User Manual of ChIP-BIT 2.0

Xi Chen

September 16, 2015

Introduction

Bayesian Inference of Target genes using ChIP-seq profiles (ChIP-BIT) is developed to reliably detect TFBSs and their target genes by modeling binding signal intensities (of both sample and input ChIP-seq data) and binding locations of TFBSs jointly. Specifically, a unique Gaussian mixture model is used to capture both binding and background signals in sample data. As a unique feature of ChIP-BIT, background signals are modeled by a local Gaussian distribution, which can be accurately estimated from input data. The effect of TFBSs on target gene transcription is also modeled based on their binding locations for target gene identification. An expectation-maximization (EM) algorithm is then used to estimate the posterior probability of binding events for TFBSs.

I. Programming language

A workflow of the ChIP-BIT implementation is shown in Fig. 1; a subset (chr1) of the PBX1 and its input ChIP-seq data are included in the package for users to run a demo of ChIP-BIT.

Steps 1-3 of ChIP-seq data preprocessing (as described in the supplementary material) of ChIP-BIT are achieved using a binary file "**PeakSeq_first_pass_intensity_output**". All source codes written by C++ language are made available at http://www.cbil.ece.vt.edu/software.htm.

Steps 4-5 and major functions of ChIP-BIT are programmed using MATLAB and R respectively, which are also made available on our website at http://www.cbil.ece.vt.edu/software.htm.

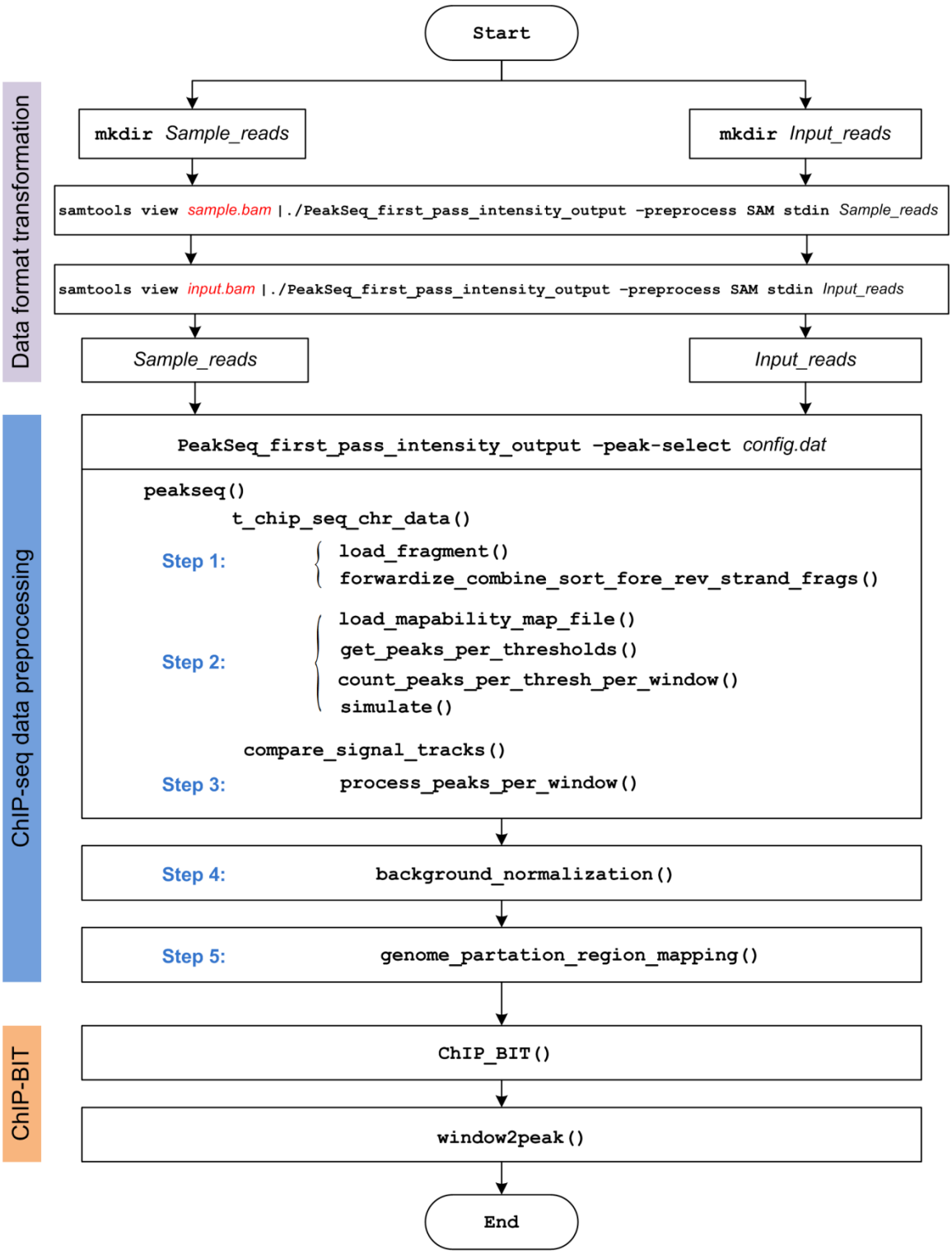


Fig. 1. Flowchart of the ChIP-BIT implementation.

