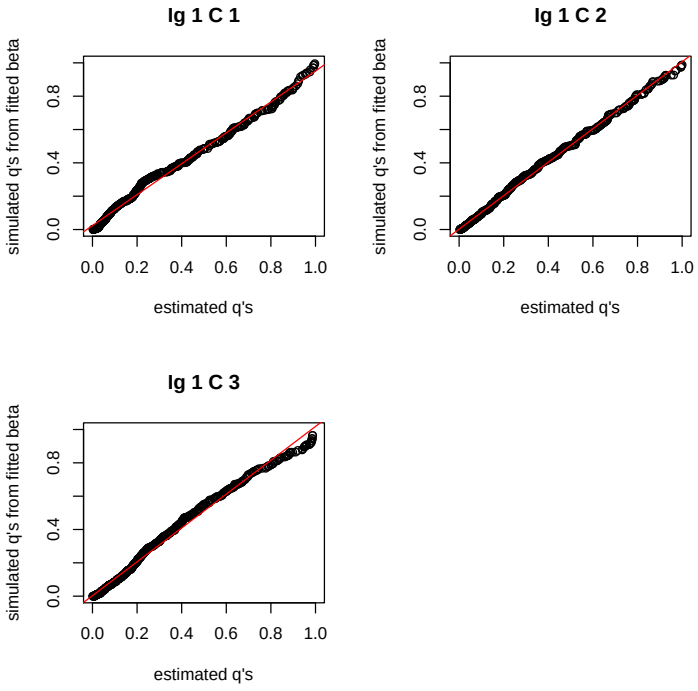

Figure 10: The QQ-plot of the fitted prior distribution within each condition

4.4 Working on isoform expression estimations with more than two conditions

Similar as 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.

```
> data(IsoMultiList)
> IsoMultiMat=IsoMultiList[[1]]
> IsoNames.Multi=IsoMultiList$IsoNames
> IsosGeneNames.Multi=IsoMultiList$IsosGeneNames
> IsoMultiSize=MedianNorm(IsoMultiMat)
> NgList.Multi=GetNg(IsoNames.Multi, IsosGeneNames.Multi)
> IsoNgTrun.Multi=NgList.Multi$IsoformNgTrun
> IsoMultiOut=EBMultiTest(IsoMultiMat,NgVector=IsoNgTrun.Multi,Conditions=Conditions,
+ AllParti=Parti.4Cond,
+ sizeFactors=IsoMultiSize, maxround=5)
```

```
> par(mfrow=c(2,2))
> DenNHist(MultiOut)
```

