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

Ning Leng, John Dawson, and Christina Kendziorski

November 8, 2022

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.*, 2013 [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.*, 2013 [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.*, 2013 [3], the model is also useful for identifying DE genes. We will briefly detail the model in Section 3 and then describe the flow of analysis in Section 4 for both isoform and gene-level inference.

2 Citing this software

Please cite the following article when reporting results from the software.
Leng, N., J.A. Dawson, J.A. Thomson, V. Ruotti, A.I. Rissman, B.M.G. Smits, J.D. Haag, M.N. Gould, R.M. Stewart, and C. Kendziorski. EBSeq: An empirical Bayes hierarchical model for inference in RNA-seq experiments, *Bioinformatics*, 2013.

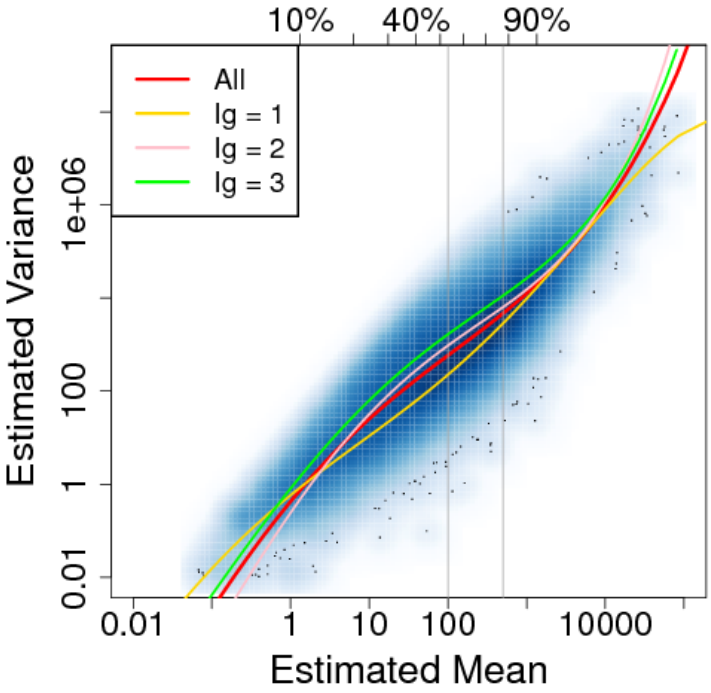

Figure 1: Empirical variance vs. mean for each isoform profiled in the ESCs vs iPSCs experiment detailed in the Case Study section of Leng *et al.*, 2013 [3]. A spline fit to all isoforms is shown in red with splines fit within the $I_g = 1$, $I_g = 2$, and $I_g = 3$ isoform groups shown in yellow, pink, and green, respectively.

3 The Model

3.1 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:

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

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

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

3.2 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:

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

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:

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

3.3 Getting a false discovery rate (FDR) controlled list of genes or isoforms

To obtain a list of DE genes with false discovery rate (FDR) controlled at α in an experiment comparing two biological conditions, the genes with posterior probability of being DE (PPDE) greater than 1 - α should be used. For example, the genes with PPDE>=0.95 make up the list of DE genes with target FDR controlled at 5%. With more than two biological conditions, there are multiple DE patterns (see Section 3.2). To obtain a list of genes in a specific DE pattern with target FDR α, a user should take the genes with posterior probability of being in that pattern greater than 1 - α. Isoform-based lists are obtained in the same way.

4 Quick Start

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

```
> library(EBSeq)
```

4.1 Gene level DE analysis (two conditions)

4.1.1 Required input

Data: The object `Data` should be a $G - by - S$ matrix containing the expression values for each gene and each sample, where G is the number of genes and S is the number of samples. These values should exhibit raw counts, without normalization across samples. Counts of this nature may be obtained from RSEM [4], Cufflinks [6], or a similar approach.

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 1,000 rows of genes and 10 columns of samples. The genes are named `Gene_1, Gene_2 ...`

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

```
 num [1:1000, 1:10] 1879 24 3291 97 485 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:1000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
  ..$ : NULL
```

4.1.2 Library size factor

As detailed in Section 3, 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])

4.1.3 Running EBSeq on gene expression estimates

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 samples 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 sample. It could be obtained by summing up the total number of reads within each sample, Median Normalization [1], scaling normalization [5], Upper-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 are usually required. Convergence should always be checked (see Section 5.1.3 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)
```

The list of DE genes and the posterior probabilities of being DE are obtained as follows

```
> EBDERes=GetDEResults(EBOut, FDR=0.05)
> str(EBDERes$DEfound)

 chr [1:95] "Gene_1" "Gene_2" "Gene_3" "Gene_4" "Gene_5" "Gene_6" "Gene_7" ...

> head(EBDERes$PPMat)

               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 4.850156e-10    1

> str(EBDERes$Status)

 Named chr [1:1000] "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" ...
 - attr(*, "names")= chr [1:1000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
```

`EBDERes$DEfound` is a list of genes identified with 5% FDR. EBSeq found 95 genes. The matrix `EBDERes$PPMat` contains two columns `PPEE` and `PPDE`, corresponding to the posterior probabilities of being EE or DE for each gene. `EBDERes$Status` contains each gene's status called by EBSeq.
Note the `GetDEResults()` was incorporated in EBSeq since version 1.7.1. By using the default settings, the number of genes identified in any given analysis may differ slightly from the previous version. The updated algorithm is more robust to outliers and transcripts with low variance. To obtain results that are comparable to results from earlier versions of EBSeq ($\leq$ 1.7.0), a user may set `Method="classic"` in `GetDEResults()` function, or use the `GetPPMat()` function.

4.2 Isoform level DE analysis (two conditions)

4.2.1 Required inputs

Data: The object `Data` should be a $I - by - S$ matrix containing the expression values for each isoform and each sample, where I is the number of isoforms and S is the number of sample. As in the gene-level analysis, 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 1,200 simulated isoforms. In which `IsoList$IsoMat` is a data matrix containing 1,200 rows of isoforms and 10 columns of samples; `IsoList$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 [1:1200, 1:10] 176 789 1300 474 1061 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:1200] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
  .. ..$ : NULL
 $ IsoNames     : chr [1:1200] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
 $ IsosGeneNames: chr [1:1200] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
```

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

```
 num [1:1200, 1:10] 176 789 1300 474 1061 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:1200] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
  ..$ : NULL
```

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

4.2.2 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)
```

4.2.3 The I_g vector

While working on isoform level data, EBSeq fits different prior parameters for different uncertainty groups (defined as I_g groups). The default setting to define the uncertainty groups consists of using the number of isoforms the host gene contains (N_g) for each isoform. The default settings will provide three uncertainty groups:

$I_g = 1$ group: Isoforms with $N_g = 1$;
$I_g = 2$ group: Isoforms with $N_g = 2$;
$I_g = 3$ group: Isoforms with $N_g \geq 3$.

The N_g and I_g group assignment can be obtained 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(1:3,201:203,601:603)]
```

```
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 5.2.

4.2.4 Running EBSeq on isoform expression estimates

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 will likely be required in practice (see Section 5.2.4 for details).

```
> IsoEBOut=EBTest(Data=IsoMat, NgVector=IsoNgTrun,
+ Conditions=as.factor(rep(c("C1","C2"),each=5)),sizeFactors=IsoSizes, maxround=5)
> IsoEBDERes=GetDEResults(IsoEBOut, FDR=0.05)
> str(IsoEBDERes$DEfound)

 chr [1:104] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" "Iso_1_5" "Iso_1_6" ...