II. ChIP-seq data analysis using MATLAB version of ChIP-BIT

The MARLAB script of ChIP-BIT has been tested using Ubuntu 12.04 64 bit MARLAB 2012b and MARLAB 2014a.

1. ChIP-seq data transformation (from bam files to PeakSeq readable files)

A folder is created for each ChIP-seq bam file to record 5' start locations of reads for each chromosome.

```
% create the dictionary for PeakSeq pre-processed reads
system('mkdir Sample_chr1_reads');
system('mkdir Input_chr1_reads');
```

Sample or input ChIP-seq data are dumped into the system using "**samtools**" and further processed using "**PeakSeq_first_pass_intensity_output**" to report 5' start locations of reads for each chromosome.

```
% convert bam files into PeakSeq readable profiles which requires samtools
installed
system('samtools view PBX1_chr1.bam |./PeakSeq_first_pass_intensity_output -
preprocess SAM stdin Sample_chr1_reads');
```

```
Read length       Strand      5' location
36M               F           10001
36M               F           10002
36M               F           10006
36M               F           10014
36M               F           10023
…
36M               R           10095
36M               F           10100
36M               R           10101
36M               F           10102
36M               R           10103
36M               F           10106
…
```

Two files, "**1_mapped_reads.txt**" and "**chr_ids.txt**", will be created in a folder called "**Sample_chr1_reads**" to record read information of chromosome 1. If whole genome is tested, chromosome 1~22, X, Y and M will be recorded individually in file "**xx_mapped_reads.txt**" where xx refers to the chromosome id.

```
system('samtools view Input_chr1.bam |./PeakSeq_first_pass_intensity_output -
preprocess SAM stdin Input_chr1_reads');
```

```
Read length       Strand      5' location
36M               F           10003
36M               F           10006
```

```
36M                F            10006
36M                F            10007
36M                F            10011
…
36M                R            10137
36M                R            10137
36M                F            10141
36M                R            10144
36M                R            10155
36M                F            10157
…
```

Two files, "**1_mapped_reads.txt**" and "**chr_ids.txt**", will be created in a folder called "**Input_chr1_reads**" to record read information of chromosome 1. If whole genome is tested, chromosome 1~22, X, Y and M will be recorded individually in file "**xx_mapped_reads.txt**" where xx refers to the chromosome id.

2. ChIP-seq data preprocessing

We use "**PeakSeq_first_pass_intensity_output**" to perform Steps 1-3 of ChIP-seq data preprocessing (as described in supplementary material) and use two MATLAB functions to perform Steps 4 and 5, respectively.

Step 1:

To perform Step 1, we use function "**t_chip_seq_chr_data()**" as follows:

```
cur_chr_data = t_chip_seq_chr_data(chip_seq_reads_data_dirs,
input_reads_data_dirs, chr_fps->at(i_chr), mappability_map_fp,
enrichment_mapped_fragment_length);
```

The value for each input parameter is set up as follows:

```
chip_seq_reads_data_dirs = 'Sample_chr1_reads';
input_reads_data_dirs = 'Input_chr1_reads';
i_chr = 1;
mappability_map_fp = 'hg19_mappability_20k.txt';
enrichment_mapped_fragment_length = 200;
```

Within function "**t_chip_seq_chr_data()**", for each chromosome of the sample ChIP-seq data, we use function "**load_fragments()**" to load forward and reverse read tags 5' locations from variable "mapped_chip_seq_reads_fp" and further store them in variables "cur_chr_chip_seq_fore_frags" and "cur_chr_chip_seq_rev_frags", respectively.

```
sprintf(mapped_chip_seq_reads_fp, "%s/%s_mapped_reads.txt",
chip_seq_reads_data_dirs->at(i_chip_seq_dir), this->chr_id);
load_fragments(mapped_chip_seq_reads_fp, this->cur_chr_chip_seq_fore_frags,
this->cur_chr_chip_seq_rev_frags);
```

