MBASIC: Matrix Based Analysis for State-space Inference and Clustering

Chandler Zuo[*]
Sündüz Keleş[†]

Contents

1 Introduction

This document provides an introduction to the power analysis of ChIP-seq data with the `MBASIC` package. This `R` package implements **MBASIC** which stands for **M**atrix **B**ased **A**nalysis for **S**tate-space **I**nference and **C**lustering in [Zuo and Keleş, 2014]. **MBASIC** provides a Bayesian framework for clustering units based on their infered states over a set of experimental conditions.

`MBASIC` is especially useful for integrative analysis for ChIP-seq experiments. In this case, a set of prespecified loci is clustered based on their activities over celltypes and transcription factors. We build a pipeline in the `MBASIC` package and will focus on this pipeline in this vignette. We will introduce the general functionalities for the **MBASIC** model at the end of the vignette.

2 MBASIC Pipeline for Sequencing Data

2.1 Workflow

The `MBASIC` framework consists of five major steps:

1. *Matching ChIP replicate files with their inputs:* This step matches ChIP replicate files with their matching input files;

2. *Calculating mapped counts and genomic scores on the target loci:* This step calculates the mapped counts from each ChIP and input replicate files on each of the target locus;

3. *Calculating the mappability and GC-content scores for the target loci:* This step computes the average mappability and GC scores for loci from external files;

[*]Department of Statistics, 1300 University Avenue, Madison, WI, 53706, USA.
[†]Departments of Statistics and of Biostatistics and Medical Informatics, 1300 University Avenue, Madison, WI, 53706, USA.

4. *Estimating the background means:* This step uses regression models for the input data and the M, GC scores to estimate the background means for the ChIP replicates;
5. *Fitting MBASIC model:* This step fits the MBASIC model to identify the binding states for each locus and cluster the loci based on their binding states across different conditions.

`MBASIC` integrates Step 2-5 in a single function called "MBASIC.pipeline". For Step 1 `MBASIC` provides a function "ChIPInputMatch" that assists matching the ChIP files with input files based on the naming convention of the ENCODE datasets. We have found that in practice, more often than not, some violations to the ENCODE file name conventions always occur, and manual adjustments to the results of our automated matching are inevitable. Therefore, we do not integrate this function in "MBASIC.pipeline".

2.2 Match ChIP and Input Datasets

To illustrate Step 1 we first generate a set of synthetic data.

`MBASIC` package provides a function "generateSyntheticData" to assist our demo. This function generates synthetic BED data for ChIP and input samples, as well as mappability and GC scores in a directory specified by the "dir" argument. It also generates a target set of loci for our analysis. By default, the number of loci is 100, each with size 20 bp. All data are generated across 5 chromosomes, each with size 10K bp. ChIP data are from 2 celltypes, and for each celltype there are K=5 TFs. Under each condition randomly 1-3 replicates for the ChIP data are generated. All ChIP data from the same celltype are matched to the same set of 3 input replicates.

```
> library(MBASIC)
> target <- generateSyntheticData( dir = "syntheticData" )
> target
GRanges with 100 ranges and 0 metadata columns:
        seqnames       ranges strand
           <Rle>    <IRanges>  <Rle>
    [1]     chr1 [  80,  100]      *
    [2]     chr1 [ 400,  420]      *
    [3]     chr1 [ 880,  900]      *
    [4]     chr1 [1000, 1020]      *
    [5]     chr1 [1240, 1260]      *
    ...      ...          ...    ...
   [96]     chr5 [8240, 8260]      *
   [97]     chr5 [8280, 8300]      *
   [98]     chr5 [9400, 9420]      *
   [99]     chr5 [9560, 9580]      *
  [100]     chr5 [9680, 9700]      *
  ---
  seqlengths:
   chr1 chr2 chr3 chr4 chr5
     NA   NA   NA   NA   NA
> system( "ls syntheticData/*/*" )
```

