BUScorrect: Batch Effects Correction with Unknown Subtypes User's Guide

Xiangyu Luo **and Yingying Wei***

The Chinese University of Hong Kong

*xyluo1991@gmail.com

October 30, 2019

Contents

BUScorrect: Batch Effects Correction with Unknown Subtypes
User's Guide

1 Introduction

High-throughput experimental data are accumulating exponentially in public databases. However, mining valid scientific discoveries from these abundant resources is hampered by technical artifacts and inherent biological heterogeneity. The former are usually termed "batch effects," and the latter is often modelled by subtypes.

Researchers have long been aware that samples generated on different days are not directly comparable. Samples processed at the same time are usually referred to as coming from the same "batch." Even when the same biological conditions are measured, data from different batches can present very different patterns. The variation among different batches may be due to changes in laboratory conditions, preparation time, reagent lots, and experimenters [1]. The effects caused by these systematic factors are called "batch effects."

Various "batch effects" correction methods have been proposed when the subtype information for each sample is known [2, 3]. Here we adopt a "broad" definition for "subtype." "Subtype" is defined as a set of samples that share the same underlying genomic profile, in other words biological variability, when measured with no technical artifacts. For instance, groupings such as "case" and "control" can be viewed as two subtypes. However, subtype information is usually unknown, and it is often the main interest of the study to learn the subtype for each collected sample, especially in personalized medicine.

Here, the R package *BUScorrect* fits a Bayesian hierarchical model—the Batch-effects-correction-with-Unknown-Subtypes model (BUS)—to correct batch effects in the presence of unknown subtypes [4]. BUS is capable of (a) correcting batch effects explicitly, (b) grouping samples that share similar characteristics into subtypes, (c) identifying features that distinguish subtypes, and (d) enjoying a linear-order computation complexity. After correcting the batch effects with BUS, the corrected value can be used for other analysis as if all samples are measured in a single batch. BUS can integrate batches measured from different platforms and allow subtypes to be measured in some but not all of the batches as long as the experimental design fulfils the conditions listed in [4].

This guide provides step-by-step instructions for applying the BUS model to correct batch effects and identify the unknown subtype information for each sample.

2 Data Preparation

To fit the BUS model, we first look at the input data format. Two types of data formats are allowed, the R list and the *SummarizedExperiment* object [5]. Specifically, assuming there are 3 batches, the R list consists of 3 gene expression matrices with genes in rows and samples in columns. Alternatively, the user may have a *SummarizedExperiment* object at hand. The *SummarizedExperiment* object is a matrix, where rows represent genes, columns are samples, and the column data record the batch information for each sample. In the following, we provide concrete examples for the two allowable input formats.

```
library(BUScorrect)
data("BUSexample_data", package="BUScorrect")

#BUSexample_data is a list
class(BUSexample_data)

## [1] "list"

#The list's length is three, thus we have three batches
length(BUSexample_data)

## [1] 3

#Each element of the list is a matrix
class(BUSexample_data[[1]])

## [1] "matrix"

#In the matrix, a row is a gene, and a column corresponds to a sample
dim(BUSexample_data[[1]])

## [1] 2000   70

dim(BUSexample_data[[2]])

## [1] 2000   80

dim(BUSexample_data[[3]])

## [1] 2000   70

#Look at the expression data
head(BUSexample_data[[1]][ ,1:4])
```

```
##          [,1]     [,2]     [,3]     [,4]
## [1,] 2.263650 2.250625 2.128499 1.614491
## [2,] 1.912706 2.568451 1.791857 2.378035
## [3,] 1.887739 2.438090 2.139468 1.697558
## [4,] 2.027666 1.818928 2.102251 1.764076
## [5,] 2.712226 1.654110 2.213404 1.482805
## [6,] 2.263879 2.070646 2.058670 1.985701
```