> head(IsoEBDERes$PPMat)

        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

> str(IsoEBDERes$Status)

 Named chr [1:1200] "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" ...
 - attr(*, "names")= chr [1:1200] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
```

We see that EBSeq found 104 DE isoforms at the target FDR of 0.05.
Note the `GetDEResults()` was incorporated in EBSeq since version 1.7.1. By using the default settings, the number of transcripts identified in any given analysis may differ slightly from the previous version. The updated algorithm is more robust to outliers and transcripts with low variance. To obtain results that are comparable to results from earlier versions of EBSeq ($\leq$ 1.7.0), a user may set `Method="classic"` in `GetDEResults()` function, or use the `GetPPMat()` function.

4.3 Gene level DE analysis (more than two conditions)

The object `MultiGeneMat` is a matrix containing 500 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 [1:500, 1:6] 411 268 768 1853 878 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:500] "Gene_1" "Gene_3" "Gene_5" "Gene_7" ...
  ..$ : NULL
```

In analysis 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 3, 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`. 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)
```

The posterior probability 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   8.574912e-94 0.3989333  5.103008e-72 0.60106672
Gene_3  9.716720e-164 0.9694232 6.670885e-109 0.03057680
Gene_5   6.282756e-26 0.9336809  6.342497e-20 0.06631906
Gene_7   0.000000e+00 0.5563573  0.000000e+00 0.44364273
Gene_9   5.044830e-16 0.9437746  2.008227e-15 0.05622539
Gene_11  1.956384e-11 0.9369721  2.157793e-12 0.06302785
Gene_13  1.429796e-08 0.7296838  6.389130e-10 0.27031623
Gene_15  3.504222e-47 0.9691736  8.501776e-41 0.03082640
Gene_17 3.354675e-184 0.6564080 4.546629e-133 0.34359199
Gene_19  1.754055e-37 0.9044564  1.309553e-24 0.09554364
```

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

```
    Gene_1     Gene_3     Gene_5     Gene_7     Gene_9    Gene_11    Gene_13
"Pattern5" "Pattern2" "Pattern2" "Pattern2" "Pattern2" "Pattern2" "Pattern2"
   Gene_15    Gene_17    Gene_19
"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$PP` provides the posterior probability of being in each pattern for every gene. `MultiPP$MAP` provides the most likely pattern of each gene based on the posterior probabilities. `MultiPP$Patterns` provides the details of the patterns.

4.4 Isoform level DE analysis (more than two conditions)

Similar to `IsoList`, the object `IsoMultiList` is an object containing the isoform expression estimates matrix, the isoform names, and the gene names of the isoforms' host genes. `IsoMultiList$IsoMultiMat` contains 300 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 to Section 4.2, the function `MedianNorm` and `GetNg` could be used for normalization and calculating the 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 perform the test:

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

The posterior probability 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   3.533233e-32 0.999882138  3.408808e-33  2.143838e-34 1.178620e-04
Iso_1_2   4.231331e-14 0.999826487  1.573392e-16  5.848567e-18 1.735129e-04
Iso_1_3   5.633772e-47 0.992627423  5.963569e-42  5.644910e-50 7.372577e-03
Iso_1_4   4.248398e-35 0.998959777  1.983567e-30  5.054181e-33 1.040223e-03
Iso_1_5   0.000000e+00 1.000000000  0.000000e+00  0.000000e+00 1.584343e-41
Iso_1_6  1.509151e-232 0.002646919 3.147566e-220 6.720686e-188 9.973531e-01
Iso_1_7  2.835263e-138 0.999439469 7.548859e-133 1.613556e-128 5.605313e-04
Iso_1_8  9.654898e-139 0.963893542 3.709303e-105 5.626105e-120 3.610646e-02
Iso_1_9   1.947187e-47 0.957423511  1.073683e-50  3.868129e-46 4.257649e-02
Iso_1_10  7.904509e-08 0.999790300  9.178739e-10  9.386672e-10 2.096196e-04
```

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

5 More detailed examples

5.1 Gene level DE analysis (two conditions)

5.1.1 Running EBSeq on simulated gene expression estimates

EBSeq is applied as described in Section 4.1.3.

```
> data(GeneMat)
> Sizes=MedianNorm(GeneMat)
> EBOut=EBTest(Data=GeneMat,
+ Conditions=as.factor(rep(c("C1","C2"),each=5)),sizeFactors=Sizes, maxround=5)
> EBDERes=GetDEResults(EBOut, FDR=0.05)

> EBDERes=GetDEResults(EBOut, FDR=0.05)
> str(EBDERes$DEfound)

 chr [1:95] "Gene_1" "Gene_2" "Gene_3" "Gene_4" "Gene_5" "Gene_6" "Gene_7" ...

> head(EBDERes$PPMat)

               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 4.850156e-10    1

> str(EBDERes$Status)

 Named chr [1:1000] "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" "DE" ...
 - attr(*, "names")= chr [1:1000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
```

EBSeq found 95 DE genes at a target FDR of 0.05.

5.1.2 Calculating FC

The function `PostFC` may be used to calculate the Fold Change (FC) of the raw data as well as the posterior FC of the normalized data. Figure 2 shows the FC vs. Posterior FC on 1,000 gene expression estimates. The genes are ranked by their cross-condition mean (adjusted by the normalization factors). The posterior FC tends to shrink genes with low expressions (small rank); in this case the differences are minor.

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

List of 3
 $ PostFC   : Named num [1:1000] 0.237 0.241 4.127 4.245 3.91 ...
  ..- attr(*, "names")= chr [1:1000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
 $ RealFC   : Named num [1:1000] 0.237 0.239 4.128 4.28 3.918 ...
  ..- attr(*, "names")= chr [1:1000] "Gene_1" "Gene_2" "Gene_3" "Gene_4" ...
 $ Direction: chr "C1 Over C2"

> PlotPostVsRawFC(EBOut,GeneFC)
```

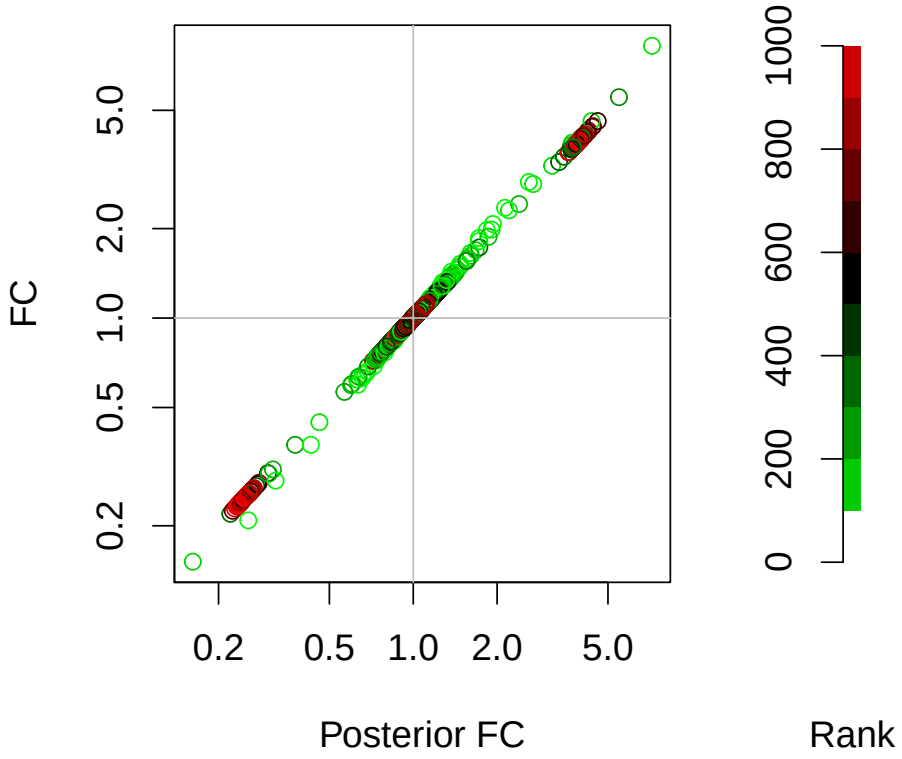

Figure 2: FC vs. Posterior FC for 1,000 gene expression estimates

5.1.3 Checking convergence

As detailed in Section 3, we assume the prior distribution of q_g^C is $Beta(\alpha, \beta)$. The EM algorithm is used to estimate 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 0.8101264
iter2 0.8076690
iter3 0.8071266
iter4 0.8074156
iter5 0.8065706

> EBOut$Beta

           Ng1
iter1 1.569830
iter2 1.580238
iter3 1.577823
iter4 1.579587
iter5 1.575817

> EBOut$P

           [,1]
iter1 0.1715152
iter2 0.1323722
iter3 0.1266082
iter4 0.1260407
iter5 0.1258774
```

In this case the differences between the 4th and 5th iterations are always less than 0.01.

