Advanced analysis using baySeq; generic distribution definitions

Thomas J. Hardcastle

April 30, 2024

1 Generic Prior Distributions

baySeq now offers complete user-specification of underlying distributions. This vignette describes using baySeq under this protocol. Familiarity with standard (negative-binomial) baySeq is assumed; please consult the other vignettes for a description of this approach.

Analysis is carried out through specification of a densityFunction class. The primary value in a densityFunction object is the @density slot, a user-defined function that should take variables x, observables and parameters. x corresponds to a row of data in a countData object. observables is a list object of observed values that may influence the model. By default, the @libsizes and @seglens values of the countData object will be internally appended to this list, unless objects with these names are otherwise specified by the user. parameters is a list object of parameters to be empirically estimated from the data with the getPriors function and used to estimate likelihoods with the getLikelihoods function. The @dist function should return a vector of log-likelihood values (or NA for invalid parameter choices) of the same length as the input variable x.

Other required slots of the densityFunction object are initiatingValues, a vector of initiating values to be used in optimisation of the parameters to be used in the @dist slot (and thus defining the length of the parameter object) and equalOverReplicates, a specification of which parameters are fixed for every replicate group and which may vary for different experimental conditions. If only one parameter is variable over experimental conditions, the Nelder-Mead optimisation used may be unstable, and one-dimensional optimisation with user defined functionally specified lower and upper bounds may (optionally) be provided; otherwise, Nelder-Mead will be attempted.

Optionally a function may be provided in @stratifyFunction to stratify the data and improve prior estimation in the tails where the samplesize argument in the getPriors function is less than the row dimension of the countData object. If this is provided, the @stratifyBreaks slot should give the number of strata to be used.

Below a model is constructed based on the normal distribution. The standard deviation is assumed to be constant for a given row of data across all experimental conditions, while the means (normalised by library scaling factor) are allowed to vary across experimental conditions.

If parallelisation is available, it is useful to use it.

```
> if(require("parallel")) cl <- makeCluster(4) else cl <- NULL
```

```
> library(baySeq)
> normDensityFunction <- function(x, observables, parameters) {
```

```
+   if(any(sapply(parameters, function(x) any(x < 0)))) return(rep(NA, length(x)))
+   dnorm(x, mean = parameters[[2]] * observables$libsizes, sd = parameters[[1]], log = TRUE)
+ }
> normDensity <- new("densityFunction", density = normDensityFunction, initiatingValues = c(0.1, 1),
+                  equalOverReplicates = c(TRUE, FALSE),
+                  lower = function(x) 0, upper = function(x) 1 + max(x) * 2,
+                  stratifyFunction = rowMeans, stratifyBreaks = 10)
```

We construct the `countData` object as before.

```
> data(simData)
> CD <- new("countData", data = simData,
+            replicates = c("simA", "simA", "simA", "simA", "simA",
+              "simB", "simB", "simB", "simB", "simB"),
+            groups = list(NDE = c(1,1,1,1,1,1,1,1,1,1),
+                           DE = c(1,1,1,1,1,2,2,2,2,2))
+            )
> libsizes(CD) <- getLibsizes(CD)
> densityFunction(CD) <- normDensity
```

We can then fit priors and calculate posterior likelihoods based on our specified distributional model. The distributional model is specified in the 'getPriors' function and will be automatically used in the ‘getLikelihoods' function

```
> normCD <- getPriors(CD, cl = cl)
> normCD <- getLikelihoods(normCD, cl = cl)

.
```

Similarly, we can construct a generic version of the negative-binomial model.

```
> nbinomDensityFunction <- function(x, observables, parameters) {
+   if(any(sapply(parameters, function(x) any(x < 0)))) return(NA)
+   dnbinom(x, mu = parameters[[1]] * observables$libsizes * observables$seglens, size = 1 / parameters[[2]
+ }
> densityFunction(CD) <- new("densityFunction", density = nbinomDensityFunction, initiatingValues = c(0.1,
+                           equalOverReplicates = c(FALSE, TRUE),
+                           lower = function(x) 0, upper = function(x) 1 + max(x) * 2,
+                           stratifyFunction = rowMeans, stratifyBreaks = 10)
> nbCD <- getPriors(CD, cl = cl)
> nbCD <- getLikelihoods(nbCD, cl = cl)

.
```

