SIBERG User Manual

Pan Tong and Kevin R Coombes

July 23, 2024

Contents

1 Introduction

SIBERG (**S**ystematic **I**dentification of **B**imodally **E**xp**R**essed **G**enes using RNAseq data) is an R package that effectively identifies bimodally expressed genes from RNAseq data based on Bimodality Index. SIBER models the RNAseq data in the finite mixture modeling framework and incorporates mechanisms for dealing with RNAseq normalization. Three types of mixture models are implemented, namely, the mixture of log normal, negative binomial or generalized poisson distribution. For completeness, we also add the normal mixture model that has been used to identify bimodal genes from microarray data.

SIBER proceeds in two steps. The first step fits a two-component mixture model. The second step calculates the Bimodality Index corresponding to the assumed mixture distribution. Four types of mixture models are implemented: log normal (LN), Negative Binomial (NB), Generalized Poisson (GP) and normal mixture (NL).

Besides identifying bimodally expressed genes, SIBER provides functionalities to fit 2-component mixture distribution from LN, NB and GP models. A degenerate case where one component becomes a point mass at zero (called 0-inflation) is also incorporated. The 0-inflated model is designed specificaly to deal with the observed zero count in real RNAseq data.

2 Using SIBER

2.1 A Quick Example

Of course, we need to load the SIBER package.

```
> library(SIBERG)
```

We simulate RNAseq count data from 1-component Negative Binomial distribution as below:

```
> set.seed(1000)
> N <- 100 # sample size
> G <- 200 # number of simulated genes
> # RNAseq count data simulated from NB model with mean 1000, dispersion=0.2
> Dat <-  matrix(rnbinom(G*N, mu=1000, size=1/0.2), nrow=G)
```

We use the first gene for our illustration. We first fit the LN mixture model and calculate BI:

```
> SIBER(y=Dat[1, ], model='LN')

      mu1       mu2    sigma1    sigma2       pi1     delta        BI
6.2581878 6.9015498 0.3641801 0.3641801 0.1032280 1.7666038 0.5375005
```

To apply the NB model:

```
> SIBER(y=Dat[1, ], model='NB')

         mu1          mu2       sigma1       sigma2          pi1        delta
 881.7878094 1292.5535708  322.5813919  472.2132984    0.7129706    0.9452263
          BI
   0.4275971
```

To apply the GP model:

```
> SIBER(y=Dat[1, ], model='GP')

         mu1          mu2       sigma1       sigma2          pi1        delta
5.789030e+02 1.022999e+03 3.059162e+02 4.066654e+02 5.248746e-02 1.423320e+00
          BI
3.174115e-01
```

For the NL model, we first transform the data such that it follows normal mixture distribution.

```
> SIBER(y=log(Dat[1, ]+1), model='NL')

      mu1       mu2    sigma1    sigma2       pi1     delta        BI
6.2297063 6.8917722 0.3672838 0.3672838 0.1016360 1.8026005 0.5446901
```

Since the data is simulated from 1-component model, all of the calculated BIs are small indicating lack of bimodality.

2.2 Dealing With RNAseq Normalization

Previously, only the raw RNAseq count data is passed to `SIBER`. It is easy to incorporate RNAseq normalization in the mixture modeling. Currently, the RPKM [Mortazavi et al., 2008], TMM [Robinson et al., 2010b] and RLE [Anders and Huber, 2010] methods have been widely used to normalize RNAseq data. Once the normalization constant is estimated, i.e. using the `edgeR` package [Robinson et al., 2010a], we can easily calculate the BI after adjusting for the normalization.

