# BBSeq: A method to handle RNA-seq count data

Yihui Zhou, Fred A. Wright

March 26, 2011

## 1 Introduction

This vignette describes how to use BBSeq to evaluate differential expression using count data and to measure the effect of design variables or covariates. BBSeq has been tuned to correspond to the tpical data structure of RNA-Seq count data, which typically is based on millions of reads and thousands of genes/transcripts. BBSeq is currently in beta evaluation form, and subject to continual improvement in the interface.

## 2 Preparation

At a minimum, BBSeq requires a *count data* matrix and a *predictor* matrix. Other data can be prepared and added to accomplish specific tasks, as described below.

- Data `data.Y`: the count data should be a matrix of counts, where rows represent genes and columns represent samples. Here the term "gene" can refer to a transcript or transcriptional isoform. If the count data are in a dataframe, BBSeq will convert to a matrix. However, the user should be alert to any potential errors that might arise in type conversion.
- Annotation `data.anno`: A data frame providing annotation data for each gene. The rows should match the rows of the count data matrix exactly. A typical annotation data frame will consist of columns including a trasncript/gene ID, chromosome, and gene symbol.
- Library size `lib.size`: the total number of reads mapped for each sample. If this information is not provided, BBSeq will calculate a library size by summing each column of the count matrix.
- Predictor type `categorical`: a vector which describes if the predictors are to be treated as categorical (and therefore treated as having multiple unordered levels). This vector is only used for the optional `X.builder` function.
- Predictor matrix `X`: a standard regression design matrix, which typically contains an initial column of `1`s and other indicator columns for the design variables or covariates. For multi-level factors (as for ANOVA analysis), the `X.builder` function simplifies the generation of `X`. Continuous covariates (or any for which the relationship between the count data and the covariate is assumed log-linear) should be entered directly as a column of `X`.

## 3 Definition and Notation

The data consist of an $m \times n$ matrix $Y$. Each entry $y_{ij}$ represents the transcriptional count for the $i$th gene in the $j$th sample. We will use $\theta_{ij}$ to denote the probability that a single read in sample $j$ maps to gene $i$, and $\theta_{i.}$ as the $n$-vector of these probabilities. The beta-binomial models $\theta$ as a random variable, which produces the overdispersion. Reads within the same sample are assumed independent. $X$ will denote an $n \times p$ design matrix, consisting of indicator variables for experimental conditions and any desired covariates. The effect of $X$ on gene $i$ is modeled as

$$logit(E(\theta_{i.})) = \log\left(\frac{E(\theta_{i.})}{1 - E(\theta_{i.})}\right) = XB_i \tag{1}$$

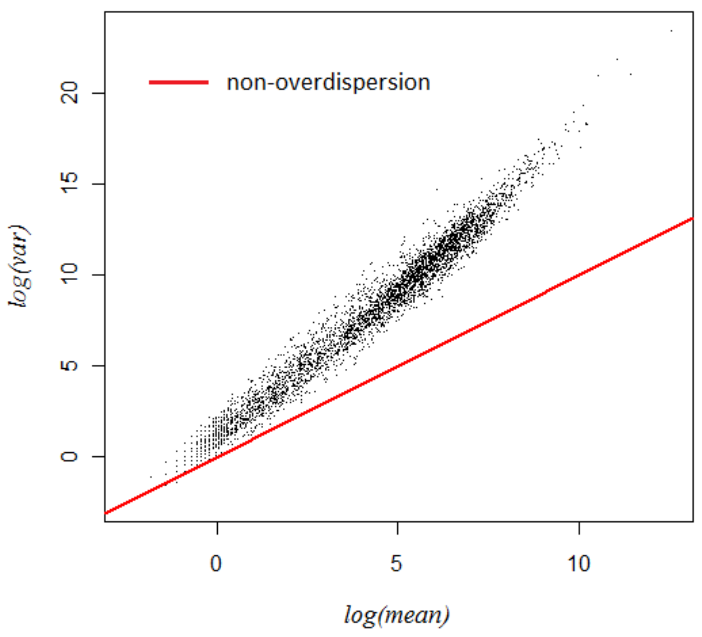

Figure 1: The mean-variance relationship in the CEU data suggests a mean-overdispersion relationship

for the $p \times 1$ matrix of regression coefficients $B_i = [\beta_{0,i}, .., \beta_{p-1,i}]^T$. $\theta$ follows a Beta distribution, parameterized so that its variance is $\phi E(\theta)(1 - E(\theta))$. Values $\phi > 0$ correspond to overdispersion compared to the binomial, after considering design effects.