We can compare this to the standard analysis of these data.

```
> CD <- getPriors.NB(CD, cl = cl)
> CD <- getLikelihoods(CD, cl = cl)

.
```

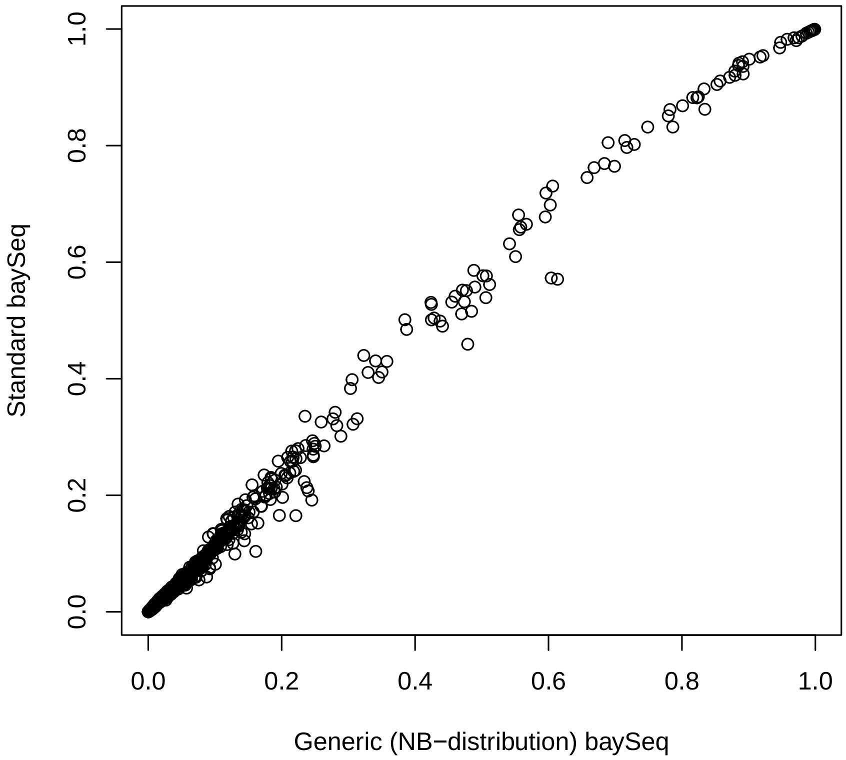

Figure 1: Likelihoods of DE estimated by standard/generic baySeq"

The generic negative-binomial data performs almost identically to standard baySeq. The methods differ in that the standard baySeq uses quasi-maximum-likelihood to estimate the priors, while generic baySeq uses maximum-likelihood (since no generic method exists for quasi-maximum-likelihood on arbitrary distributions).

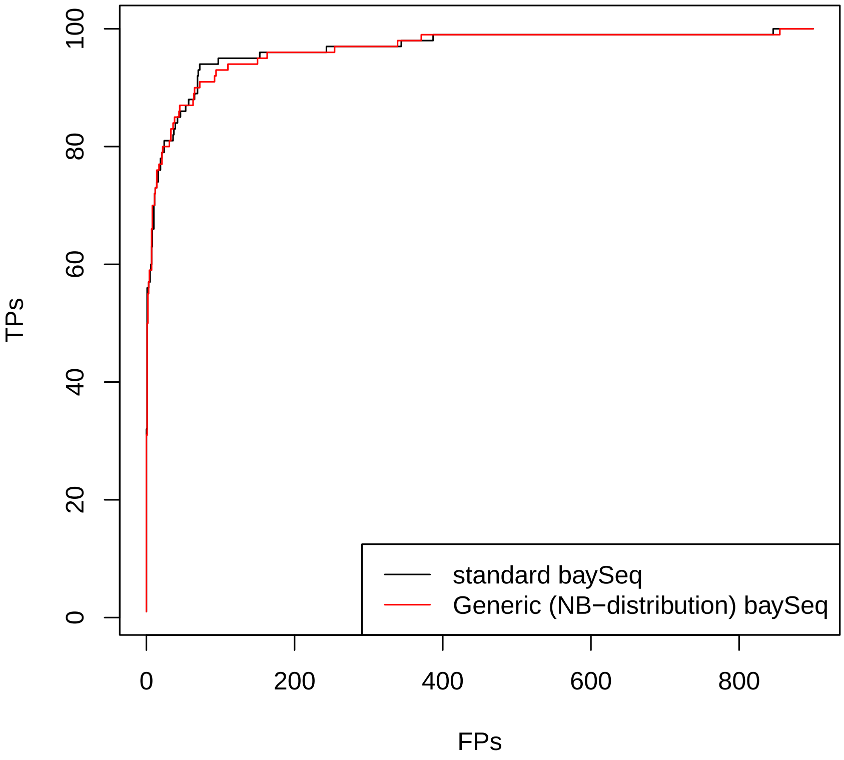

Figure 2: ROC curves estimated by standard/generic baySeq"