In the following, we use `edgeR` package to calculate the normalization factor using TMM approach.

```
> if (require(edgeR)) {
   TMM <- calcNormFactors(Dat, method='TMM')
 } else {
   # manually set factors from previous computations
   TMM <- c(1.0390711, 0.9813734, 1.0091593, 0.9641022, 1.0137000,
            1.0188657, 0.9648757, 0.9956814, 0.9689530, 0.9774278,
            1.0059115, 1.0076910, 0.9923854, 1.0121838, 1.0249094,
            1.0403172, 0.9887074, 1.0003546, 0.9998479, 0.9844905,
            1.0040203, 0.9692244, 0.9987567, 1.0063895, 0.9954510,
            1.0204917, 0.9717720, 1.0317981, 0.9826344, 0.9817171,
            0.9949059, 0.9745569, 0.9652138, 1.0075196, 0.9879748,
            0.9929244, 0.9895606, 1.0144117, 1.0612923, 0.9626716,
            1.0049376, 1.0192416, 0.9826612, 1.0234523, 0.9921186,
            1.0029780, 1.0199930, 1.0054256, 1.0152748, 0.9655475,
            0.9919175, 1.0231102, 0.9750882, 0.9958528, 1.0268000,
            0.9651300, 1.0158949, 0.9803130, 1.0385707, 0.9870510,
            1.0211765, 1.0326759, 1.0234579, 0.9524254, 0.9742719,
            0.9887936, 1.0476640, 0.9787385, 0.9992178, 1.0046021,
            0.9929379, 0.9595237, 1.0690364, 0.9910940, 1.0158325,
            0.9799790, 1.0316363, 1.0341890, 1.0036944, 0.9728850,
            1.0080238, 1.0190104, 0.9735436, 0.9744903, 0.9974915,
            0.9804733, 1.0243671, 0.9881085, 0.9923432, 0.9638553,
            1.0178705, 1.0476191, 1.0260725, 1.0474791, 1.0449745,
            0.9987096, 1.0028339, 0.9971751, 0.9487246, 0.9696386)
 }
```

We now incorporate the TMM normalization into SIBER. We use the LN model below. The calculation with other models is similar. Note that our definition of the normalization factor differs from `edgeR` package. In our notation, $\mathrm{E}[C_s] = d_s \mu_{c(s)}$ where C_s is the observed raw count for sample s, d_s is the normalization factor applied to sample s, c(s)=$\{1, 2\}$ denotes which of the two components sample s comes from and μ_1, μ_2 are mean parameters for the two components. Therefore, our definition of d_s maps the true expression level to the observed counts. In contrast, the normalization constant estimated by `edgeR` maps the observed counts to the estimated true expression. As a result, we need to pass the reciprocal of the normalization vector estimated by `edgeR` to SIBER.

```
> SIBER(y=Dat[1, ], d=1/TMM, model='LN')

      mu1       mu2    sigma1    sigma2       pi1     delta        BI
6.2533386 6.9024425 0.3666528 0.3666528 0.1036901 1.7703504 0.5397056
```

2.3 Parallelizing `SIBER`

When there are many genes to be fitted, we can easily parallel `SIBER` to speed up the computation. There are several ways for parallelization. Here we choose the `foreach` package for the backend. The workers are requested and registered by the `doSNOW` package.

```
library(doParallel)
cl <- makeCluster(3, type = "SOCK")
registerDoParallel(cl)
```

Note that the above command also works on Linux servers. However, it requests master nodes when run within R. For good practice, we can use qsub such that the computation is done in the compute nodes.

Below we illustrate how to use SIBER with parallel computation.

```
    func <- function(i) {
 SIBER(y=Dat[i, ], model='LN')
    }
    BIinfo_LN <- foreach(i=1:nrow(Dat),
                         .combine='rbind',
                         .packages='SIBER') %dopar% {
 func(i)
    }
```

3 Fitting Two-component Mixture Models

SIBER package provides functions to fit three types of mixture models besides detecting bimodally expressed genes. These include: (1) 2-component mixture with equal dispersion or variance (E model); (2) 2-component mixture with unequal dispersion or variance (V model); (3) 0-inflated model. All three types of distributions are implemented.

The rule to fit 0-inflated model is that the observed percentage of count exceeds the user specified threshold. This rule overrides the model argument (E or V) when observed percentae of zero count exceeds the threshold.