Function "ChIPInputMatch" assists Step 1. It reads all files with suffix ".bed" in directories specified by the argument ".dir", and matches the files assuming ENCODE naming convention. It looks up files up to the number of levels of subdirectories specified by "depth". The output of this function contains multiple columns. The first column contains the file name for each ChIP replicate. The second column is the initial string for the matching input replicates, because for each ChIP replicate there are possibly multiple input replicates. The rest of the columns contains information for lab, experiment identifier, factor and control identifier. This information is parsed from the file names.

```
> tbl <- ChIPInputMatch( dir = paste( "syntheticData/",
+                          c( "chip", "input" ), sep = "" ),
+                       suffix = ".bed", depth = 5 )
> head( tbl )
                                                   chipfile
1 syntheticData/chip/wgEncodeLabExpCell1Fac1CtrlAlnRep1.bed
2 syntheticData/chip/wgEncodeLabExpCell1Fac1CtrlAlnRep2.bed
3 syntheticData/chip/wgEncodeLabExpCell1Fac2CtrlAlnRep1.bed
```

```
4 syntheticData/chip/wgEncodeLabExpCell1Fac2CtrlAlnRep2.bed
5 syntheticData/chip/wgEncodeLabExpCell1Fac2CtrlAlnRep3.bed
6 syntheticData/chip/wgEncodeLabExpCell1Fac3CtrlAlnRep1.bed
                                          inputfile
1 syntheticData/input/wgEncodeLabExpCell1InputCtrl
2 syntheticData/input/wgEncodeLabExpCell1InputCtrl
3 syntheticData/input/wgEncodeLabExpCell1InputCtrl
4 syntheticData/input/wgEncodeLabExpCell1InputCtrl
5 syntheticData/input/wgEncodeLabExpCell1InputCtrl
6 syntheticData/input/wgEncodeLabExpCell1InputCtrl
  lab experiment  cell factor control
1 Lab        Exp Cell1   Fac1    Ctrl
2 Lab        Exp Cell1   Fac1    Ctrl
3 Lab        Exp Cell1   Fac2    Ctrl
4 Lab        Exp Cell1   Fac2    Ctrl
5 Lab        Exp Cell1   Fac2    Ctrl
6 Lab        Exp Cell1   Fac3    Ctrl
```

We also need to prepare the following meta data information. Below the vector "n" specifies the number of replicates within the same experimental conditions.

```
> conds <- paste( tbl$cell, tbl$factor, sep = "." )
```

Now we are in a position to continue the next steps in the pipeline. There are two ways to execute these steps: (1) use function "MBASIC.pipeline", which wraps up all the consecutive steps; or (2) execute each step separately.

2.3 Pipeline Execution

The following code calls the function "MBASIC.pipeline":

```
> MBASIC.fit <- MBASIC.pipeline( chipfile = tbl$chipfile,
+                                 inputfile = tbl$inputfile,
+                                 input.suffix = ".bed",
+                                 target = target,
+                                 format = "BED",
+                                 fragLen = 150,
+                                 pairedEnd = FALSE,
+                                 unique = TRUE,
+                                 m.prefix = "syntheticData/mgc/",
+                                 m.suffix = "_M.txt",
+                                 gc.prefix = "syntheticData/mgc/",
+                                 gc.suffix = "_GC.txt",
+                                 fac = conds,
+                                 struct = NULL, J = 3,
+                                 family = "negbin",
+                                 burnin = 20, maxitr = 100,
+                                 tol = 1e-4, nsig = 2,
+                                 datafile = NULL )
```

"MBASIC.pipeline" requires a number of arguments. We list these arguments in Table 1. For details users are recommended to read our manual.

2.4 Stepwise Execution

Alternatively, each step in the pipeline can be executed separately.

```
> ## Step 2: Generate mapped count matrices
> dat <- generateReadMatrices( chipfile = tbl$chipfile,
+                             inputfile = tbl$inputfile,
+                             input.suffix = ".bed",
+                             target = target,
```

Table 1: Arguments for the "MBASIC.pipeline" function.