## 4 BBSeq: Examples from real data

We randomly selected 5000 genes from Montgomery et al. 2010 ([Montgomery, 2010]) for subset of 12 samples (out of 60 samples). For illustrative purposes, we ensured that all of the genes on the X chromosome were retained among the 5000 genes. As a discrete covariate, we will use the gender/sex information (6 males vs. 6 females). We also simulate a continuous covariate, which we'll refer to as the `score`. Together these two covariates are used to illustrate the BBSeq approach, in which the effect of covariates can be examined individually or together.

### 4.1 Overdispersion in RNA-Seq count data

To illustrate the overdispersion patterns typical of RNA-Seq counts, we show in Figure 1 the log(variance) vs. log (mean) for these data. Note the increasing overdispersion (the excess variance above the unit line) as the mean increases. This relationship is directly modeled using the "constrained" approach described below.

### 4.2 Two group comparisons

#### 4.2.1 Read the data into R

First, we'll read an example dataset and annotation files into R.

```
library(BBSeq)
data(data.Y)
data(data.anno)
```

Note: If data.Y is a dataframe, it will be automatically converted to a matrix of type numeric. BBSeq expects integers, and so the user should take care that the matrix conversion does not produce unintended consequences that might arise from (e.g.) character values in the data frame.

If we want to explicitly convert to a matrix, that's easy.

```
> data.Y = as.matrix(data.Y)
```

The annotation data object contains a reference ID, the chromosome, and the gene symbol:

```
> data.anno[1:5,]
     ref.id  chr  gene
1 NM_000032 chrX ALAS2
2 NM_000033 chrX ABCD1
3 NM_000044 chrX    AR
4 NM_000047 chrX  ARSE
5 NM_000052 chrX ATP7A
```

### 4.2.2 Construct the design matrix

In `data.Y`, samples 1-3 and 6-9 are from females, while the remaining samples are male. We will use males as a "reference" - i.e. the corresponding column of `X` will be 1 for female, 0 for male.

```
> X = cbind(1,rep(rep(c(1,0),each=3),2))
```

### 4.2.3 A simple beta-binomial generalized linear model (the *free* model)

Here we fit the free model. The 5000 genes take about 35 minutes on a modern PC.

```
> output = free.estimate(data.Y,X)
> beta.free = output$betahat.free
> beta.free[1:5,]
             [,1]         [,2]
[1,] -20.25020046   4.641884825
[2,]  -9.37800267  -0.297522032
[3,] -13.19392779  -0.059594806
[4,] -15.11446472  -0.95397864
[5,] -10.03989835  -0.050189874
> p.free = output$p.free
> psi.free = output$psi.free
```

Note: the $\beta$ estimation from `free.estimate` is an $m \times n$ matrix; while the input matrix of intial values for $\beta$ in `constrained.estimate` is n × m. Please translate the output from `free.estimate` first before plugging into next function.

### 4.2.4 Mean-overdisersion modeling (constrained model)

We empirically find there is a strong relationship between $X \times Beta.free$ and $psi.free$ (see figure 2). We fit the relationship between these values, using $Beta.free$ and $psi.free$ to perform regression modeling of the relationship, and (for $Beta.free$) as starting values for the constrained likelihood optimization.

```
> Beta.free = t(beta.free)
> x = apply(X%*%as.matrix(Beta.free),2,mean)
> y = psi.free
> plot(x,y)
```