5.1.4 Checking the model fit and other diagnostics

As noted in Leng *et al.*, 2013 [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 conditions, which indicates that the Beta prior is appropriate.

```
> par(mfrow=c(1,2))
> QQP(EBOut)
```

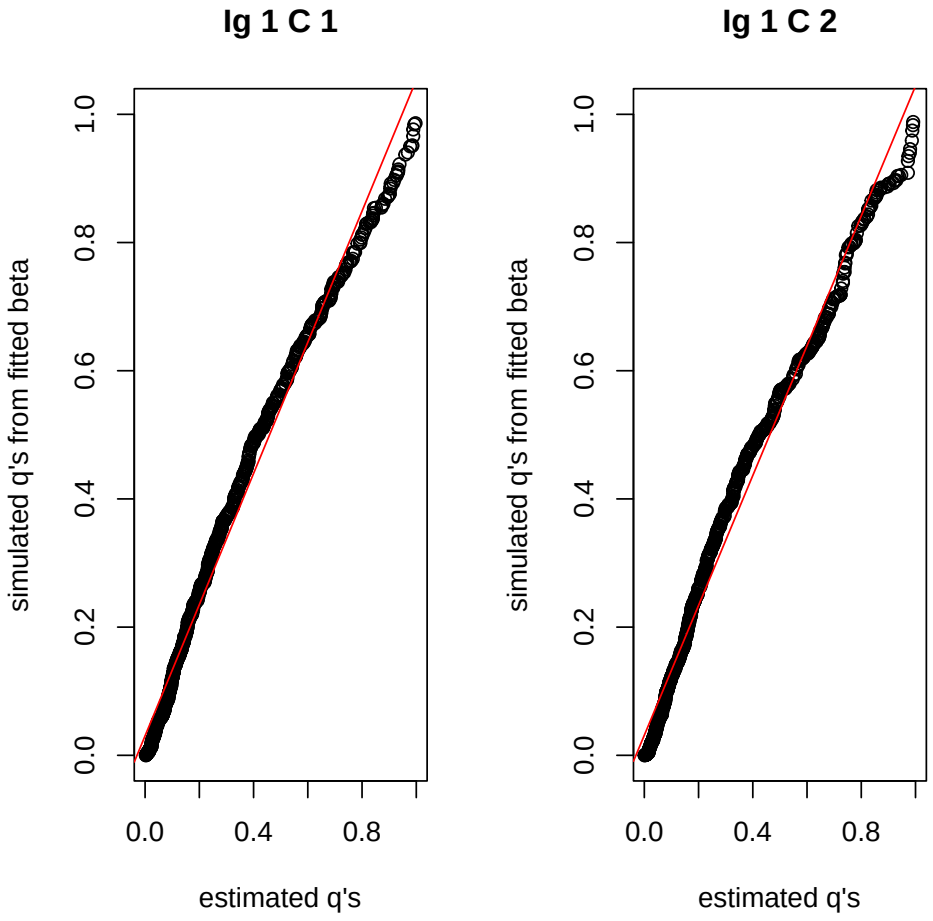

Figure 3: QQ-plots for checking the assumption of a Beta prior (upper panels) as well as the model fit using data from condition 1 and condition 2 (lower panels)

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(1,2))
> DenNHist(EBOut)
```

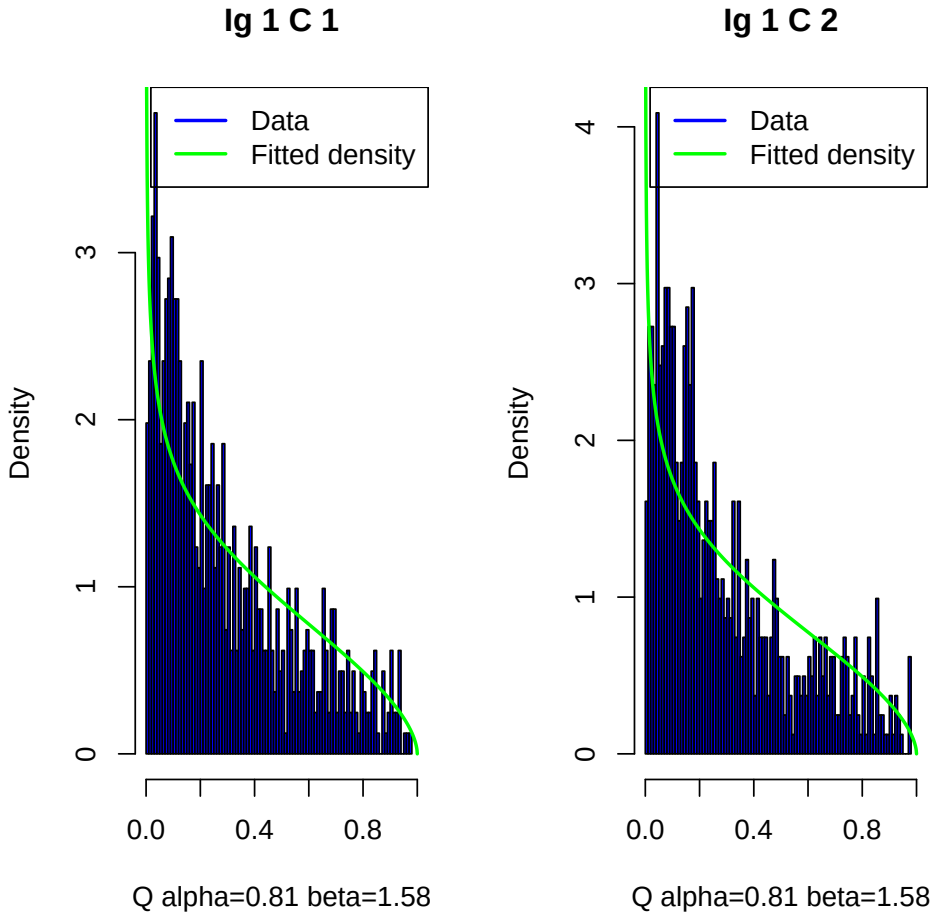

Figure 4: Density plots for checking the model fit using data from condition 1 and condition 2

5.2 Isoform level DE analysis (two conditions)

5.2.1 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_200`; The isoforms in the $I_g = 2$ group belong to genes `Gene_201, ..., Gene_400`; and isoforms in the $I_g = 3$ group belong to `Gene_401, ..., Gene_600`.

```
> data(IsoList)
> 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(1:3,201:203,601:603)]

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
```

The output of `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 I_g group assignments. The default number of groups is 3, which means the isoforms with N_g greater than 3 will be assigned to $I_g = 3$ group. 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.

5.2.2 Using mappability ambiguity clusters instead of the I_g vector when the gene-isoform relationship is unknown

When 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 a k-means algorithm on the unmapability scores that are provided by RSEM. A user can read in the mapping ambiguity cluster information using:

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

Where `"output_name.ngvec"` is the output file obtained from RSEM function rsem-generate-ngvector. 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.

5.2.3 Running EBSeq on simulated isoform expression estimates

EBSeq can be applied as described in Section 4.2.4.

```
> IsoSizes=MedianNorm(IsoMat)
> IsoEBOut=EBTest(Data=IsoMat, NgVector=IsoNgTrun,
+ Conditions=as.factor(rep(c("C1","C2"),each=5)),sizeFactors=IsoSizes, maxround=5)
> IsoEBDERes=GetDEResults(IsoEBOut, FDR=0.05)

> str(IsoEBDERes)

List of 3
 $ DEfound: chr [1:104] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
 $ PPMat  : num [1:1200, 1:2] 0 0 0 0 0 0 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:1200] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
  .. ..$ : chr [1:2] "PPEE" "PPDE"
 $ Status : Named chr [1:1200] "DE" "DE" "DE" "DE" ...
  ..- attr(*, "names")= chr [1:1200] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
```

We see that EBSeq found 104 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 [1:1200] 0.286 0.281 3.554 0.305 3.756 ...
  ..- attr(*, "names")= chr [1:1200] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
 $ RealFC   : Named num [1:1200] 0.285 0.281 3.556 0.305 3.759 ...
  ..- attr(*, "names")= chr [1:1200] "Iso_1_1" "Iso_1_2" "Iso_1_3" "Iso_1_4" ...
 $ Direction: chr "C1 Over C2"
```

5.2.4 Checking convergence

For isoform level data, we assume the prior distribution of q_{gi}^C is $Beta(\alpha, \beta^{I_g})$. As in Section 5.1.3, the optimized parameters at each iteration may be obtained as follows (recall we are using 5 iterations for demonstration purposes):

```
> IsoEBOut$Alpha
```

```
           [,1]
iter1 0.7060900
iter2 0.7126779
iter3 0.7112128
iter4 0.7103945
iter5 0.7101250
```

```
> IsoEBOut$Beta
```

```
           Ng1      Ng2      Ng3
iter1 1.592741 2.285690 2.952408
iter2 1.630033 2.394687 3.115835
iter3 1.636704 2.392290 3.111714
iter4 1.637340 2.386478 3.107970
iter5 1.633741 2.383986 3.105093
```

```
> IsoEBOut$P
```

```
           [,1]
iter1 0.2107669
iter2 0.1628520
iter3 0.1503443
iter4 0.1465272
iter5 0.1457991
```

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 changing less than 10^{-2} or 10^{-3}. In practice, we require changes less than 10^{-3} to declare convergence.