Data Sources	
chipfile	A string vector for the ChIP files.
inputfile	A string vector for the matching input files. The length must be the same as "chipfile".
input.suffix	A string for the suffix of input files. If NULL, "inputfile" will be treated as the full names of the input files. Otherwise, all inputfiles with the initial "inputfile" and this suffix will be merged.
format	A string specifying the type of the file. Currently two file types are allowed: "BAM" or "BED". Default: "BAM".
m.prefix	A string for the prefix of the mappability files.
m.suffix	A string for the suffix of the mappability files. See our man files for more details. Default: NULL.
gc.prefix	A string for the prefix of the GC files.
gc.suffix	A string for the suffix of the GC files. See our man files for more details. Default: NULL.
Genomic Information	
target	A GenomicRanges object for the target intervals where the reads are mapped.
fragLen	Either a single value or a 2-column matrix of the fragment lengths for the chip and input files. Default: 150.
pairedEnd	Either a boolean value or a 2-column boolean matrix for whether each file is a paired-end data set. Currently this function only allows "BAM" files for paired-end data. Default: FALSE.
unique	A boolean value for whether only reads with distinct genomic coordinates or strands are mapped. Default: TRUE.
Model Parameters	
fac	A vector of length N for the experimental condition of each ChIP replicate.
struct	A matrix indicating the levels of the signal matrix.
J	The number of clusters to be identified.
family	The distribution of family to be used. Either "lognormal" or "negbin". See our man files for more information.
nsig	The number of mixture components for the distribution of the signal state. Default: 2.
Tuning Parameters	
burnin	An integer value for the number of iterations in initialization. Default: 20.
maxitr	The maximum number of iterations in the E-M algorithm. Default: 100.
tol	Tolerance for error in checking the E-M algorithm's convergence. Default: 1e-04.
datafile	The location to save the count matrices.

```
+                             format = "BED",
+                             fragLen = 150,
+                             pairedEnd = FALSE,
+                             unique = TRUE )
> ## Step 3: Compute M and GC scores
> target <- averageMGC( target = target,
+                      m.prefix = "syntheticData/mgc/",
+                      m.suffix = "_M.txt",
+                      gc.prefix = "syntheticData/mgc/",
+                      gc.suffix = "_GC.txt" )
> ## Step 4: Compute the background means
> Mu0 <- bkng_mean( inputdat = dat$input,
+                  target = target,
+                  family = "negbin" )

> ## Step 5: Fit an MBASIC model
> MBASIC.fit <- MBASIC.binary( Y = t( dat$chip ),
+                             Mu0 = t( Mu0 ),
+                             fac = conds,
+                             J=3,
+                             zeta=0.2,
+                             maxitr = 100,
+                             burnin = 20,
+                             outfile=NULL,
+                             out=NULL,
+                             init.mod = NULL,
+                             struct = NULL,
+                             family="negbin",
+                             tol = 1e-4,
+                             nsig = 2 )
```

2.5 The “MBASICFit” Class

The outputs of both “MBASIC.binary” and “MBASIC.pipeline” functions are of S-4 class “MBASICFit”.

```
> showClass( "MBASICFit" )
Class "MBASICFit" [package "MBASIC"]

Slots:

Name:         Theta          W          V
Class:       matrix     matrix     matrix

Name:             Z          b        aic
Class:       matrix    numeric    numeric

Name:           bic       aicc        lik
Class:      numeric    numeric    numeric

Name:        alllik       zeta         Mu
Class:      numeric    numeric     matrix

Name:         Sigma     sigma0          e
Class:       matrix    numeric    numeric

Name:         probz          P  converged
Class:      numeric     matrix    logical

Name:     Theta.err        ARI      W.err
Class:      numeric    numeric    numeric
```

```
Name:  MisClassRate
Class:      numeric
```

Slot "Theta" is a matrix for the estimated state, where each row corresponds to an experimental condition, and each column corresponds to a locus. Each entry is the probability for the locus to be un-binding at the corresponding state.

```
> dim( MBASIC.fit@Theta )
[1]  10 300
> rownames( MBASIC.fit@Theta )
 [1] "Cell1.Fac1" "Cell1.Fac2" "Cell1.Fac3"
 [4] "Cell1.Fac4" "Cell1.Fac5" "Cell2.Fac1"
 [7] "Cell2.Fac2" "Cell2.Fac3" "Cell2.Fac4"
[10] "Cell2.Fac5"
> head( MBASIC.fit@Theta[ 1, ] )
[1] 1.00000000 1.00000000 1.00000000 0.11893983
[5] 1.00000000 0.08438015
```

Slot "Z" is a matrix for the posterior probablity of each locus to belong to each cluster. The rows correspond to the loci and the columns correspond to the clusters.

```
> dim( MBASIC.fit@Z )
[1] 100   3
> head( MBASIC.fit@Z )
          [,1]      [,2]      [,3]
[1,] 0.2102689 0.5740238 0.2157073
[2,] 0.2102689 0.5740238 0.2157073
[3,] 0.2102689 0.5740238 0.2157073
[4,] 0.2101648 0.5742544 0.2155808
[5,] 0.2102688 0.5740240 0.2157072
[6,] 0.4506035 0.3993366 0.1500599
```