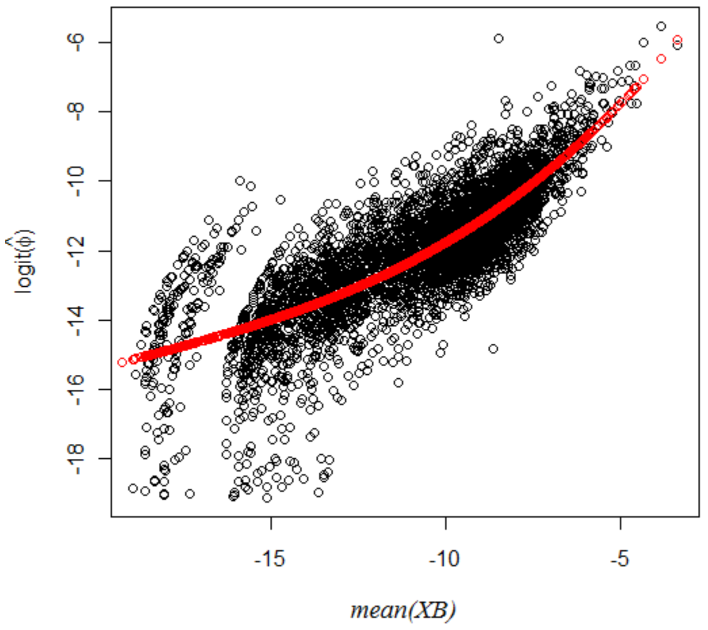

Figure 2: Fitted values for the mean-overdispersion relationship

```
> out.model = constrained.estimate(data.Y,X,gn=3,output$betahat.free,psi.free=output$psi.free)
> names(out.model)
[1] "betahat.model" "bvar.model"    "p.model"
> beta.constrained = out.model$betahat.model
> p.constrained = out.model$p.model
```

We "flag" the gene as suspicious if its maximum library-scaled value per gene is more than $c$ times as great as its second-largest value, provided that the second-largest value is greater than zero. The default value of $c$ is 5.0.

```
> flag = outlier.flag(data.Y)
> mean(flag)
[1] 0.0208
```

so about 2.08% of the genes are flagged. If genes are considered highly significant by the constrained model but are flagged, they should be subject to further scrutiny.

### 4.2.5 Generating a $p$-value data frame

Now we obtain the $p$-value data frame which combines the annotation information, $p$-values, and the flag. The top 5 genes are shown.

```
> p.df = data.frame(data.anno,p.free=output$p.free , p.constrained=out.model$p.model,
+ flag=outlier.flag(data.Y))
> dim(p.matrix)
[1] 5000  6
### the top 5 genes:
> p.matrix[order(p.free)[1:5],]
           ref.id  chr   gene       p.free p.constrained flag
```

```
435      NM_004979 chrX  KCND1 0.0003226803  4.562947e-03    0
3985 NM_001080124 chr2  CASP8 0.0003591038  2.522168e-02    0
722      NR_001564 chrX   XIST 0.0007598883  1.978532e-05    0
349      NM_001412 chrX EIF1AX 0.0011548680  1.088131e-03    0
70       NM_001007 chrX  RPS4X 0.0011646955  2.815752e-04    0
```