5.2.5 Checking the model fit and other diagnostics

In Leng *et al.*, 2013[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 `PolyFitPlot` 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)
```

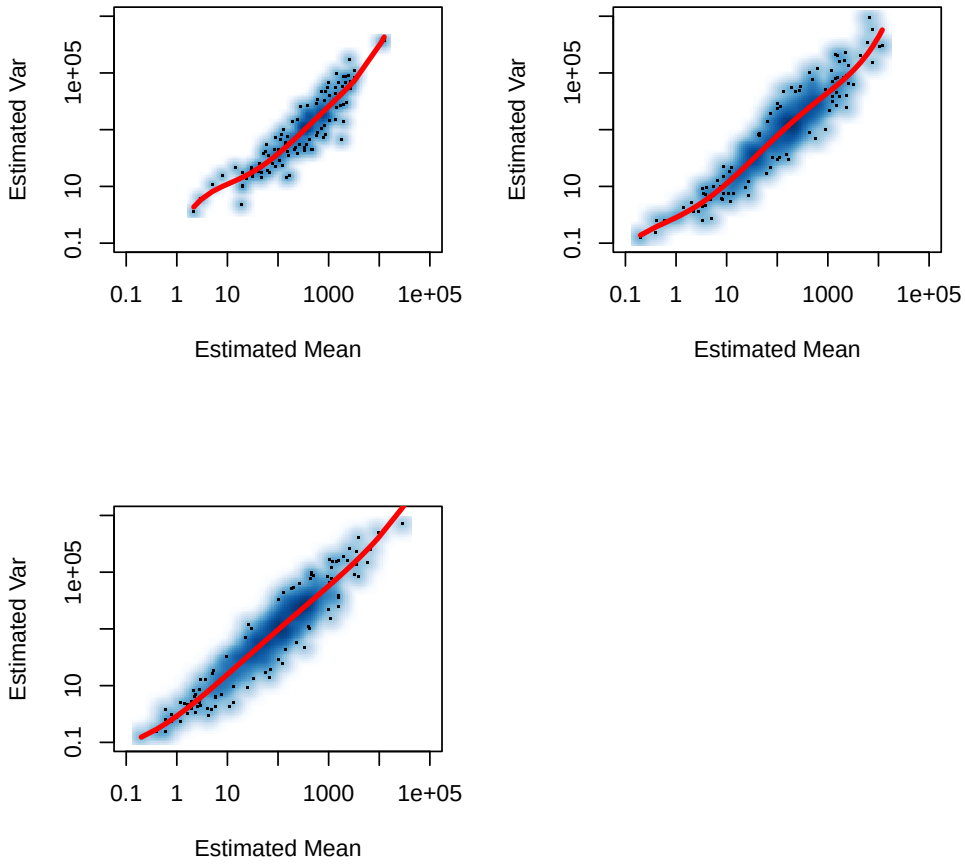

Figure 5: The mean-variance fitting plot for each Ng group

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)
```

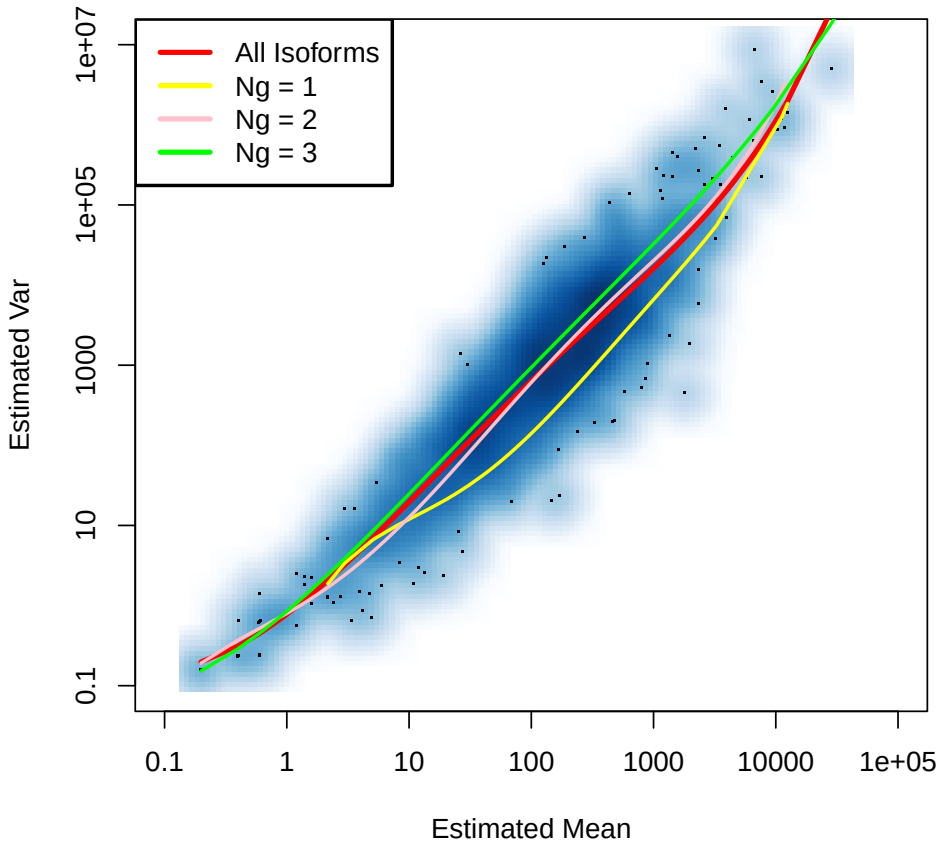

Figure 6: The mean-variance plot for each Ng group

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 6 panels (as shown in Figure 7).