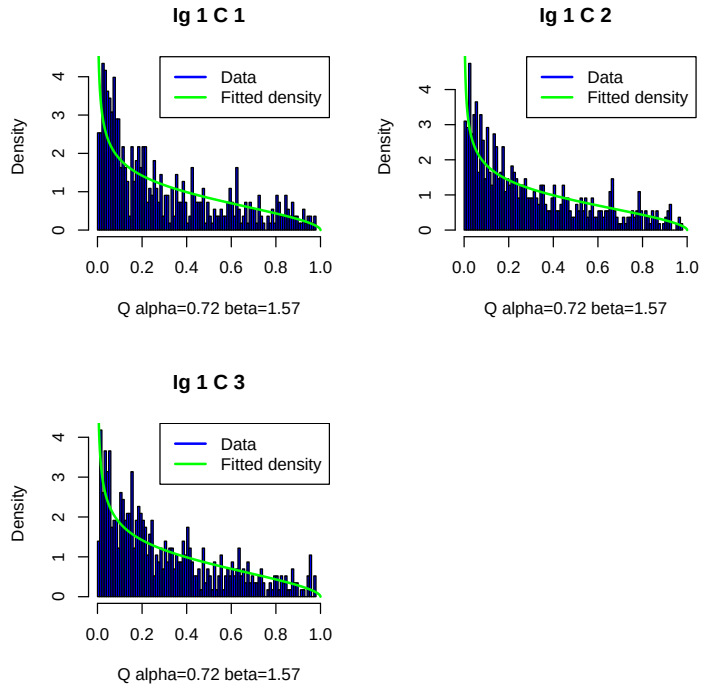

Figure 11: The prior distribution fitting within each condition

```
iteration 1 done
time 24.491
iteration 2 done
time 24.467
iteration 3 done
time 9.23300000000006
iteration 4 done
time 10.424
iteration 5 done
time 8.88199999999995

> IsoMultiPP=GetMultiPP(IsoMultiOut)
> names(MultiPP)

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

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

```
              Pattern1    Pattern2      Pattern3      Pattern8    Pattern15
Iso_1_1   4.491089e-34 0.999067581  1.357994e-33  2.728660e-34 9.324189e-04
Iso_1_2   2.645937e-14 0.998132248  3.172235e-15  2.816801e-16 1.867752e-03
Iso_1_3   1.270054e-43 0.978252880  4.196709e-38  6.630693e-45 2.174712e-02
Iso_1_4   1.585237e-36 0.994599329  6.156303e-30  1.145819e-32 5.400671e-03
Iso_1_5   0.000000e+00 1.000000000  0.000000e+00  0.000000e+00 2.830424e-41
Iso_1_6  4.942113e-228 0.003241202 5.563985e-214 1.657836e-183 9.967588e-01
Iso_1_7  7.900232e-115 0.997756178 3.895311e-110 2.050441e-105 2.243822e-03
Iso_1_8  5.430600e-130 0.851933966  1.834973e-96 1.987346e-111 1.480660e-01
Iso_1_9   4.182136e-45 0.804797880  5.085895e-47  3.408432e-42 1.952021e-01
Iso_1_10  6.948051e-08 0.997979087  3.026767e-08  5.123101e-08 2.020762e-03
```

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

```
    Iso_1_1     Iso_1_2     Iso_1_3     Iso_1_4     Iso_1_5     Iso_1_6
 "Pattern2"  "Pattern2"  "Pattern2"  "Pattern2"  "Pattern2" "Pattern15"
    Iso_1_7     Iso_1_8     Iso_1_9    Iso_1_10
 "Pattern2"  "Pattern2"  "Pattern2"  "Pattern2"
```

```
> IsoMultiPP$Patterns
```

```
          C1 C2 C3 C4
Pattern1   1  1  1  1
Pattern2   1  1  1  2
Pattern3   1  1  2  1
Pattern8   1  2  2  2
Pattern15  1  2  3  4
```

```
> IsoMultiFC=GetMultiFC(IsoMultiOut)
```

The FC and posterior FC for multiple condition data could be obtained by function `GetMultiFC`:
To generate a QQ-plot of the fitted Beta prior distribution and the $\hat{q}^C$'s within condition, a user could also use function `DenNHist` and `QQP`.

```
> Conditions=c("C1","C1","C2","C2","C3","C3","C4","C4")
> PosParti.4Cond=GetPatterns(Conditions)
> PosParti.4Cond
```

```
          C1 C2 C3 C4
Pattern1   1  1  1  1
Pattern2   1  1  1  2
Pattern3   1  1  2  1
Pattern4   1  1  2  2
Pattern5   1  2  1  1
Pattern6   1  2  1  2
Pattern7   1  2  2  1
Pattern8   1  2  2  2
Pattern9   1  1  2  3
Pattern10  1  2  1  3
Pattern11  1  2  2  3
Pattern12  1  2  3  1
Pattern13  1  2  3  2
Pattern14  1  2  3  3
Pattern15  1  2  3  4
```

```
> PlotPattern(PosParti.4Cond)
> Parti.4Cond=PosParti.4Cond[c(1,2,3,8,15),]
> Parti.4Cond
```