First, we illustrate how to fit the E and V models. We use the simulated data from LN model. The gene we use is not 0-inflated. By default, the minimum observed percentage of zero is not achieved (zeroPercentThr=0.2). Hence, the 0-inflated model is disabled. In this case, the model specification will be effective.

```
> data(simDat)
> ind <- 1
> # true parameter generating the simulated data
> parList$LN[ind, ]

   mu1    mu2 sigma1 sigma2    pi1
   5.0    9.0    1.0    1.0    0.1

> # fit by E model
> fitLN(y=dataList$LN[ind, ], base=exp(1), eps=1, model='E')

          mu1           mu2        sigma1        sigma2           pi1
    4.6990876     9.1145635     0.9417622     0.9417622     0.1029831
       logLik           BIC
-2067.7614236  4156.7161167

> # fit by V model.
> fitLN(y=dataList$LN[ind, ], base=exp(1), eps=1, model='V')

          mu1           mu2        sigma1        sigma2           pi1
    4.6373527     9.1072744     0.7071008     0.9650604     0.1000980
       logLik           BIC
-2066.6375398  4159.7666664
```

```
>
```

Now we choose a gene that has zero inflation and illustrate how to fit a 0-inflated model:

```
> ind <- 5 # 0-inflated gene
> # true parameter generating the simulated data
> parList$LN[ind, ]

   mu1    mu2 sigma1 sigma2    pi1
   0.0    4.0    1.0    1.0    0.3

> # fit by E model. 0-inflated model is disabled by setting zeroPercentThr=1.
> # the result is biased.
> fitLN(y=dataList$LN[ind, ], base=exp(1), eps=1, model='E', zeroPercentThr=1)

         mu1           mu2        sigma1        sigma2           pi1
   0.05104833    4.01559275    0.78377975    0.78377975    0.30833327
       logLik           BIC
-914.36505926 1849.92338799

> # fit by 0-inflated model. 0-inflated model overrides the E model since percentage
> # of observed zero counts exceeds the threshold.
> fitLN(y=dataList$LN[ind, ], base=exp(1), eps=1, model='E', zeroPercentThr=0.2)

         mu1          mu2       sigma1       sigma2          pi1       logLik
   0.0000000    3.9612703    0.0000000    0.9722536    0.3000000 -870.9626887
         BIC
1757.8203295

>
```

Here we see that when there is severe 0-inflation, fitting a E (or V) model gives biased estimate. Instead, our 0-inflated model works pretty well.

The usage of fitNB(), fitGP() is quite similar and is omitted in this manual.

4 Session Info

After all the computations, we close the connection to the workers.

```
    stopCluster(cl)
```

```
> getwd()

[1] "/tmp/Rtmp8UkWkh/Rbuild5d05551f9a4/SIBERG/vignettes"

> sessionInfo()

R version 4.4.1 (2024-06-14)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 24.04 LTS
```

```
Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.26.so;  LAPACK version 3.12.0

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

time zone: Etc/UTC
tzcode source: system (glibc)

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

other attached packages:
[1] edgeR_4.3.5  limma_3.61.4 SIBERG_2.0.3

loaded via a namespace (and not attached):
 [1] compiler_4.4.1   mclust_6.1.1     tools_4.4.1      Rcpp_1.0.13
 [5] maketools_1.3.0  buildtools_1.0.0 grid_4.4.1       locfit_1.5-9.10
 [9] knitr_1.48       xfun_0.46        sys_3.4.2        lattice_0.22-6
[13] statmod_1.5.0
```

References

Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data. *Genome biol*, 11(10):R106, 2010.

Ali Mortazavi, Brian A Williams, Kenneth McCue, Lorian Schaeffer, and Barbara Wold. Mapping and quantifying mammalian transcriptomes by rna-seq. *Nature methods*, 5(7):621–628, 2008.

Mark D Robinson, Davis J McCarthy, and Gordon K Smyth. edger: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–140, 2010a.

Mark D Robinson, Alicia Oshlack, et al. A scaling normalization method for differential expression analysis of rna-seq data. *Genome Biol*, 11(3):R25, 2010b.