User guide for *BPSC* package version 0.99.1

Trung Nghia Vu

February 24, 2017

1 Introduction

Single-cell RNA-sequencing technology allows detection of gene expression in single-cell level. One typical feature of the data is a bimodality in the cellular distribution even for highly expressed genes, primarily caused by a proportion of non-expressing cells. The standard and the over-dispersed gamma-Poisson models that are commonly assumed in bulk-cell RNA-sequencing are not able to capture this property. In the recent study [1], we introduce a beta-Poisson mixture model to capture the bimodality of the single-cell RNA-seq gene expression distribution. We also propose a method that combines the beta-Poisson model with a generalized linear model to do differential expression analysis for single-cell RNAseq data.

In this document, we present practical uses of the beta-Poisson model for differential expression analysis and model fitting.

2 Differential expression analysis

BPSC integrates the beta-Poisson model and generalized linear model (BPglm) for differential expression analysis. For real datasets, the BPglm requires input from a dataset after normalized, for example by fragments per kilo-base per million (FPKM) or counts per million (cpm). In this document, we use beta-Poisson models to generate a small simulated dataset. Then, the function *BPglm*() is applied to discover differentially expressed genes.

2.1 Simulation dataset

In order to do differential expression analysis, we randomly generate a dataset consisting of two groups of 100 treated cells and 100 control cells.

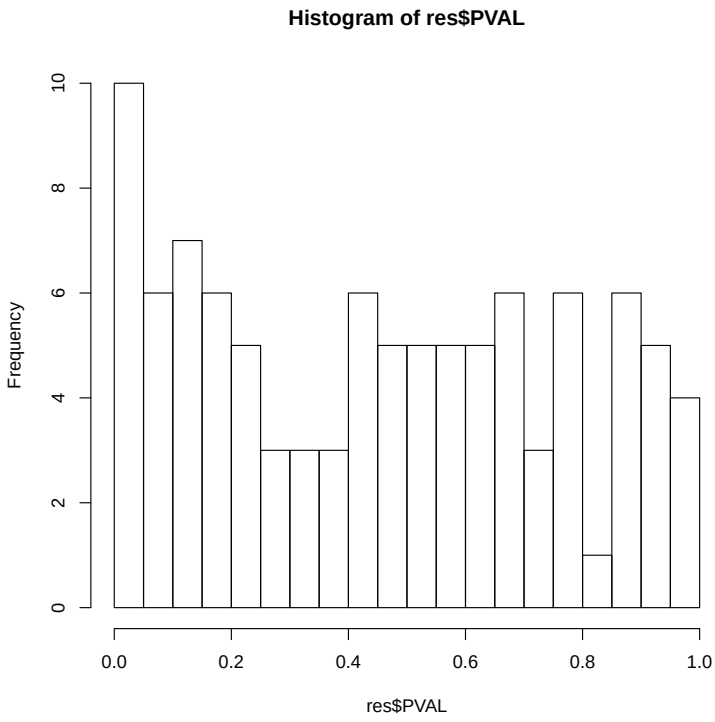
```
library(BPSC)

## Loading required package:  statmod
## Loading required package:  doParallel
## Loading required package:  foreach
## Loading required package:  iterators
## Loading required package:  parallel

set.seed(2015)
###Generate a random data matrix from a beta-poisson model
#Set the number of genes
N=100
#Generate randomly the parameters of BP models
alp=sample(100,N,replace=TRUE)*0.1;
bet=sample(100,N,replace=TRUE)*0.1;
lam1=sample(100,N,replace=TRUE)*10;
lam2=sample(100,N,replace=TRUE)*0.01
```

```
#Generate a control group
n1=100
control.mat=NULL
for (i in 1:N) control.mat=rbind(control.mat,rBP(n1,alp=alp[i],
        bet=bet[i],lam1=lam1[i],lam2=lam2[i]))

#To create biological variation, we randomly set 10% as differentially expressed genes
#by simply replacing the parameter lam1 in treated group by a fold-change fc
DE.ids= sample(N,N*0.1)
fc=2.0
lam1[DE.ids]=lam1[DE.ids] * fc

#Generate a treated group
n2=100
treated.mat=NULL
for (i in 1:N)treated.mat=rbind(treated.mat,rBP(n2,alp=alp[i],
        bet=bet[i],lam1=lam1[i],lam2=lam2[i]))

#Create a data set by merging the control group and the treated group
bp.mat=cbind(control.mat,treated.mat)
rownames(bp.mat)=c(1:nrow(bp.mat));
colnames(bp.mat)=c(1:ncol(bp.mat))
group=c(rep(1,ncol(control.mat)),rep(2,ncol(treated.mat)))
```