```
          C1 C2 C3 C4
Pattern1   1  1  1  1
Pattern2   1  1  1  2
Pattern3   1  1  2  1
Pattern8   1  2  2  2
Pattern15  1  2  3  4
```

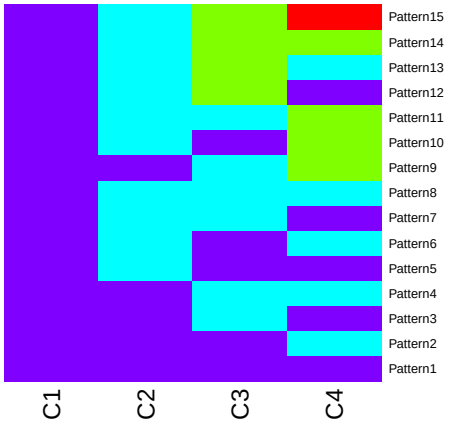

Figure 12: All possible patterns for 4 conditions

```
> par(mfrow=c(3,4))
> QQP(IsoMultiOut)
>
```

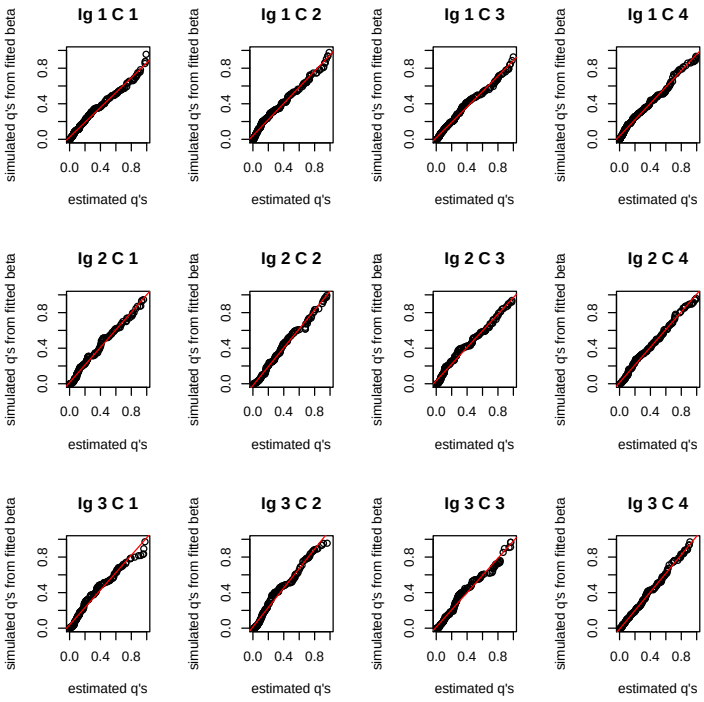

Figure 13: The QQ-plot of the fitted prior distribution within each condition and Ig group