Then, we extend each read tag to a 200 bps long fragment and cluster forward or reverse fragments together if they overlap from each other. This step is prepared for further candidate region identification.

```
forwardize_combine_sort_fore_rev_strand_frags(cur_chr_chip_seq_fore_frags,
cur_chr_chip_seq_rev_frags, enrichment_mapped_fragment_length)
```

Finally, since input ChIP-seq data are not used for candidate region searching (using the first pass of the PeakSeq package), we only load all read tags from "mapped_control_reads_fp" and store in variables "cur_chr_fore_control_frags" and "cur_chr_rev_control_frags" as follows:

```
sprintf(mapped_control_reads_fp, "%s/%s_mapped_reads.txt",
input_reads_data_dirs->at(i_input_dir), this->chr_id);
load_fragments(mapped_control_reads_fp, cur_chr_fore_control_frags,
cur_chr_rev_control_frags);
```

Step 2:

To perform Step 2, within function "**t_chip_seq_chr_data()**", we use a sub-function "**load_mapability_map_file()**" to load genome mappability file "**hg19_mappability_20k.txt**".

```
mappability_map_fp = 'hg19_mappability_20k.txt';

chr          index        number of unique nucleotide
1            0            3206
1            1            611
1            2            3453
1            3            4275
1            4            5034
1            5            2034
1            6            2558
…
```

For each segment, based on the mappability situation, we calculate a threshold for candidate region selection. In detail, we set the threshold using each value from "min_thresh" to "max_thresh" and calculate the number of continuous regions in each segment.

```
peak_regions_per_threshold = cur_chr_data->chip_seq_enr_prof-
>get_peaks_per_thresholds(min_thresh, max_thresh, min_gap_per_bw_peaks);
n_peaks_per_thresh_per_window = count_peaks_per_thresh_per_window
(cur_chr_data->n_meg_wins(),peak_regions_per_threshold, min_thresh,
max_thresh);
```

The values of parameters of this function are set as follows:

```
min_thresh = 1;
max_thresh = 1000;
min_gap_per_bw_peaks = 200;
```

We simulate a null background model using function "**simulate()**" and calculate a threshold meeting the FDR requirement for each segment.

```
thresholds_per_win = simulate(cur_chr_data, n_peaks_per_thresh_per_window,
n_sims, target_FDR, enrichment_mapped_fragment_length, min_gap_per_bw_peaks,
min_thresh, max_thresh);
```

Values of parameters in this function are set as follows:

```
n_sims = 10;
target_FDR = 0.05;
enrichment_mapped_fragment_length = 200;
min_gap_per_bw_peaks = 200;
min_thresh = 1;
max_thresh = 1000;
```

Step 3:

To perform Step 3, based on the calculated threshold from Step 2, in each segment we identify a number of candidate regions using function "**compare_signal_tracks()**" as follows:

```
cur_chr_annotated_peaks = compare_signal_tracks(cur_chr_data,
peak_regions_per_threshold, n_peaks_per_thresh_per_window,
enrichment_mapped_fragment_length, min_gap_per_bw_peaks, thresholds_per_win,
peak_profs_dump_dir, min_thresh, max_thresh);
```

Furthermore, we use function "**process_peaks_per_window()**" to report read count and read intensity in sample and input ChIP-seq data, respectively, for each candidate region. The chromosome, start, summit and end locations for each region are also reported.

```
process_peaks_per_window(i_win, cur_win_thresh, cur_chr_data,
peak_regions_per_threshold[cur_win_thresh - min_thresh],
last_peak_list_indices[cur_win_thresh - min_thresh], last_peak_start,
last_peak_end, min_gap_per_bw_peaks, enrichment_mapped_fragment_length,
last_chip_seq_frag_i, last_control_frag_i, cur_chr_annotated_peaks);
```

All above major functions used in Steps 1-3 have been compiled into a binary file "**PeakSeq_first_pass_intensity_output**". We can directly call this binary file in MATLAB under Linux system as follows:

```
PeakSeq_firstpass_Region_cmd = './PeakSeq_first_pass_intensity_output -
peak_select config.dat';

system(PeakSeq_firstpass_Region_cmd);
```

We set up parameter values in the config.dat file as follows:

```
Experiment_id                           Candidate_regions
Enrichment_mapped_fragment_length       200
target_FDR                              0.05
N_Simulations                           10
Minimum_interpeak_distance              200
Mappability_map_file                    hg19_mappability_20k.txt
ChIP_Seq_reads_data_dirs                Sample_reads
Input_reads_data_dirs                   Input_reads
Background_model                        Simulated
```