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

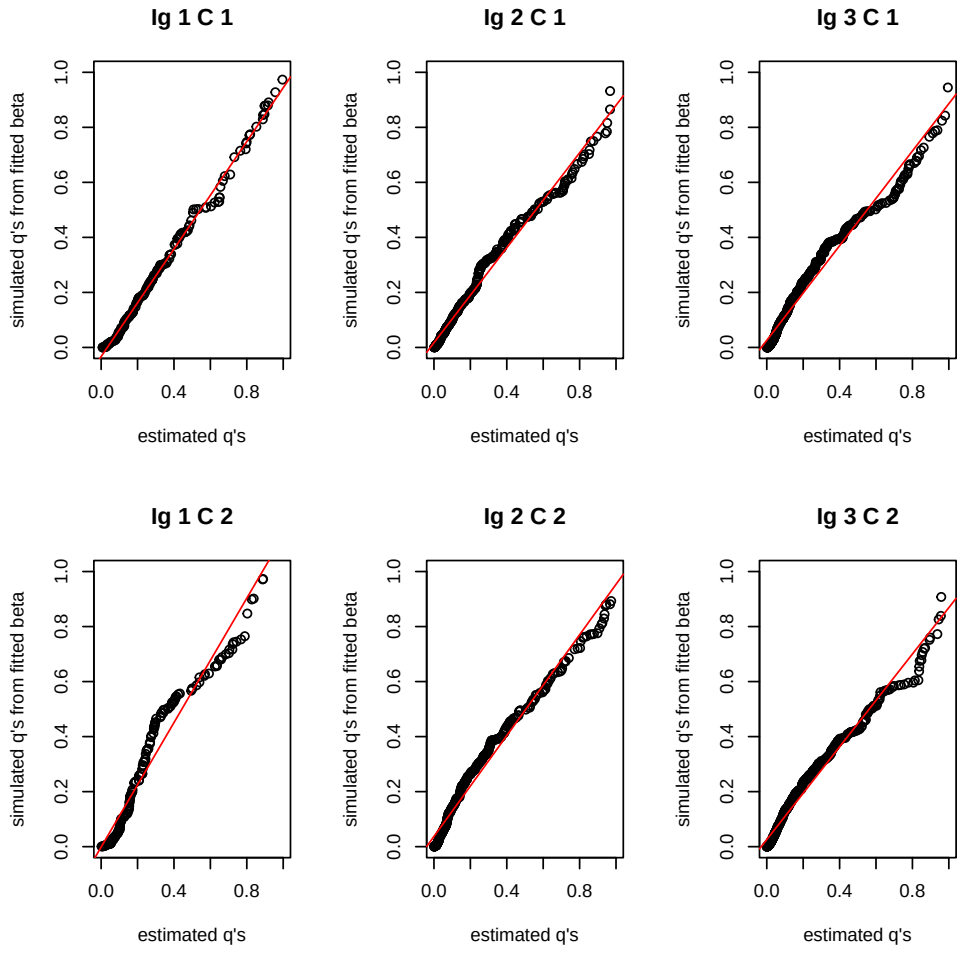

Figure 7: QQ-plots of the fitted prior distributions within each condition and each Ig group

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)
```

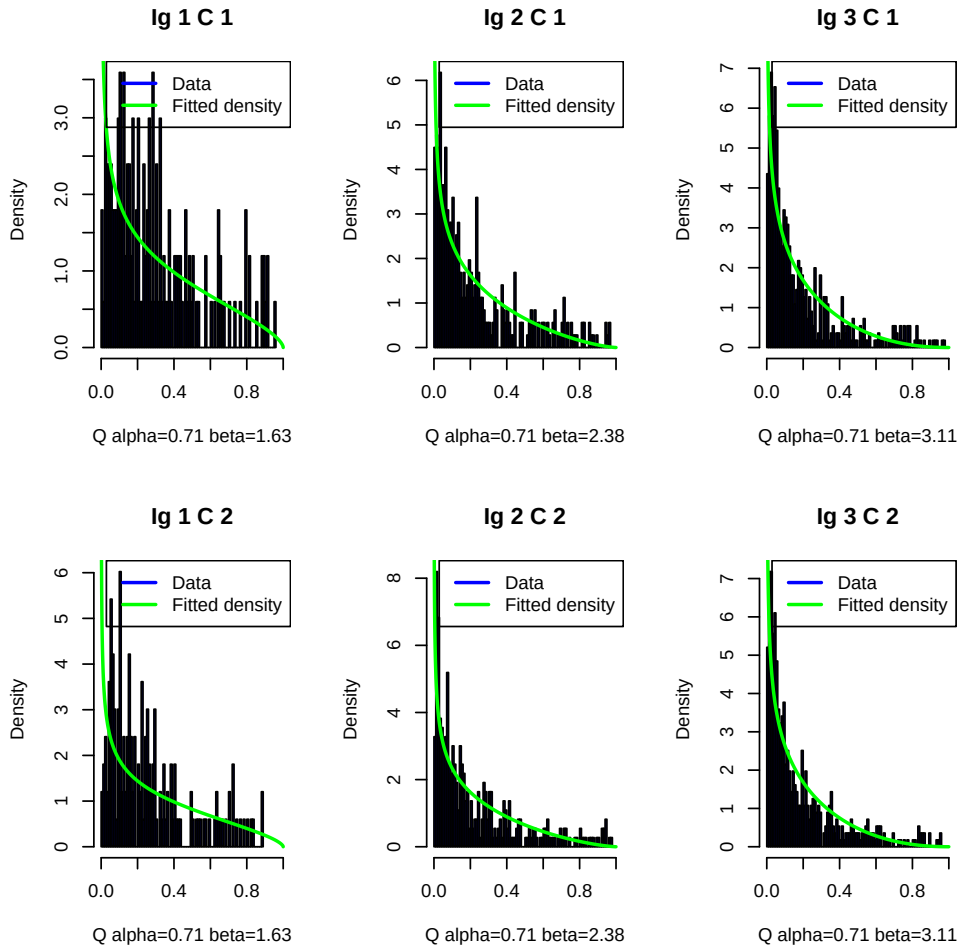

Figure 8: Prior distribution fit within each condition and each Ig group. (Note only a small set of isoforms are considered here for demonstration. Better fitting should be expected while using full set of isoforms.)

5.3 Gene level DE analysis (more than two conditions)

As described in Section 4.3, the function `GetPatterns` allows the user to generate all possible patterns given the conditions. To visualize the patterns, the function `PlotPattern` may be used.

```
> 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)
```

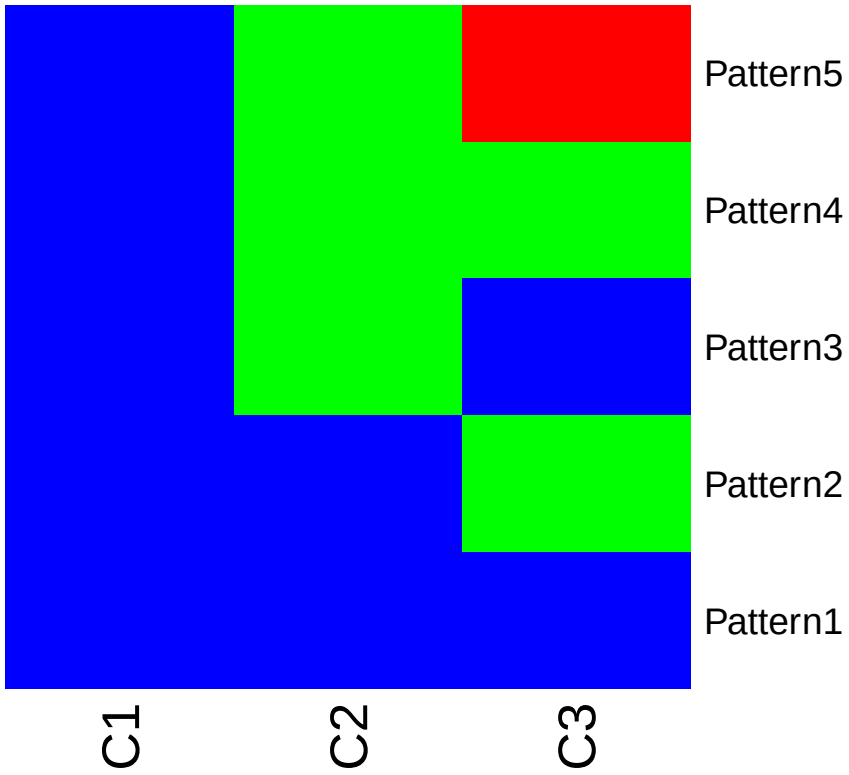

Figure 9: All possible patterns

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`.

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

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   8.574912e-94 0.3989333  5.103008e-72 0.60106672
Gene_3  9.716720e-164 0.9694232 6.670885e-109 0.03057680
Gene_5   6.282756e-26 0.9336809  6.342497e-20 0.06631906
Gene_7   0.000000e+00 0.5563573  0.000000e+00 0.44364273
Gene_9   5.044830e-16 0.9437746  2.008227e-15 0.05622539
Gene_11  1.956384e-11 0.9369721  2.157793e-12 0.06302785
Gene_13  1.429796e-08 0.7296838  6.389130e-10 0.27031623
Gene_15  3.504222e-47 0.9691736  8.501776e-41 0.03082640
Gene_17 3.354675e-184 0.6564080 4.546629e-133 0.34359199
Gene_19  1.754055e-37 0.9044564  1.309553e-24 0.09554364
```