```
> par(mfrow=c(3,4))
> DenNHist(IsoMultiOut)
```

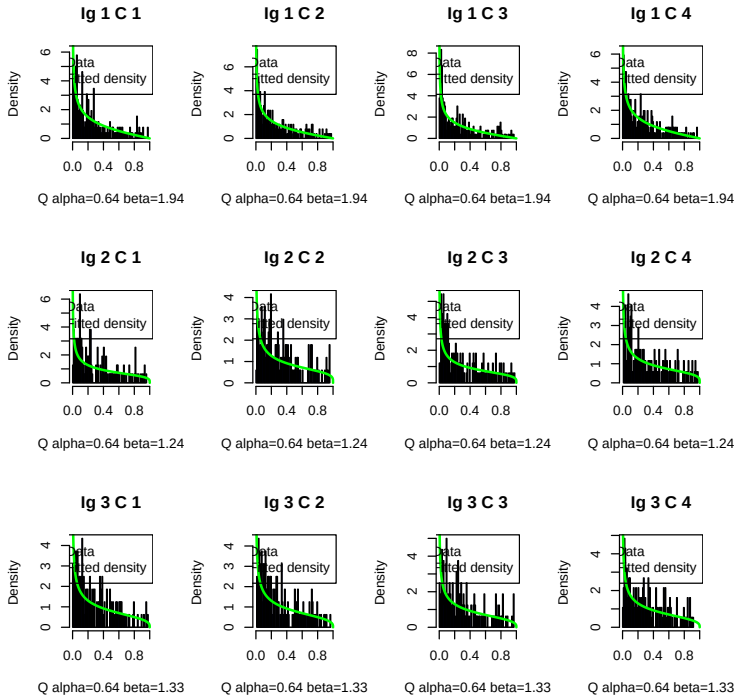

Figure 14: The prior distribution fitting within each condition and Ig group

4.5 Working without replicates

4.5.1 Gene counts with two conditions

Sometimes people are working on the data with no replicates within any condition. In this case, the transcript specific variance is hard to estimate while we only have one value for each gene within condition. Estimating the variance across condition is dangerous without the prior knowledge of being DE or EE. While working without replicates, EBSeq estimate the variance by pooling similar genes together. By defining `NumBin = 1000` (default in `EBTest`), EBSeq will group the genes with similar means together into 1,000 bins. With the assumption that no more than 50% genes are DE in the data set, We take genes whose FC are in the 25% - 75% quantile of all FC's as the candidate genes. For each bin, the bin-wise variance estimation would be the median of the across-condition variance estimations of the candidate genes within that bin. We use the across-condition variance estimations for the candidate genes and the bin-wise variance estimations of the host bin for the non-candidate genes.

Using the same data as in Section 4.1, we take only sample 1 from condition 1 and sample 6 from condition 2. Functions `MedianNorm`, `GetPPMat` and `PostFC` could also work on data without replicates.

```
> data(GeneMat)
> GeneMat.norep=GeneMat[,c(1,6)]
> Sizes.norep=MedianNorm(GeneMat.norep)
> EBOut.norep=EBTest(Data=GeneMat.norep,
+ Conditions=as.factor(rep(c("C1","C2"))),
+ sizeFactors=Sizes.norep, maxround=5)

Remove transcripts with all zero
No Replicate - estimate phi 0.00368711916180493
iteration 1 done
time 11.481
iteration 2 done
time 5.40200000000004
iteration 3 done
time 5.01599999999996
iteration 4 done
time 4.24599999999998
iteration 5 done
time 4.57299999999998

> PP.norep=GetPPMat(EBOut.norep)
> DEfound.norep=rownames(PP.norep)[which(PP.norep[,"PPDE"]>=.95)]
> GeneFC.norep=PostFC(EBOut.norep)
```

4.5.2 Isoform counts with two conditions

We also take sample 1 and sample 6 in the data we used in Section 4.2. Example codes are shown below.

```
> data(IsoList)
> IsoMat=IsoList$IsoMat
> IsoNames=IsoList$IsoNames
> IsosGeneNames=IsoList$IsosGeneNames
> NgList=GetNg(IsoNames, IsosGeneNames)
> IsoNgTrun=NgList$IsoformNgTrun
> IsoMat.norep=IsoMat[,c(1,6)]
> IsoSizes.norep=MedianNorm(IsoMat.norep)
> IsoEBOut.norep=EBTest(Data=IsoMat.norep, NgVector=IsoNgTrun,
+ Conditions=as.factor(c("C1","C2")),
+ sizeFactors=IsoSizes.norep, maxround=5)

Remove transcripts with all zero
No Replicate - estimate phi 0.0154195839877669
iteration 1 done
time 24.758
iteration 2 done
time 4.66200000000003
iteration 3 done
time 5.21399999999994
iteration 4 done
time 5.15700000000004
iteration 5 done
time 4.73300000000006

> IsoPP.norep=GetPPMat(IsoEBOut.norep)
> IsoDE.norep=rownames(IsoPP.norep)[which(IsoPP.norep[,"PPDE"]>=.95)]
> IsoFC.norep=PostFC(IsoEBOut.norep)
```

4.5.3 Gene counts with more than two conditions

We take one sample from each condition (sample 1, 3 and 5) in the data we used in Section 4.3. Example codes are shown below.