A temporary file "**Candidate_regions_chr1.txt**" is created to store all information.

```
Chromosome    Begin  End    Sample_boudary      Sample_central      Sample_count
       Input_boudary Input_central Input_count   Max_Position
1      150601421     150602387     0.492  88.238 1793   0.024  4.281  87     150602387
1      151254060     151254918     0.771  93.612 1700   0.049  6.002  109    151254918
1      1207738       1208355       0.975  87.556 1168   0.039  3.523  47     1208355
1      110881200     110882214     0.423  57.378 1221   0.021  2.867  61     110882214
1      23405332      23406309      3.116  47.955 985    0.038  0.584  12     23406309
1      154955463     154956021     1.809  78.536 955    0.034  1.480  18     154956021
…
```

Step 4:

Before peak calling can be made, input ChIP-seq data should be properly normalized against sample ChIP-seq data. We load candidate regions of chromosome 1 to the working space of MATLAB as follows:

```
%load candidate regions at whole genome
[WG_Region.Chr,WG_Region.point1,WG_Region.point2,WG_Region.Sample_boundary,WG
_Region.Sample_central,WG_Region.Sample_count,WG_Region.Input_boundary,WG_Reg
ion.Input_central,WG_Region.Input_count,WG_Region.Max_Position]=
textread('Candidate_regions_chr1.txt', '%s %d %d %f %f %d %f %f %d %d',
'headerlines', 1);
```

Then we use function "**background_normalization()**" to calculate a scaling factor as follows:

```
%input data normalization by using background signals
function [WG_Region, scaling_factor] = background_normalization(WG_Region)

scaling_factor = 1.6029
```

Step 5:

In this step, we load gene promoter regions from file "**hg19_RefSeq.txt"** as follows:

```
%load reference genome from hg19
[transcripts.chr, transcripts.strand, transcripts.point1, transcripts.point2,
transcripts.gene_symbol]=textread('hg19_RefSeq.txt', '%s %s %d %d %s',
'headerlines', 1);
```

```
chrom       strand      txStart     txEnd       name2
chr1        -           33772366    33786699    A3GALT2
chr1        +           12776117    12788726    AADACL3
chr1        +           12776117    12788726    AADACL3
chr1        +           12704565    12727097    AADACL4
chr1        -           94458393    94586705    ABCA4
chr1        -           229652328   229694442   ABCB10
chr1        +           94883932    94984219    ABCD3
chr1        +           94883932    94944260    ABCD3
…
```

The gene annotation file downloaded from UCSC Genome Browser (https://genome.ucsc.edu/) is represented at transcript level. One gene may have multiple transcripts with the same or different 5' starting locations. We use another function "**genome_partation_region_mapping()**" to extract transcription starting site (TSS) of each gene (the minimum value of left points for gene at '+' strand or the maximum value of right points for gene at '-' strand). Then, using this function we partition gene promoter regions into 200 bps long windows, map candidate regions to windows and for each window calculate read intensities in sample and input ChIP-seq data, respectively.

```
%Gene promoter region extraction and window partition
[partition_promoters, WG_Region]=genome_partation_region_mapping(WG_Region,
transcripts);
```

We extract each window with a higher read intensity in the sample ChIP-seq data than that in the input ChIP-seq data as follows:

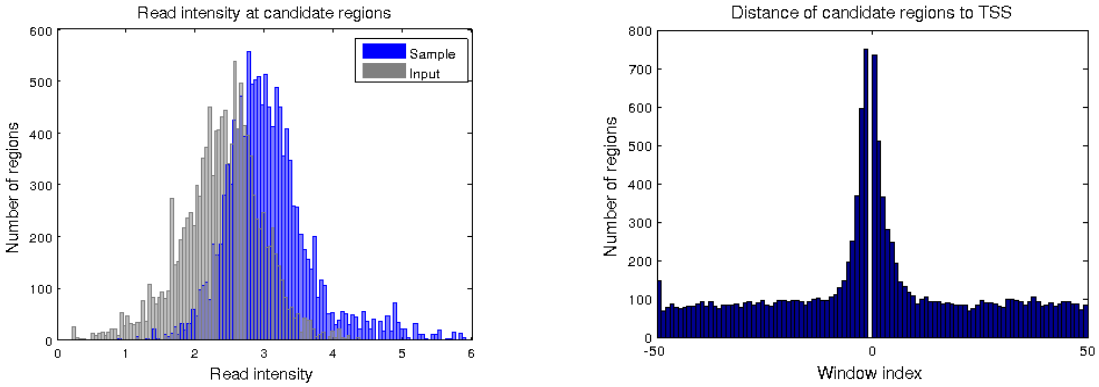
```
%*****************background region filtering**************
Candidate_window_index=find(partition_promoters.sample_intensity>0 &
partition_promoters.sample_intensity>partition_promoters.input_intensity);
Number_candidate_regions=length(Candidate_window_index);

Number_candidate_regions = 12486
```