2 Paired Data Analysis

We illustrate the possibilities of 'null' data, in which two separate models are applied to data equivalently expressed across all samples. The process for analysing paired data follows approximately the same steps as for analysing unpaired data, however, two different types of differential expression can exist within paired data. Firstly, we can find differential expression between replicate groups, as before. However, we can also find (consistent) differential expression between pairs; this would occur when for a single row of data, the first member of each pair differs from the second member of each pair. baySeq can identify both these types of differential expression simultaneously, and we implement this proceedure below.

We begin by loading a simulated dataset containing counts for four paired datasets.

```
> data(pairData)
```

The first four columns in these data are paired with the second four columns. We construct a count data containing paired data in a similar fashion to the countData object. Note that the data are now three dimensional; for each row and each sample there are two observations.

```
> pairCD <- new("countData", data = array(c(pairData[,1:4], pairData[,5:8]), dim = c(nrow(pairData), 4, 2))
+                  replicates = c(1,1,2,2),
+                  groups = list(NDE = c(1,1,1,1), DE = c(1,1,2,2)),
+               densityFunction = bbDensity)
```

We can find the library sizes for the data with the `getLibsizes` function.

```
> libsizes(pairCD) <- getLibsizes(pairCD)
```

We estimate an empirical distribution on the parameters of a beta-binomial distribution by bootstrapping from the data, taking individual counts and finding the maximum likelihood parameters for a beta-binomial distribution. By taking a sufficiently large sample, an empirical distribution on the parameters is estimated. A sample size of around 10000 iterations is suggested, depending on the data being used), but 1000 is used here to rapidly generate the plots and tables.

```
> pairCD <- getPriors(pairCD, samplesize = 1000, cl = cl)
```

We then acquire posterior likelihoods as before. The use of 'nullData = TRUE' in this context allows us to identify pairs which show no differential expression between replicate groups, but does show deviation from a one-to-one ratio of data between pairs.

```
>   pairCD <- getLikelihoods(pairCD, pET = 'BIC', nullData = TRUE, cl = cl)

.
```

We can ask for the top candidates for differential expression between replicate groups using the `topCounts` function as before.

```
>   topCounts(pairCD, group = 2)

       X1.1  X1.2     X2.1   X2.2     likes  DE      FDR.DE     FWER.DE
NA   159:73 44:24     0:49   0:68 0.9978811 1>2 0.002118861 0.002118861
NA.1  53:12  19:7     0:77    0:6 0.9949696 1>2 0.003574636 0.007138613
NA.2  709:0 895:0  373:191 124:60 0.9934378 1>2 0.004570497 0.013653987
NA.3   25:0  73:0      8:3  36:13 0.9907160 1>2 0.005748878 0.022811244
NA.4   80:0  48:0    36:50   12:3 0.9877493 1>2 0.007049242 0.034782490
NA.5   63:0  21:0    47:80   6:13 0.9850680 1>2 0.008363042 0.049195158
NA.6  268:0  39:0   74:107  98:36 0.9808506 1>2 0.009903951 0.067402506
NA.7 123:63 38:36 1198:179 350:18 0.9661292 2>1 0.012899802 0.098990286
NA.8    8:0  15:0    21:16    2:1 0.9655802 1>2 0.015290908 0.130002824
NA.9  43:19 44:46    106:6  133:5 0.9649375 2>1 0.017268064 0.160507075
```