The example data `BUSexample_data` consist of 3 batches. In total, 2000 genes are measured. The sample sizes of each batch are 70, 80, and 70, respectively. Because it is a simulation data set, we actually know that all of the samples come from 3 subtypes. In addition, we can prepare a *SummarizedExperiment* object named `BUSdata_SumExp` using the `BUSexample_data` as follows.

```
#require the SummarizedExperiment from Bioconductor
require(SummarizedExperiment)

## Loading required package:  SummarizedExperiment

## Loading required package:  GenomicRanges

## Loading required package:  stats4

## Loading required package:  BiocGenerics

## Loading required package:  parallel

##
## Attaching package:  'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##    clusterExport, clusterMap, parApply, parCapply, parLapply,
##    parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':
##
##    IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##    Filter, Find, Map, Position, Reduce, anyDuplicated, append,
##    as.data.frame, basename, cbind, colnames, dirname, do.call,
##    duplicated, eval, evalq, get, grep, grepl, intersect,
##    is.unsorted, lapply, mapply, match, mget, order, paste, pmax,
##    pmax.int, pmin, pmin.int, rank, rbind, rownames, sapply,
##    setdiff, sort, table, tapply, union, unique, unsplit, which,
##    which.max, which.min

## Loading required package:  S4Vectors

##
## Attaching package:  'S4Vectors'

## The following object is masked from 'package:base':
##
##    expand.grid

## Loading required package:  IRanges

##
## Attaching package:  'IRanges'

## The following object is masked from 'package:grDevices':
##
##    windows

## Loading required package:  GenomeInfoDb

## Loading required package:  Biobase

## Welcome to Bioconductor
##
##    Vignettes contain introductory material; view with
##    'browseVignettes()'.  To cite Bioconductor, see
##    'citation("Biobase")', and for packages 'citation("pkgname")'.

## Loading required package:  DelayedArray

## Loading required package:  matrixStats

##
## Attaching package:  'matrixStats'
```

```
## The following objects are masked from 'package:Biobase':
##
##    anyMissing, rowMedians

## Loading required package:  BiocParallel

##
## Attaching package:  'DelayedArray'

## The following objects are masked from 'package:matrixStats':
##
##    colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges

## The following objects are masked from 'package:base':
##
##    aperm, apply, rowsum

#batch number
B <- length(BUSexample_data)

#sample size vector
n_vec <- sapply(1:B, function(b){
                                                ncol(BUSexample_data[[b]])})

#gene expression matrix
GE_matr <- NULL
for(b in 1:B){
        GE_matr <- cbind(GE_matr, BUSexample_data[[b]])
}
rownames(GE_matr) <- NULL
colnames(GE_matr) <- NULL

#batch information
Batch <- NULL
for(b in 1:B){
        Batch <- c(Batch, rep(b, n_vec[b]))
}

#column data frame
colData <- DataFrame(Batch = Batch)
```

```
#construct a SummarizedExperiment object
BUSdata_SumExp <- SummarizedExperiment(assays=list(GE_matr=GE_matr), colData=colData)

head(assays(BUSdata_SumExp)$GE_matr[ ,1:4])

##          [,1]     [,2]     [,3]     [,4]
## [1,] 2.263650 2.250625 2.128499 1.614491
## [2,] 1.912706 2.568451 1.791857 2.378035
## [3,] 1.887739 2.438090 2.139468 1.697558
## [4,] 2.027666 1.818928 2.102251 1.764076
## [5,] 2.712226 1.654110 2.213404 1.482805
## [6,] 2.263879 2.070646 2.058670 1.985701

head(colData(BUSdata_SumExp)$Batch)

## [1] 1 1 1 1 1 1
```

In a nutshell, the user can use either the R list or the *SummarizedExperiment* object as input. Regarding the R list, the list length is equal to the batch number, and each list component is a gene expression matrix from a batch. With respect to the *SummarizedExperiment* object, it is a matrix where rows are genes and columns represent samples, and the user has to specify the sample-specific batch indicators through the `Batch` vector in the `colData` column data frame.