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

```
    Gene_1     Gene_3     Gene_5     Gene_7     Gene_9    Gene_11    Gene_13
"Pattern5" "Pattern2" "Pattern2" "Pattern2" "Pattern2" "Pattern2" "Pattern2"
   Gene_15    Gene_17    Gene_19
"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$PP` provides the posterior probability of being in each pattern for every gene. `MultiPP$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 can be obtained by the function `GetMultiFC`:

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

List of 6
 $ FCMat         : num [1:500, 1:3] 1.217 0.951 1.069 0.923 0.983 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:500] "Gene_1" "Gene_3" "Gene_5" "Gene_7" ...
  .. ..$ : chr [1:3] "C1OverC2" "C1OverC3" "C2OverC3"
 $ Log2FCMat     : num [1:500, 1:3] 0.2828 -0.0724 0.0969 -0.1151 -0.0251 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:500] "Gene_1" "Gene_3" "Gene_5" "Gene_7" ...
  .. ..$ : chr [1:3] "C1OverC2" "C1OverC3" "C2OverC3"
 $ PostFCMat     : num [1:500, 1:3] 1.216 0.951 1.069 0.923 0.983 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:500] "Gene_1" "Gene_3" "Gene_5" "Gene_7" ...
  .. ..$ : chr [1:3] "C1OverC2" "C1OverC3" "C2OverC3"
 $ Log2PostFCMat : num [1:500, 1:3] 0.2819 -0.072 0.0967 -0.115 -0.0251 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:500] "Gene_1" "Gene_3" "Gene_5" "Gene_7" ...
  .. ..$ : chr [1:3] "C1OverC2" "C1OverC3" "C2OverC3"
 $ CondMeans     : num [1:500, 1:3] 499 253 813 1843 753 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:500] "Gene_1" "Gene_3" "Gene_5" "Gene_7" ...
  .. ..$ : chr [1:3] "C1" "C2" "C3"
 $ ConditionOrder: 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)
```

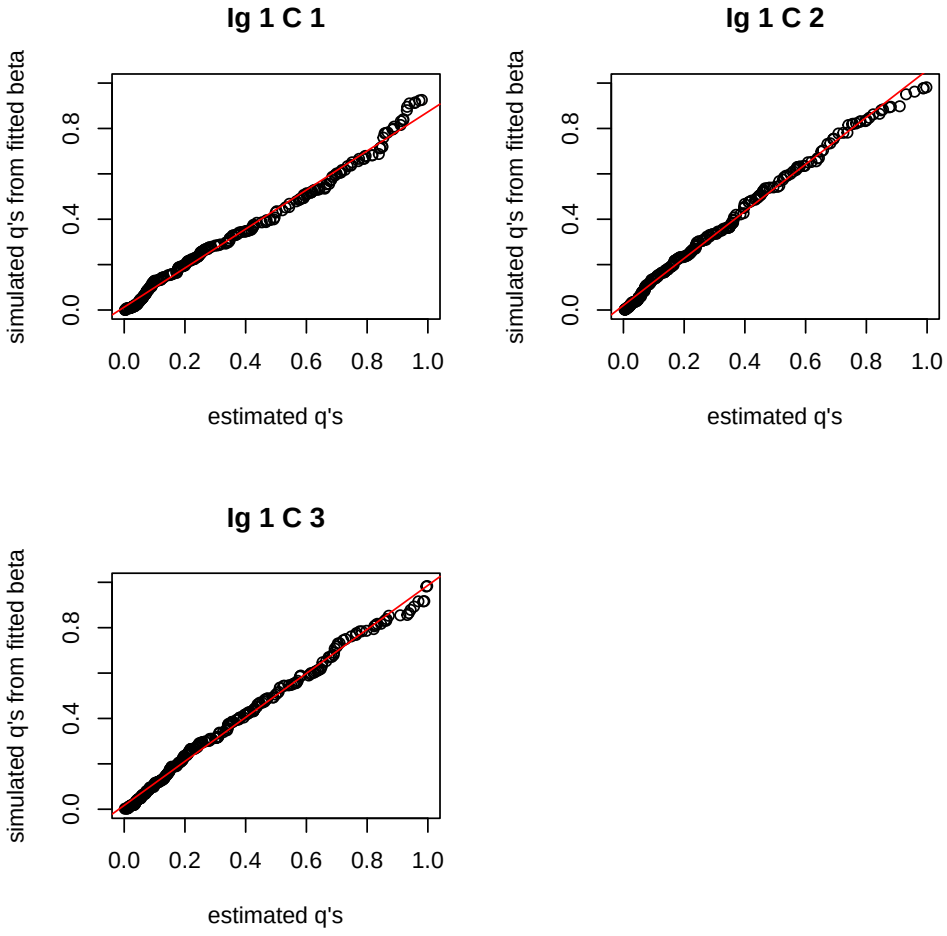

Figure 10: QQ-plots of the fitted prior distributions within each condition and each Ig group

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

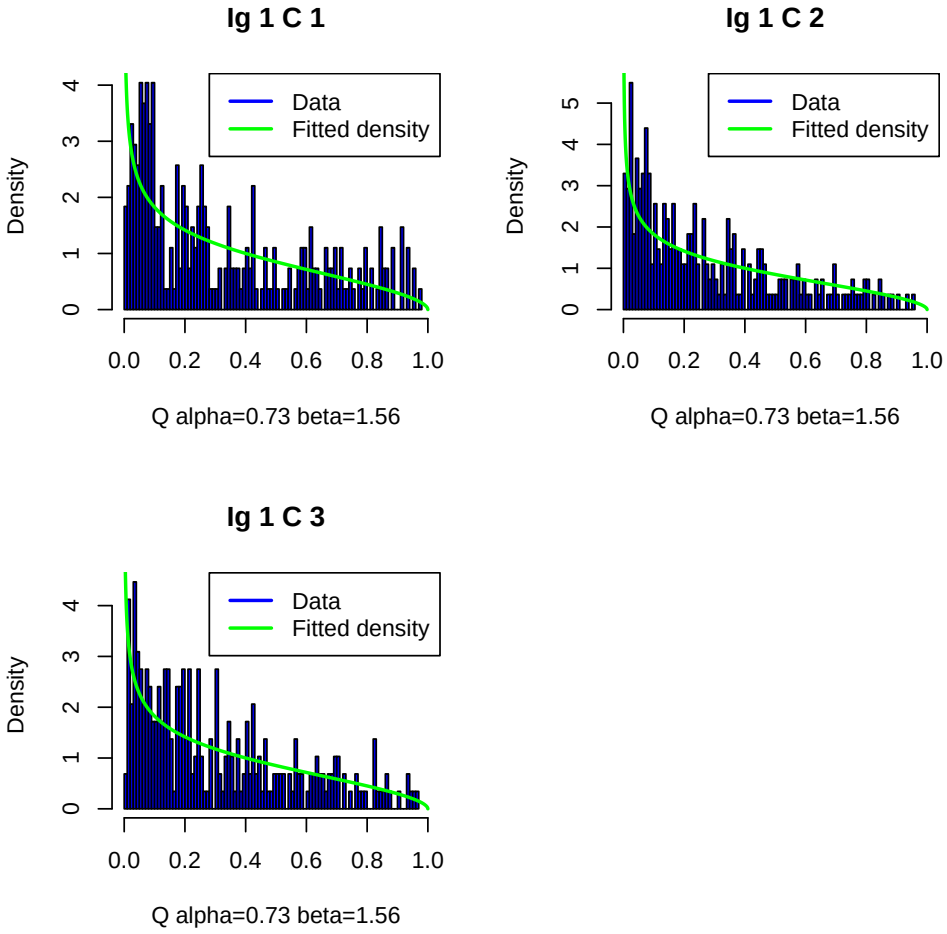

Figure 11: Prior distributions fit within each condition. (Note only a small set of genes are considered here for demonstration. Better fitting should be expected while using full set of genes.)