Slot "b" is a vector for the probability of each locus not to belong to any cluster.

```
> length( MBASIC.fit@b )
[1] 100
> head( MBASIC.fit@b )
[1] 1.0000000 1.0000000 1.0000000 0.9994259
[5] 0.9999995 0.6984494
```

Slot "W" is a matrix for the probability of loci in each group to be unenriched at each condition.

```
> rownames( MBASIC.fit@W )
 [1] "Cell1.Fac1" "Cell1.Fac2" "Cell1.Fac3"
 [4] "Cell1.Fac4" "Cell1.Fac5" "Cell2.Fac1"
 [7] "Cell2.Fac2" "Cell2.Fac3" "Cell2.Fac4"
[10] "Cell2.Fac5"
> dim( MBASIC.fit@W )
[1] 10  3
> head( MBASIC.fit@W )
                   [,1]         [,2]         [,3]
Cell1.Fac1 0.0000000001 8.034974e-01 1.000000e-10
Cell1.Fac2 0.9999999999 1.000000e+00 1.000000e+00
Cell1.Fac3 0.9999999999 2.319580e-07 1.000000e-10
Cell1.Fac4 0.1073035701 7.970337e-01 1.000000e+00
Cell1.Fac5 0.9999999999 1.000000e+00 7.477143e-10
Cell2.Fac1 0.9999999999 6.313526e-01 5.884637e-01
```

2.6 Advanced Model Initialization

An important argument that is accessible by the "MBASIC.binary" function but not "MBASIC.pipeline" is "init.mod". This argument allows the user to pass a "MBASICFit" object, whose values are used to initialize the parameters in this function. It can be useful in two conditions:

1. The model fitted by a previous call of function "MBASIC.binary" has not yet converged, and the user wishes to increase the number of iterations;
2. The user wants to fit a MBASIC model with a larger number of clusters ("J") from a previously fitted model;
3. The user wants to fit a MBASIC model with a different structural constraints ("struct") from a previously fitted model.

In both cases, "MBASIC.binary" uses the results from "init.mod" before starting its iterations. This is significantly time saving than restarting the model fitting from scratch. However, this argument is dangerous if the user passes along a fitted model using different data sources or model structures. To avoid that, the user need to check that the arguments used to get "init.mod" and in the current "MBASIC.binary" must be consistent following rules below (which currently cannot be checked by our codes):

1. "Y", "Mu0", "fac", "family", "nsig" must be the same;
2. "J" for the "MBASIC.binary" function must be larger than or equal to the value used in "init.mod".

```
> ## Fit a MBASIC model with 4 clusters
> MBASIC.binary( Y = t( dat$chip ),  Mu0 = t( Mu0 ),
+               fac = conds,  J=4,  zeta=0.2,
+               maxitr = 100, burnin = 20,
+               init.mod = MBASIC.fit,
+               struct = NULL, family="negbin",
+               tol = 1e-4,  nsig = 2 )
> ## Fit a MBASIC model with more iterations
> MBASIC.binary( Y = t( dat$chip ),  Mu0 = t( Mu0 ),
+               fac = conds,  J=3,  zeta=0.2,
+               maxitr = 200, burnin = 20,
+               init.mod = MBASIC.fit,
+               struct = NULL, family="negbin",
+               tol = 1e-4,  nsig = 2 )
```

3 General Functionalities

The `MBASIC` package also provides additional functions to simulate and fit general MBASIC models with $S \geq 2$ states.

3.1 Simulation for General MBASIC Models

Function "MBASIC.sim" simulates data with "I" units with "J" clusters. The "S" argument specifies the number of different states, and "zeta" is the proportion of unclustered units. "fac" specifies the condition for each experiment. The "xi" argument relates to the magnitude of the simulated data. Its detailed description is included in our manual.

```
> ## Simulate data across I=1000 units with J=3 clusters
> ## There are S=3 states
> dat.sim <- MBASIC.sim( xi = 2, family = "lognormal",
+                       I = 1000, fac = rep( 1:10, each = 2 ),
+                       J = 3, S = 3, zeta = 0.1 )
```