3 Model Fitting

Once we have prepared the input data and specified the subtype number, we are able to fit the BUS model, which requires the function `BUSgibbs`. The first argument, `Data`, of `BUSgibbs` should be either an R list or a SummarizedExperiment object. If `Data` is an R list, each element of `Data` is a data matrix for a specific batch, where each row corresponds to a gene or a genomic feature and each column corresponds to a sample. If `Data` is a SummarizedExperiment object, `assays(Data)` must contain a gene expression matrix named "GE_matr," where one row represents a gene and one column corresponds to a sample. `colData(Data)` must include a vector named "Batch", which indicates the batch information for each sample. The second argument, `n.subtypes`, is the number of subtypes among samples, which needs to be specified by the user in advance. As discussed later, if `n.subtypes` is unknown, we can vary the subtype number and use BIC to select the optimal number. The third argument,

`n.iterations`, is the total number of iterations to run by the Gibbs sampler for posterior inference of the BUS model. The first `n.iterations`/2 iterations are treated as burn-in, and posterior samples from the second `n.iterations`/2 iterations are kept for statistical inference. The fourth argument, `showIteration`, lets the user decide whether `BUSgibbs` should display the number of iterations that have been run. To reproduce the results, the users are highly recommended to set `set.seed` before running `BUSgibbs`.

```
#For R list input format
set.seed(123)
BUSfits <- BUSgibbs(Data = BUSexample_data, n.subtypes = 3, n.iterations = 300,
                                                    showIteration = FALSE)

##  running the Gibbs sampler ...

##  The Gibbs sampler takes:  0.343 mins

##  calculating posterior means and posterior modes...

##  calculating BIC...

#For SummarizedExperiment object input format
#set.seed(123)
#BUSfits_SumExp <- BUSgibbs(Data = BUSdata_SumExp, n.subtypes = 3, n.iterations = 300,
# showIteration = FALSE)
```

The `summary` command provides an overview of the output object `BUSfits` from `BUSgibbs`. `BUSfits` collects all the posterior samples and posterior estimates for the intrinsic gene indicators, the subtype class for each sample, the subtype proportions for each batch, the baseline expression levels, the subtype effects, the location batch effects, and the scale batch effects.

```
summary(BUSfits)

## B =  3  batches
## G =  2000  genes
## K =  3  subtypes
## n.records =  150  iterations are recorded.
##
## BUSfits is an R list that contains the following main elements:
##
##    BUSfits$Subtypes : estimated subtype indicators, an R list with length B.
```

```
##    BUSfits$pi : estimated subtype proportions across batches, a B by K matrix.
##    BUSfits$alpha : estimated baseline expression levels, a vector with length G.
##    BUSfits$gamma : estimated location batch effects a G by B matrix.
##    BUSfits$mu : estimated subtype effects, a G by K matrix.
##    BUSfits$sigma_sq : estimated variances across batches, a G by B matrix.
##    BUSfits$BIC : estimated BIC, a scalar.
##    BUSfits$L_PosterSamp : the posterior samples of the intrinsic gene indicators,
##                           a G by K-1 by n.records array.
##    For more output values, please use "?BUSgibbs"
```

4 Estimated Subtypes and Batch Effects

Our main interests lie in the estimation of the subtype class for each sample and the batch effects. We can call the `Subtypes` function to extract the subtype information from `BUSfits`.

```
est_subtypes <- Subtypes(BUSfits)

## Batch 1 samples' subtype indicators:  222...  ...

## Batch 2 samples' subtype indicators:  222...  ...

## Batch 3 samples' subtype indicators:  222...  ...

## The output format is a list with length equal to the batch number.

## Each element of the list is a subtype indicator vector in that batch.
```

There is a message from the function `Subtypes` to remind the user of the format of `est_subtypes`. `est_subtypes` is a list of length 3, corresponding to the three batches in the study. `est_subtypes[[1]]` shows the subtype for each of the 70 samples on batch 1.