5.4 Isoform level DE analysis (more than two conditions)

Similar to Section 4.3, the function `GetPatterns` allows a user to generate all possible patterns given the conditions. To visualize the patterns, the function `PlotPattern` may be used.

```
> 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
```

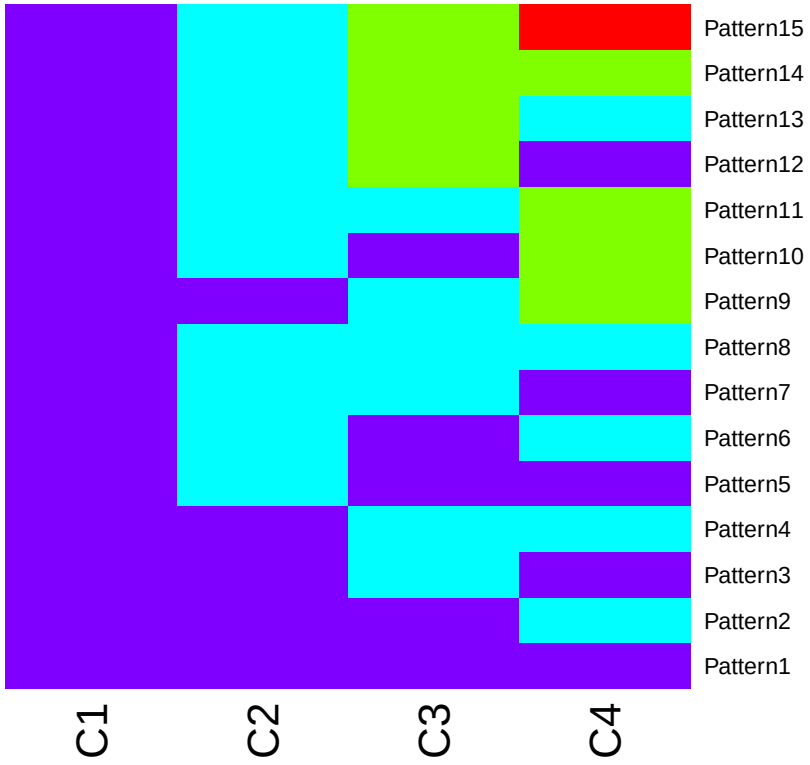

Figure 12: All possible patterns for 4 conditions

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

> names(MultiPP)

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

> IsoMultiPP$PP[1:10,]

              Pattern1    Pattern2      Pattern3      Pattern8    Pattern15
Iso_1_1   3.533233e-32 0.999882138  3.408808e-33  2.143838e-34 1.178620e-04
Iso_1_2   4.231331e-14 0.999826487  1.573392e-16  5.848567e-18 1.735129e-04
Iso_1_3   5.633772e-47 0.992627423  5.963569e-42  5.644910e-50 7.372577e-03
Iso_1_4   4.248398e-35 0.998959777  1.983567e-30  5.054181e-33 1.040223e-03
Iso_1_5   0.000000e+00 1.000000000  0.000000e+00  0.000000e+00 1.584343e-41
Iso_1_6  1.509151e-232 0.002646919 3.147566e-220 6.720686e-188 9.973531e-01
Iso_1_7  2.835263e-138 0.999439469 7.548859e-133 1.613556e-128 5.605313e-04
Iso_1_8  9.654898e-139 0.963893542 3.709303e-105 5.626105e-120 3.610646e-02
Iso_1_9   1.947187e-47 0.957423511  1.073683e-50  3.868129e-46 4.257649e-02
Iso_1_10  7.904509e-08 0.999790300  9.178739e-10  9.386672e-10 2.096196e-04

> 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 can be obtained by the 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 the functions `DenNHist` and `QQP`.

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

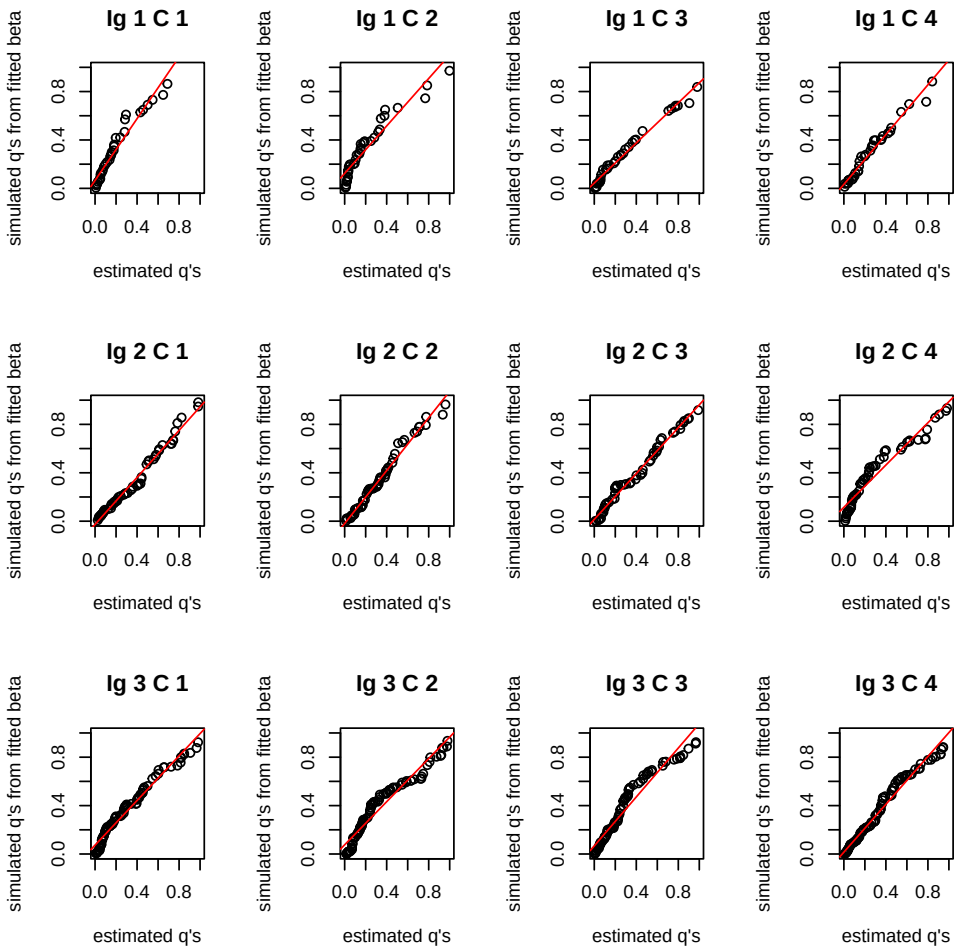

Figure 13: QQ-plots of the fitted prior distributions within each condition and Ig group. (Note only a small set of isoforms are considered here for demonstration. Better fitting should be expected while using full set of isoforms.)

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

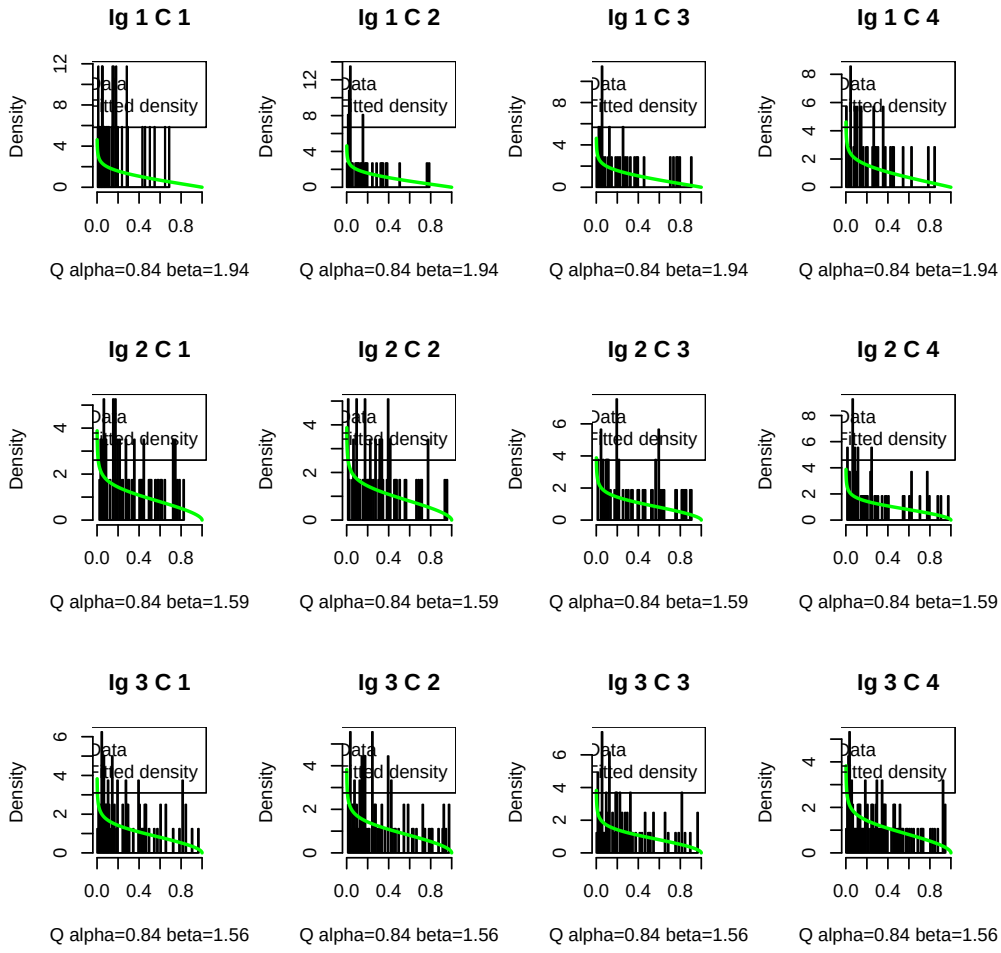

Figure 14: Prior distributions fit within each condition and Ig group. (Note only a small set of isoforms are considered here for demonstration. Better fitting should be expected while using full set of isoforms.)