However, we can also look for consistent differential expression between the pairs.

```
>   topCounts(pairCD, group = 1)

        X1.1  X1.2    X2.1  X2.2     likes     FDR.NDE    FWER.NDE
NA     17:70  1:40   9:117  3:45 0.9924372 0.007562821 0.007562821
NA.1 1027:27 835:8 1155:29 138:0 0.9900925 0.008735169 0.017395409
NA.2    1:38  0:68    0:28  0:26 0.9860246 0.010481900 0.031127665
NA.3     1:4  1:11     0:5  1:14 0.9827337 0.012177991 0.047856471
NA.4     1:2  1:16    2:41   0:2 0.9787194 0.013998511 0.068118647
NA.5    69:1  10:1  119:17  53:5 0.9781813 0.015301877 0.088451095
NA.6    0:12   0:4     0:4  0:13 0.9659482 0.017980439 0.119490985
NA.7    0:30   0:5    0:60  0:24 0.9657384 0.020015587 0.149658651
NA.8     0:4  0:21     0:2  0:12 0.9656937 0.021603445 0.178830724
NA.9     0:3  0:12    0:15   0:4 0.9656224 0.022880859 0.207060539
```

3 Different Model Priors

It is now possible to use different model priors for different subsets of the countData object. If we expect a certain class of genes (for example) to have a different prior likelihood towards differential expression than another such class, we can separate the two sets and estimate (or set) the model priors independently.

Let us suppose that we have reason to believe that the first hundred genes in the 'CD' object are likely to behave differently to the remaining genes. Then

```
> # FAILS Bioc 3.17
> CDv <- getLikelihoods(nbCD, modelPriorSets = list(A = 1:100, B = 101:1000), cl = cl)
```

The model priors used are recorded in the @priorModels slot.

```
> CDv@priorModels
```

We can see the difference in performance by computing the ROC curves as before. Using different model priors can substantially improve performance, although obviously we have cheated here by splitting exactly those data simulated as DE and those as none-DE. It should also be recognised that this approach may bias downstream analyses; e.g. GO enrichment analysis.

Figure 3: ROC curves estimated by standard/generic/variable model priors baySeq"

Several pre-existing distributions are built into baySeq. Here we use a pre-developed zero-inflated negative binomial distribution to analyse zero-inflated data.

```
> data(zimData)
> CD <- new("countData", data = zimData,
+           replicates = c("simA", "simA", "simA", "simA", "simA",
+             "simB", "simB", "simB", "simB", "simB"),
+           groups = list(NDE = c(1,1,1,1,1,1,1,1,1,1),
+                         DE = c(1,1,1,1,1,2,2,2,2,2))
+           )
> libsizes(CD) <- getLibsizes(CD)
> densityFunction(CD) <- nbinomDensity
> CD <- getPriors(CD, cl = cl)
> CD <- getLikelihoods(CD, cl = cl)

.

> CDz <- CD
> densityFunction(CDz) <- ZINBDensity
> CDz <- getPriors(CDz, cl = cl)
> CDz <- getLikelihoods(CDz, cl = cl)
```

```
.
```

Finally, we shut down the cluster (assuming it was started to begin with).

```
> if(!is.null(cl)) stopCluster(cl)
```

Session Info

```
> sessionInfo()

R version 4.4.0 beta (2024-04-15 r86425)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 22.04.4 LTS

Matrix products: default
BLAS:   /home/biocbuild/bbs-3.19-bioc/R/lib/libRblas.so 
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8       
 [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8   
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C              
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C       

time zone: America/New_York
tzcode source: system (glibc)

attached base packages:
[1] parallel  stats     graphics  grDevices utils     datasets  methods   base     

other attached packages:
[1] baySeq_2.38.0

loaded via a namespace (and not attached):
 [1] httr_1.4.7              cli_3.6.2               knitr_1.46             
 [4] rlang_1.1.3             xfun_0.43               UCSC.utils_1.0.0       
 [7] jsonlite_1.8.8          statmod_1.5.0           S4Vectors_0.42.0       
[10] BiocStyle_2.32.0        htmltools_0.5.8.1       stats4_4.4.0           
[13] locfit_1.5-9.9          rmarkdown_2.26          grid_4.4.0             
[16] evaluate_0.23           abind_1.4-5             fastmap_1.1.1          
[19] yaml_2.3.8              IRanges_2.38.0          GenomeInfoDb_1.40.0    
[22] BiocManager_1.30.22     compiler_4.4.0          Rcpp_1.0.12            
[25] limma_3.60.0            edgeR_4.2.0             XVector_0.44.0         
[28] lattice_0.22-6          digest_0.6.35           R6_2.5.1               
[31] GenomeInfoDbData_1.2.12 GenomicRanges_1.56.0    tools_4.4.0            
[34] zlibbioc_1.50.0         BiocGenerics_0.50.0    
```