The "MBASIC.sim" function returns a list object. The "Y" field contains the simulated data matrix at each unit (column) for each experiment (row). The "Theta" field is the matrix for the states for each unit (column) and each experimental condition (column). The "W" field is a matrix with dimensions KS × J, where the (S(k-1)+s,j)-th entry is the probability that units in the j-th cluster have state s under the k-th condition.

```
> names( dat.sim )
 [1] "Theta"      "Y"          "W"
 [4] "Z"          "delta"      "zeta"
 [7] "prior.mean" "prior.sd"   "stdev"
[10] "Mu"         "bkng"       "snr"
[13] "non.id"
> dim( dat.sim$Y )
[1]   20 1000
> dim( dat.sim$W )
[1] 30  3
> dim( dat.sim$Theta )
[1]   10 1000
```

3.2 Fitting General MBASIC Models

The general MBASIC model is fitted by the function “MBASIC”. The arguments here are similar to the “MBASIC.binary” function. The arguments that are not common to both functions are::

Argument	Meaning	“MBASIC.binary”	“MBASIC”
“nsig”	Number of signal components for the binding state.	Y	N
“Mu0”	Background means.	Y	N
“S”	Number of states.	N	Y
“method”	Fitting method.	N	Y
“para”	True parameters.	N	Y

The “method” argument specifies the fitting algorithm to be used. Currently, thre algorithms are provided. “em” is the nested E-M algorithm that is also used by “MBASIC.binary”; “2em” and “naive” are benchmark methods. Both “2em” and “naive” fit the model in two phases. In Phase 1 they estimate the state space matrix, and in Phase 2 “2em” uses a mixture model for clustering the units while “naive” uses hierarchical clustering.

The “para” argument allows users to pass the true model parameters into the model. In the following codes we pass the simulated list object “dat.sim”. In that case, the output of the model contains a few slots that calculate the estimation error:

- *ARI*: Adjusted Rand Index;
- *W.err*: The mean squared error in matrix W;
- *Theta.err*: The mean squared error in state estimation;
- *MisClassRate*: The mis-classification rate.

For more details of the algorithms, as well as these metrics to assess model fitting, users may refer to [Zuo and Keleş, 2014].

```
> dat.sim.fit <- MBASIC( Y = dat.sim$Y, S = 3,
+                        fac = rep( 1:10, each = 2),
+                        J = 3, maxitr = 3,
+                        para = dat.sim, family = "lognormal",
+                        method = "em",
+                        zeta = 0.1, tol = 1e-04)
> dat.sim.fit@ARI
> dat.sim.fit@W.err
> dat.sim.fit@Theta.err
> dat.sim.fit@MisClassRate
```

3.3 Degenerate MBASIC Models

In a degenerate MBASIC model, the states for each unit under each condition are directly observed. “MBASIC.sim.state” and “MBASIC.state” functions allows users to simulate and fit such models. The usage of these functions are similar to functions “MBASIC.sim” and “MBASIC”.

“MBASIC.sim.state” simulates data from a degenerate MBASIC model. Most arguments are similar to “MBASIC.sim”. “MBASIC.sim.state” does not need arguments “fac” and “family”, but it needs the “K” argument, specifying the number of experimental conditions.

```
> state.sim <- MBASIC.sim.state( I = 1000, K = 10,
+                                J = 4, S = 3,
+                                zeta = 0.1)
```

“MBASIC.state” fits a degenerate MBASIC model. Compared to function “MBASIC”, it does not need arguments “Y” and “family”. Instead, it needs the argument “Theta” to pass the observed states.

```
> state.sim.fit <- MBASIC.state( Theta = state.sim$Theta, J = 4,
+                                method = "2em", zeta = 0.1,
+                                maxitr = 100, tol = 1e-04 )
```

4 Session Information

```
R version 3.0.2 (2013-09-25)
Platform: x86_64-redhat-linux-gnu (64-bit)

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

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

other attached packages:
 [1] MBASIC_0.99.0        MASS_7.3-29
 [3] GenomicRanges_1.14.4 XVector_0.2.0
 [5] IRanges_1.20.7       BiocGenerics_0.8.0
 [7] mclust_4.3           Rcpp_0.11.1
 [9] msm_1.3              gtools_3.3.1

loaded via a namespace (and not attached):
[1] expm_0.99-1.1    grid_3.0.2
[3] lattice_0.20-29  Matrix_1.1-3
[5] mvtnorm_0.9-9998 splines_3.0.2
[7] stats4_3.0.2     survival_2.37-7
[9] tools_3.0.2
```

References

[Zuo and Keleş, 2014] Zuo, C. and Keleş, S. (2014). A bayesian framework for state space matrix clustering. Submitted.