Similarly, you can call `location_batch_effects` and `scale_batch_effects` functions to get the estimated location and scale batch effects.

```
est_location_batch_effects <- location_batch_effects(BUSfits)

## The output format is a matrix.
```

```
## Each row represents a gene, and each column corresponds to a batch.

head(est_location_batch_effects)

##      [,1]     [,2]      [,3]
## [1,]    0 3.027846 0.9788055
## [2,]    0 2.956818 1.0375224
## [3,]    0 2.974575 1.0098305
## [4,]    0 3.008553 1.0328079
## [5,]    0 3.019663 0.9759860
## [6,]    0 2.882450 0.8925550

est_scale_batch_effects <- scale_batch_effects(BUSfits)

## The output format is a matrix.

## Each row represents a gene, and each column corresponds to a batch.

head(est_scale_batch_effects)

##      [,1]     [,2]      [,3]
## [1,]    1 1.420293 1.1468249
## [2,]    1 1.179968 1.2829231
## [3,]    1 1.066596 0.9038884
## [4,]    1 1.456537 1.1398424
## [5,]    1 1.343309 1.2217145
## [6,]    1 1.187621 1.0607870
```

The first batch is taken as the reference batch, therefore its location batch effects are zeros for all the genes and its scale batch effects are all ones.

The subtype effects can be obtained by the `subtype_effects` function. Notice that the first subtype is taken as the baseline subtype.

```
est_subtype_effects <- subtype_effects(BUSfits)

## The output format is a matrix.

## Each row represents a gene, and each column corresponds to a subtype.
```

5 Intrinsic Gene Identification

The intrinsic genes are the genes that differentiate subtypes [6]. We use the following functions to identify the intrinsic genes by controlling the false discovery rate.

```
#select posterior probability threshold to identify the intrinsic gene indicators
thr0 <- postprob_DE_thr_fun(BUSfits, fdr_threshold=0.01)

## Posterior probability threshold = 0.42

## The output is a scalar.

est_L <- estimate_IG_indicators(BUSfits, postprob_DE_threshold=thr0)

## The output format is a matrix.

## Each row represents a gene, and each column corresponds to a subtype from 2
to K

#obtain the intrinsic gene indicators
intrinsic_gene_indices <- IG_index(est_L)

## 199intrinsic genes are found.

## The output format is a vector showing the intrinsic gene indices.
```

The function `postprob_DE_thr_fun` calculates the best posterior probability threshold that results in a false discovery rate less than `fdr_threshold`. `postprob_DE_thr_fun` also gives the user a message about the selected posterior probability threshold. `estimate_IG_indicators` then obtain the selected threshold to estimate intrinsic gene indicators. Finally, one can obtain the intrinsic gene indices by calling the function `IG_index`.

The `postprob_DE` function calculates the posterior probability of being differentially expressed for genes in subtypes k ($k \geq 2$).

```
postprob_DE_matr <- postprob_DE(BUSfits)

## Showing the posterior probability of being differentiallyexpressed

##                 for gene g in subtype k (k>=2) compared to subtype 1.

## The output format is a matrix.
```

```
## Each row represents a gene, and each column corresponds to a subtype.
```