## 4.3 The case with two covariates/factors

### 4.3.1 Building the design matrix

This `X.builder` function returns four design matrices, for two factors. `X.builder$D1` is the design matrix if we only consider gender effect; `X.builder$Q1` is the one for considering the (simulated) "score" effect only; `X.builder$Big` gives the full design matrix including both gender and score. The last one `X.builder$Int` corresponds to the intercept only. We simulate a `score` vector with length 12, to use it as a continuous covariate.

```
> set.seed(5)
> score = rnorm(12,mean=0,sd=1)
> categorical = c(1,0)
> a1 = rep(rep(c(1,0),each=3),2)
> a2 = score
> predictor = cbind(a1,a2)
> X.builder(predictor,categorical)
$D1
      X  
 [1,] 1 1
 [2,] 1 1
 [3,] 1 1
 [4,] 1 0
 [5,] 1 0
 [6,] 1 0
 [7,] 1 1
 [8,] 1 1
 [9,] 1 1
[10,] 1 0
[11,] 1 0
[12,] 1 0

$Q1
      X           
 [1,] 1 -0.84085548
 [2,] 1  1.38435934
 [3,] 1 -1.25549186
 [4,] 1  0.07014277
 [5,] 1  1.71144087
 [6,] 1 -0.60290798
 [7,] 1 -0.47216639
 [8,] 1 -0.63537131
 [9,] 1 -0.28577363
[10,] 1  0.13810822
[11,] 1  1.22763034
[12,] 1 -0.80177945

$Big
```

```
      X
 [1,] 1 1 -0.84085548
 [2,] 1 1  1.38435934
 [3,] 1 1 -1.25549186
 [4,] 1 0  0.07014277
 [5,] 1 0  1.71144087
 [6,] 1 0 -0.60290798
 [7,] 1 1 -0.47216639
 [8,] 1 1 -0.63537131
 [9,] 1 1 -0.28577363
[10,] 1 0  0.13810822
[11,] 1 0  1.22763034
[12,] 1 0 -0.80177945

$Int
      [,1]
 [1,]    1
 [2,]    1
 [3,]    1
 [4,]    1
 [5,]    1
 [6,]    1
 [7,]    1
 [8,]    1
 [9,]    1
[10,]    1
[11,]    1
[12,]    1
```

### 4.3.2 Get the likelihood-based $p$-value matrix

We use the function `like.matrix` to obtain the maximum loglikelihood matrix for each corresponding design matrix.

```
> Y.size = lib.size(data.Y)
> like = like.matrix(data.Y,Y.size,categorical,predictor)
> like[1:5,]
          [,1]       [,2]       [,3]       [,4]
[1,]  -4.47587  -1.308759  -1.376661  -5.219298
[2,] -69.32111 -70.044081 -69.300171 -70.047540
[3,] -31.37401 -31.141502 -31.129230 -31.379074
[4,] -10.81478  -9.206885  -9.164114 -11.182220
[5,] -61.80823 -61.375619 -60.958209 -61.830574


> stat=matrix(0,dim(data.Y)[1],3)
> stat[,1]=-2*(like[,4]-like[,1])
> stat[,2]=-2*(like[,4]-like[,2])
> stat[,3]=-2*(like[,4]-like[,3])
> p.matrix=matrix(0,dim(data.Y)[1],3)

> p.matrix[,1]=1-pchisq(stat[,1], df=length(unique(a1))-1)
> p.matrix[,2]=1-pchisq(stat[,2], df=1)
> p.matrix[,3]=1-pchisq(stat[,3], df=length(unique(a1)))
> p.matrix
          [,1]        [,2]       [,3]
[1,] 0.2227048 0.005164037 0.02143699
[2,] 0.2280698 0.933708097 0.47361119
[3,] 0.9198015 0.490630244 0.77892282
[4,] 0.3913040 0.046852743 0.13290701
[5,] 0.8325812 0.340138321 0.41796213
```

The first column of `p.matrix` is for factor 1; the second column is for `score`; the third column is for both factors.

```
### sort by the "score" main effect p-value and show the top 5 genes,
### with annotation
> p.complete[order(p.complete$p.score)[1:5],]
         p.sex      p.score      p.joint   chr       ref.id     gene
4383 0.1512731 8.474767e-06 3.294133e-05 chr19 NM_001085384   ZNF154
960  0.2152014 5.420721e-04 3.067627e-03 chr15    NM_000814   GABRB3
4807 0.8653363 8.797603e-04 3.411514e-03  chr7 NM_001105533  C7orf58
32   0.9990547 1.420259e-03 3.793506e-03  chrX    NM_000354 SERPINA7
101  0.9990547 1.420259e-03 3.793506e-03  chrX NM_001013628 DCAF12L2
```

## References

[Zhou and Wright, 2011] Yihui Zhou and Fred A. Wright (2011) A very powerful and flexible method for RNA-Seq data analysis *Bioinformatics,* **submitted in Mar, 2011**

[Montgomery, 2010] Montgomery, S.B. et al. (2010)Transcriptome genetics using second generation sequencing in a Caucasian population. *Nature*, **464**(7289), 773-777.