2.2 Differential expression analysis using BPgml

Now we apply the BPglm model on the simulated dataset for differential expression analysis.

```
# First, choose IDs of all cells of the control group for estimating
# parameters of BP models
controlIds = which(group == 1)

# Create a design matrix including the group labels. All batch effects can
# be also added here if they are available
design = model.matrix(~group)
# Select the column in the design matrix corresponding to the coefficient
# (the group label) for the GLM model testing
coef = 2

# Run BPglm for differential expression analysis
res = BPglm(data = bp.mat, controlIds = controlIds, design = design, coef = coef,
    estIntPar = FALSE, useParallel = FALSE)
# In this function, user can also set estIntPar=TRUE to have better
# estimated beta-Poisson models for the generalized linear model. However, a
# longer computational time is required.

# Plot the p-value distribution
hist(res$PVAL, breaks = 20)
```




```
# Summarize the resutls
ss = summary(res)

## Top five fdr:
##     t value     Pr(>|t|)          fdr
## 64 26.94352 3.761350e-68 3.761350e-66
## 62 26.64435 2.132749e-67 1.066375e-65
## 70 21.32475 3.474072e-53 1.158024e-51
## 24 17.94747 2.148172e-43 5.370430e-42
## 29 16.94198 2.209021e-40 4.418042e-39

# Compare the discovered DE genes and the true DE genes predefined
# beforeward
fdr = p.adjust(res$PVAL, method = "BH")
bpglm.DE.ids = which(fdr <= 0.05)
# Print the indices of the true DE genes:
cat(sort(DE.ids))

## 13 18 24 29 36 62 64 65 68 70

# Print the indices of the DE genes discovered by BPglm:
cat(sort(bpglm.DE.ids))

## 13 18 24 29 36 62 64 65 68 70
```

Parallel computing

To speedup the performance, we can run the functions in parallel by setting *useParallel*=TRUE in these functions. However, before running, we need to register the number of cores for the parallel. The performance is linear to the number of cores.

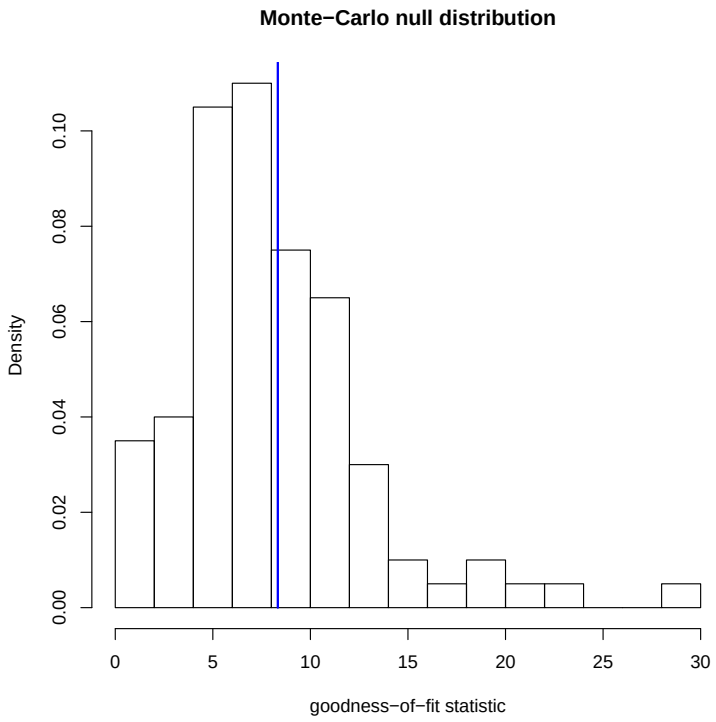
```
library(doParallel)
registerDoParallel(cores=16)
```

3 Model fitting

In this section, we present how to get the best fitted beta-Poisson model for gene expression of single-cell RNA-seq. The parameters of the beta-Poisson model are possibly linked to biological interpretation of burst size and burst frequency of the cell-level expression [1]. First, we show an example of using beta-Poisson model to fit gene expresion of a single gene, then further to a gene expression dataset consisting of multiple genes.