6 Visualize the Adjusted Genomic Data

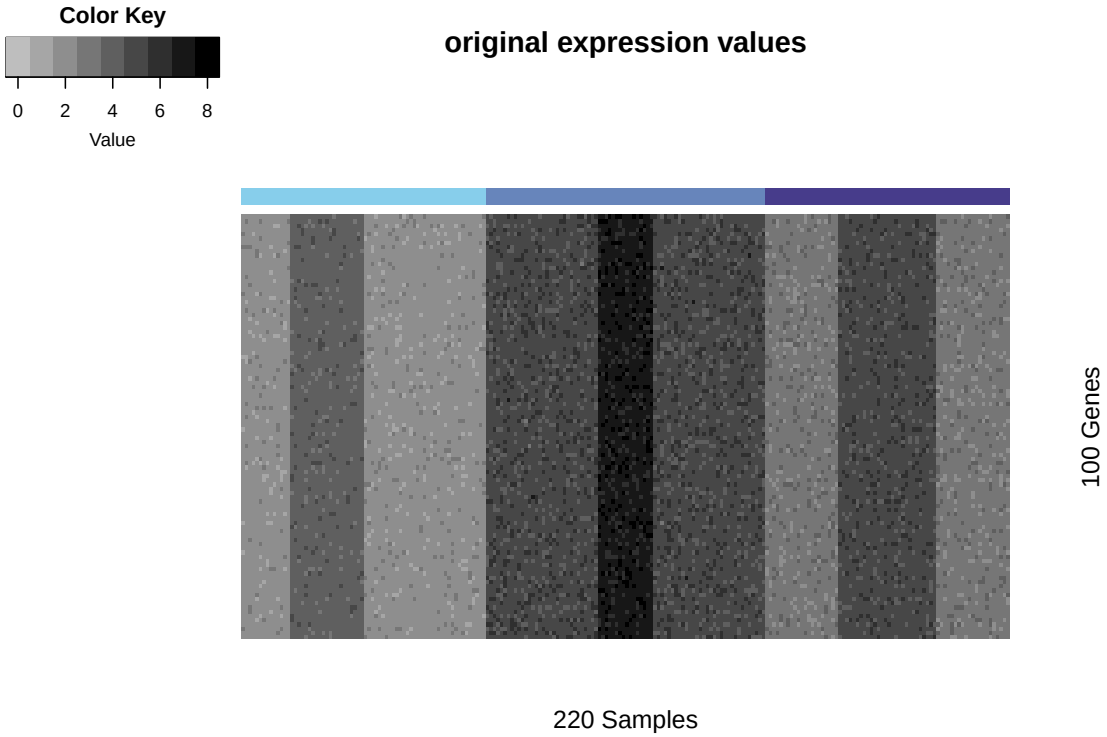
The function `adjusted_values` adjusts the batch effects for the original data.

```
adjusted_data <- adjusted_values(BUSfits, BUSexample_data)

## The output format is a list with length equal to the batch number.

## Each element of the list is the adjusted gene expression matrix.

## In the matrix, each row represents a gene, and each column corresponds to a
sample.
```

The message is a reminder of the output format. Subsequently, we can compare the original data suffering from batch effects and the adjusted data with the batch effects removed.