```
> data(MultiGeneMat)
> MultiGeneMat.norep=MultiGeneMat[,c(1,3,5)]
> Conditions=c("C1","C2","C3")
> PosParti=GetPatterns(Conditions)
> Parti=PosParti[-3,]
> MultiSize.norep=MedianNorm(MultiGeneMat.norep)
> MultiOut.norep=EBMultiTest(MultiGeneMat.norep,
+ NgVector=NULL,Conditions=Conditions,
+ AllParti=Parti, sizeFactors=MultiSize.norep,
+ maxround=5)

No Replicate - estimate phi 0.00277960151525143
iteration 1 done
```

```
time 6.31100000000004
iteration 2 done
time 3.56999999999994
iteration 3 done
time 6.48500000000001
iteration 4 done
time 3.92900000000009
iteration 5 done
time 2.75399999999991
```

```
> MultiPP.norep=GetMultiPP(MultiOut.norep)
> MultiFC.norep=GetMultiFC(MultiOut.norep)
```

4.5.4 Isoform counts with more than two conditions

We take one sample from each condition (sample 1, 3, 5 and 7) in the data we used in Section 4.4. Example codes are shown below.

```
> data(IsoMultiList)
> IsoMultiMat=IsoMultiList[[1]]
> IsoNames.Multi=IsoMultiList$IsoNames
> IsosGeneNames.Multi=IsoMultiList$IsosGeneNames
> IsoMultiMat.norep=IsoMultiMat[,c(1,3,5,7)]
> IsoMultiSize.norep=MedianNorm(IsoMultiMat.norep)
> NgList.Multi=GetNg(IsoNames.Multi, IsosGeneNames.Multi)
> IsoNgTrun.Multi=NgList.Multi$IsoformNgTrun
> Conditions=c("C1","C2","C3","C4")
> PosParti.4Cond=GetPatterns(Conditions)
> PosParti.4Cond
```

```
          C1 C2 C3 C4
Pattern1   1  1  1  1
Pattern2   1  1  1  2
Pattern3   1  1  2  1
Pattern4   1  1  2  2
Pattern5   1  2  1  1
Pattern6   1  2  1  2
Pattern7   1  2  2  1
Pattern8   1  2  2  2
Pattern9   1  1  2  3
Pattern10  1  2  1  3
Pattern11  1  2  2  3
Pattern12  1  2  3  1
Pattern13  1  2  3  2
Pattern14  1  2  3  3
Pattern15  1  2  3  4
```

```
> Parti.4Cond=PosParti.4Cond[c(1,2,3,8,15),]
> Parti.4Cond

          C1 C2 C3 C4
Pattern1   1  1  1  1
Pattern2   1  1  1  2
Pattern3   1  1  2  1
Pattern8   1  2  2  2
Pattern15  1  2  3  4

> IsoMultiOut.norep=EBMultiTest(IsoMultiMat.norep,
+ NgVector=IsoNgTrun.Multi,Conditions=Conditions,
+ AllParti=Parti.4Cond, sizeFactors=IsoMultiSize.norep,
+ maxround=5)

No Replicate - estimate phi 0.00244688865176316
iteration 1 done
time 20.26
iteration 2 done
time 20.14
iteration 3 done
time 10.567
iteration 4 done
time 9.94200000000001
iteration 5 done
time 7.11299999999994

> IsoMultiPP.norep=GetMultiPP(IsoMultiOut.norep)
> IsoMultiFC.norep=GetMultiFC(IsoMultiOut.norep)
```

References

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

[2] J H Bullard, E A Purdom, K D Hansen, and S Dudoit. Evaluation of statistical methods for normalization and differential expression in mrna-seq experiments. *BMC Bioinformatics*, 11:94, 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] 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.