5.5 Working without replicates

When replicates are not available, it is difficult to estimate the transcript specific variance. In this case, EBSeq estimates the variance by pooling similar genes together. Specifically, we take genes with FC in the 25% - 75% quantile of all FC's as candidate genes. By defining `NumBin = 1000` (default in `EBTest`), EBSeq will group genes with similar means into 1,000 bins. For each candidate gene, we use the across-condition variance estimate as its variance estimate. For each bin, the bin-wise variance estimation is taken to be the median of the across-condition variance estimates of the candidate genes within that bin. For each non-candidate gene, we use the bin-wise variance estimate of the host bin (the bin containing this gene) as its variance estimate. This approach works well when there are no more than 50% DE genes in the data set.

5.5.1 Gene counts with two conditions

To generate a data set with no replicates, we take the first sample of each condition. For example, using the data from Section 5.1, we take sample 1 from condition 1 and sample 6 from condition 2. Functions `MedianNorm`, `GetDEResults` and `PostFC` may be used 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)

Removing transcripts with 100 th quantile < = 0
999 transcripts will be tested

> EBDERes.norep=GetDEResults(EBOut.norep)
> GeneFC.norep=PostFC(EBOut.norep)
```

5.5.2 Isoform counts with two conditions

To generate an isoform level data set with no replicates, we also take sample 1 and sample 6 in the data we used in Section 5.2. 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)

Removing transcripts with 100 th quantile < = 0
1192 transcripts will be tested

> IsoEBDERes.norep=GetDEResults(IsoEBOut.norep)
> IsoFC.norep=PostFC(IsoEBOut.norep)
```

5.5.3 Gene counts with more than two conditions

To generate a data set with multiple conditions and no replicates, we take the first sample from each condition (sample 1, 3 and 5) in the data we used in Section 5.3. 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)
> MultiPP.norep=GetMultiPP(MultiOut.norep)
> MultiFC.norep=GetMultiFC(MultiOut.norep)
```

5.5.4 Isoform counts with more than two conditions

To generate an isoform level data set with multiple conditions and no replicates, we take the first sample from each condition (sample 1, 3, 5 and 7) in the data we used in Section 5.4. 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)
> IsoMultiPP.norep=GetMultiPP(IsoMultiOut.norep)
> IsoMultiFC.norep=GetMultiFC(IsoMultiOut.norep)
```

6 EBSeq pipelines and extensions

6.1 RSEM-EBSeq pipeline: from raw reads to differential expression analysis results

EBSeq is coupled with RSEM [4] as an RSEM-EBSeq pipeline which provides quantification and DE testing on both gene and isoform levels.

For more details, see `http://deweylab.biostat.wisc.edu/rsem/README.html#de`

6.2 EBSeq interface: A user-friendly graphical interface for differetial expression analysis

EBSeq interface provides a graphical interface implementation for users who are not familiar with the R programming language. It takes .xls, .xlsx and .csv files as input. Additional packages need be downloaded; they may be found at `http://www.biostat.wisc.edu/~ningleng/EBSeq_Package/EBSeq_Interface/`

6.3 EBSeq Galaxy tool shed

EBSeq tool shed contains EBSeq wrappers for a local Galaxy implementation. For more details, see `http://www.biostat.wisc.edu/~ningleng/EBSeq_Package/EBSeq_Galaxy_toolshed/`

7 Acknowledgment

We would like to thank Haolin Xu for checking the package and proofreading the vignette.

8 News

2014-1-30: In EBSeq 1.3.3, the default setting of EBTest function will remove low expressed genes (genes whose 75th quantile of normalized counts is less than 10) before identifying DE genes. These two thresholds can be changed in EBTest function. Because low expressed genes are disproportionately noisy, removing these genes prior to downstream analyses can improve model fitting and increase robustness (e.g. by removing outliers).

2014-5-22: In EBSeq 1.5.2, numerical approximations are implemented to deal with underflow. The underflow is likely due to large number of samples.

2015-1-29: In EBSeq 1.7.1, EBSeq incorporates a new function GetDEResults() which may be used to obtain a list of transcripts under a target FDR in a two-condition experiment. The results obtained by applying this function with its default setting will be more robust to transcripts with low variance and potential outliers. By using the default settings in this function, the number of genes identified in any given analysis may differ slightly from the previous version (1.7.0 or order). To obtain results that are comparable to results from earlier versions of EBSeq (1.7.0 or older), a user may set Method="classic" in GetDEResults() function, or use the original GetPPMat() function. The GeneDEResults() function also allows a user to modify thresholds to target genes/isoforms with a pre-specified posterior fold change.

Also, in EBSeq 1.7.1, the default settings in EBTest() and EBMultiTest() function will only remove transcripts with all 0's (instead of removing transcripts with 75th quantile less than 10 in version 1.3.3-1.7.0). To obtain a list of transcripts comparable to the results generated by EBSeq version 1.3.3-1.7.0, a user may change Qtrm = 0.75 and QtrmCut = 10 when applying EBTest() or EBMultiTest() function.

9 Common Q and A

9.1 Read in data

csv file:

```
In=read.csv("FileName", stringsAsFactors=F, row.names=1, header=T)
Data=data.matrix(In)
```

txt file:

```
In=read.table("FileName", stringsAsFactors=F, row.names=1, header=T)
Data=data.matrix(In)
```

Check `str(Data)` and make sure it is a matrix instead of data frame. You may need to play around with the `row.names` and `header` option depends on how the input file was generated.

9.2 GetDEResults() function not found

You may on an earlier version of EBSeq. The GetDEResults function was introduced since version 1.7.1. The latest release version could be found at:

```
http://www.bioconductor.org/packages/release/bioc/html/EBSeq.html
```

The latest devel version:

```
http://www.bioconductor.org/packages/devel/bioc/html/EBSeq.html
```

And you may check your package version by typing `packageVersion("EBSeq")`.

9.3 Visualizing DE genes/isoforms

To generate a heatmap, you may consider the heatmap.2 function in gplots package. For example, you may run

```
heatmap.2(NormalizedMatrix[GenesOfInterest,], scale="row", trace="none", Colv=F)
```

The normalized matrix may be obtained from `GetNormalizedMat()` function.

9.4 My favorite gene/isoform has NA in PP (status "NoTest")

The NoTest status comes from two sources:

1) In version 1.3.3-1.7.0, using the default parameter settings of EBMultiTest(), the function will not test on genes with more than 75% values ≤ 10 to ensure better model fitting. To disable this filter, you may set Qtrm=1 and QtrmCut=0.

2) numerical over/underflow in R. That happens when the within condition variance is extremely large or small. we did implemented a numerical approximation step to calculate the approximated PP for these genes with over/underflow. Here we use 10^{-10} to approximate the parameter p in the NB distribution for these genes (we set it to a small value since we want to cover more over/underflow genes with low within-condition

variation). You may try to tune this value (to a larger value) in the approximation by setting `ApproxVal` in `EBTest()` or `EBMultiTest()` function.

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] Ning Leng, John A Dawson, James A Thomson, Victor Ruotti, Anna I Rissman, Bart MG Smits, Jill D Haag, Michael N Gould, Ron M Stewart, and Christina Kendziorski. Ebseq: an empirical bayes hierarchical model for inference in rna-seq experiments. *Bioinformatics*, 29(8):1035–1043, 2013.

[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 A Oshlack. 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.