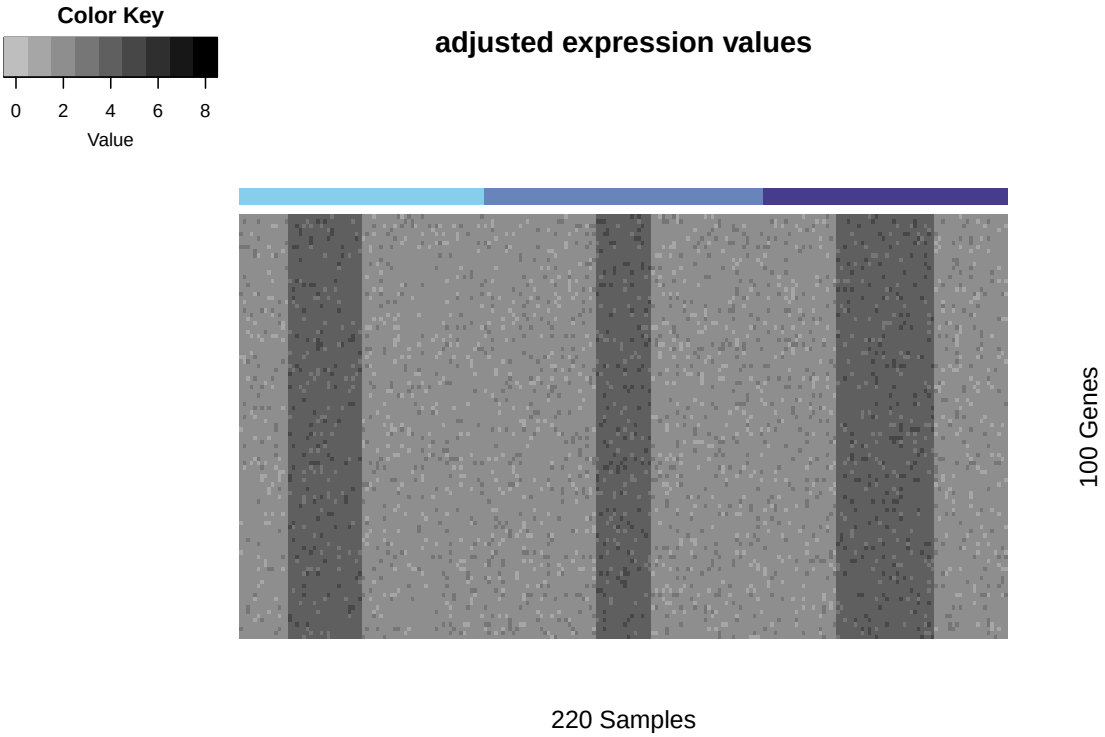
The function `visualize_data` plots a heatmap for the expression values across batches. The user can specify the argument `gene_ind_set`, which is a vector, to select the genes to be displayed in the heatmap.

```
#only show the first 100 genes
visualize_data(BUSexample_data, title_name = "original expression values",
                                                        gene_ind_set = 1:100)
```




```
#try the following command to show the whole set of genes
#visualize_data(BUSexample_data, title_name = "original expression values",
# gene_ind_set = 1:2000)
```

```
#only show the first 100 genes
visualize_data(adjusted_data, title_name = "adjusted expression values",
                                                    gene_ind_set = 1:100)
```




```
#try the following command to show the whole set of genes
#visualize_data(adjusted_data, title_name = "adjusted expression values",
# gene_ind_set = 1:2000)
```

In these two heatmaps, the top bar indicates the batch origin for each sample. Samples under the same colour are from the same batch. The batch effects present in the original data are correctly removed, and only the biological variability is kept.

7 Model Selection using BIC

If we have no prior knowledge about the subtype number, we can vary the argument `n.subtypes` in the function `BUSgibbs`, e.g., from 2 to 10 and identify the underlying true subtype number K as the one that achieves the minimal BIC.

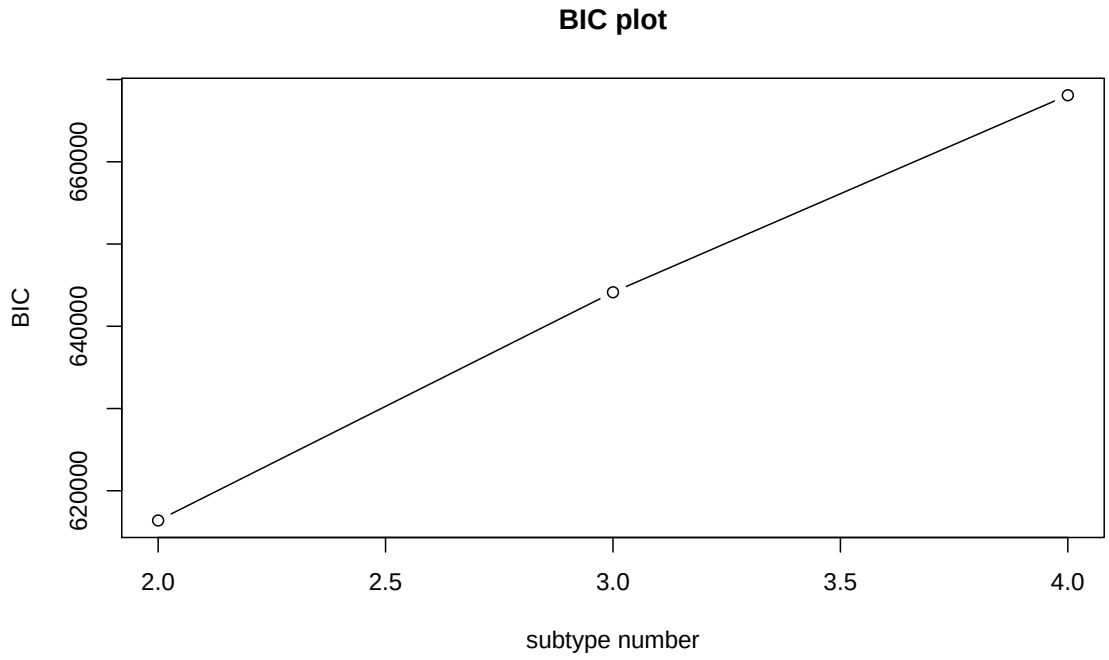
In BUScorrect R package, one can obtain the BIC value as follows.

```
BIC_val <- BIC_BUS(BUSfits)

## BIC is 644136.818607818

## The output is a scalar.
```