3.1 Fitting for a single gene

```
library(BPSC)
set.seed(2015)
#Create a simulated gene expression by randomly generating 100 data points
#from a beta-poisson model
alp=0.6;bet=1.5;lam1=20;lam2=0.05
par0=c(alp,bet,lam1,lam2)
bp.vec=rBP(100,par0)

#Estimate parameters of the four-parameter beta-Poisson model from the data points
res=estimateBP(bp.vec,para.num=4)
#Print the goodness-of-fit of the model and the optimal parameters
res$X2

## [1] 8.330293

res$par

## [1]  0.5308205  1.0579389 33.4361406  0.0254297

#Generate Monte-Carlo null distribution of the model.
#Due to time limit, the number of simulations (sim.num) here is set by 100.
MCnull.res=getBPMCnull(res$par,n=100,tbreak=res$tbreak,sim.num=100)
#Compute Monte-Carlo p-value
MCpval=sum(MCnull.res$X2 >= res$X2)/length(MCnull.res$X2)
MCpval

## [1] 0.4

#Plot the Monte-Carlo null distribution and the goodness-of-fit from the model (blue line).
hist(MCnull.res$X2,xlab="goodness-of-fit statistic",
main="Monte-Carlo null distribution",breaks=20, prob=TRUE)
lines(c(res$X2,res$X2),c(0,par("usr")[4]),
      col="blue",lwd=2.0)
```



Thus, the estimated model has a well MC p-value, indicating that the model has a good fit.

3.2 Fitting for a gene expression dataset

We also can use *estimateBPMatrix*() function to do fitting for a data matrix of gene expression of multiple genes. We start by creating a simulated dataset containing 10 genes.

```
set.seed(2015)
#create random data matrix from a beta-poisson model
N=10
alp=sample(100,N,replace=TRUE)*0.1;
bet=sample(100,N,replace=TRUE)*0.1;
lam1=sample(100,N,replace=TRUE)*10;
lam2=sample(100,N,replace=TRUE)*0.01;
n=100
bp.mat=NULL
for (i in 1:N)
  bp.mat=rbind(bp.mat,rBP(n,alp=alp[i],bet=bet[i],lam1=lam1[i],lam2=lam2[i]))
```

Then, we do fitting for this dataset.

```
#Estimate parameters from the data set
mat.res=estimateBPMatrix(bp.mat,para.num=4,fout=NULL,estIntPar=FALSE,useParallel=FALSE)
```

Note that one can use *estIntPar*=TRUE to estimate initial parameters for the beta-Poisson models from non-zero expressed values to select better results. However, that requires longer overall computational time.

We also can use *getBPMCnullmatrix*() function to massivelly generate the Monte-Carlo null distributions for all the BP models of the dataset, then calculate Monte-Carlo p-values.

```
MCnullmatrix.res=getBPMCnullmatrix(bp.model.list=mat.res$bp.model.list,fout=NULL,
                                   sim.num=100,useParallel=FALSE)
#Get Monte-Carlo p-values
```

```
MC.pval=getMCpval(bp.model.list=mat.res$bp.model.list,
                  MCdis.list=MCnullmatrix.res$MCdis.list)
MC.pval

##  [1] 0.79 0.27 0.08 0.08 0.05 0.33 0.10 0.14 0.28 0.76
```

4 References

1. Vu, Trung Nghia, Quin F. Wills, Krishna R. Kalari, Nifang Niu, Liewei Wang, Mattias Rantalainen, and Yudi Pawitan. "Beta-Poisson Model for Single-Cell RNA-Seq Data Analyses." Bioinformatics, April 19, 2016, btw202. doi:10.1093/bioinformatics/btw202.