In this example, the underlying true subtype number is three. For an illustration, we vary the `n.subtypes` from 2 to 4.

```
BIC_values <- NULL
for(k in 2:4){
        set.seed(123)
        BUSfits <- BUSgibbs(Data = BUSexample_data, n.subtypes = k, n.iterations = 300,
                                                showIteration = FALSE)
        BIC_values <- c(BIC_values, BIC_BUS(BUSfits))
}

##  running the Gibbs sampler ...

##  The Gibbs sampler takes:  0.309 mins

##  calculating posterior means and posterior modes...

##  calculating BIC...

## BIC is 616387.188250276

## The output is a scalar.
```

```
##  running the Gibbs sampler ...
##  The Gibbs sampler takes:  0.328 mins
##  calculating posterior means and posterior modes...
##  calculating BIC...
## BIC is 644136.818607818
## The output is a scalar.
##  running the Gibbs sampler ...
##  The Gibbs sampler takes:  0.345 mins
##  calculating posterior means and posterior modes...
##  calculating BIC...
## BIC is 668089.736809492
## The output is a scalar.
plot(2:4, BIC_values, xlab="subtype number", ylab="BIC", main="BIC plot", type="b")
```



The BIC attains the minimum at `n.subtypes`$= 3$, thus correctly recovering the true subtype number.

References

[1] Jeffrey T Leek, Robert B Scharpf, Héctor Corrada Bravo, David Simcha, Benjamin Langmead, et al. Tackling the widespread and critical impact of batch effects in high-throughput data. *Nature Reviews Genetics*, 11(10):733–739, 2010.

[2] W Evan Johnson, Cheng Li, and Ariel Rabinovic. Adjusting batch effects in microarray expression data using empirical bayes methods. *Biostatistics*, 8(1):118–127, 2007.

[3] Jeffrey T Leek and John D Storey. Capturing heterogeneity in gene expression studies by surrogate variable analysis. *PLoS Genetics*, 3(9):e161, 2007.

[4] Xiangyu Luo and Yingying Wei. Batch effects correction with unknown subtypes. *Accepted in Journal of the American Statistical Association*.

[5] Martin Morgan, Valerie Obenchain, Jim Hester, and Herv'e Pag'es. *SummarizedExperiment: SummarizedExperiment container*, 2018. R package version 1.10.1.

[6] Zhiguang Huo, Ying Ding, Silvia Liu, Steffi Oesterreich, and George Tseng. Meta-analytic framework for sparse k-means to identify disease subtypes in multiple transcriptomic studies. *Journal of the American Statistical Association*, 111(513):27–42, 2016.