In variable "partition_promoters", candidate window info is stored as follows:

```
chr    point1      point2      sample_intensity  input_intensity   direction
window_index       gene_symbol
'1'   33776700    33776899    2.345       1.862 '-'   50    'A3GALT2'
'1'   33776900    33777099    2.345       1.862 '-'   49    'A3GALT2'
'1'   33779900    33780099    2.439       1.407 '-'   34    'A3GALT2'
'1'   33780900    33781099    2.469       1.688 '-'   29    'A3GALT2'
'1'   33781100    33781299    2.469       1.688 '-'   28    'A3GALT2'
'1'   33786900    33787099    2.615       2.260 '-'   -2    'A3GALT2'
'1'   12772918    12773117    2.099       0.779 '+'   -16   'AADACL3'
'1'   12773118    12773317    2.099       0.779 '+'   -15   'AADACL3'
'1'   12705766    12705965    2.405       1.672 '+'   7     'AADACL4'
'1'   12705966    12706165    2.405       1.672 '+'   8     'AADACL4'
'1'   12706166    12706365    2.405       1.672 '+'   9     'AADACL4'
'1'   12710166    12710365    3.266       2.757 '+'   29    'AADACL4'
```

```
'1'    94580706    94580905    2.762        2.541 '-'    30     'ABCA4'
'1'    94580906    94581105    2.762        2.541 '-'    29     'ABCA4'
'1'    229694243   229694442   3.140        2.156 '-'    1      'ABCB10'
'1'    229694443   229694642   3.140        2.156 '-'    -1     'ABCB10'
'1'    229694643   229694842   3.140        2.156 '-'    -2     'ABCB10'
'1'    229694843   229695042   3.140        2.156 '-'    -3     'ABCB10'
'1'    229695043   229695242   3.140        2.156 '-'    -4     'ABCB10'
…
```

The histogram of "sample_intensity" and "input_intensity" for all candidate windows is shown in Fig. 2(a), while the histogram of "window_index" (denoting relative distance to TSS) for all candidate windows is shown in Fig. 2(b).



(a) Histogram of read intensity (b) Histogram of window index

Fig. 2. Histograms of read intensity and relative distance to TSS of candidate windows.

3. ChIP-BIT core function

The core function for peak detection is achieved by function "**ChIP_BIT()**". For each candidate window, we estimate a posterior probability as follows (note that an example of the histogram of posterior probabilities for all candidate windows is shown in Fig. 3):

```
%*****************ChIP-BIT model***************************
L = 100;%Number of rounds of EM estimation
fprintf('EM starts!\n')
[Candidate_window_index, a_1, a_0, pi_1, pi_0, mean_TFBS, variance_TFBS,
variance_Background, lambda_TFBS]=ChIP_BIT(partition_promoters,
Candidate_window_index, L);
partition_promoters.a_1=zeros(length(partition_promoters.chr), 1);
partition_promoters.a_1(Candidate_window_index)=a_1;

pi_1 = 0.6004
pi_0 = 0.3996
mean_TFBS = 3.1324
variance_TFBS = 0.4125
variance_Background = 0.3448
lambda_TFBS = 0.8013
```

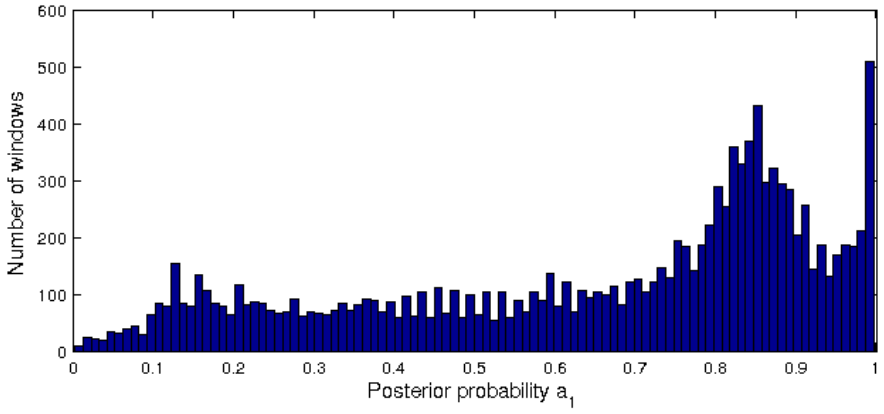
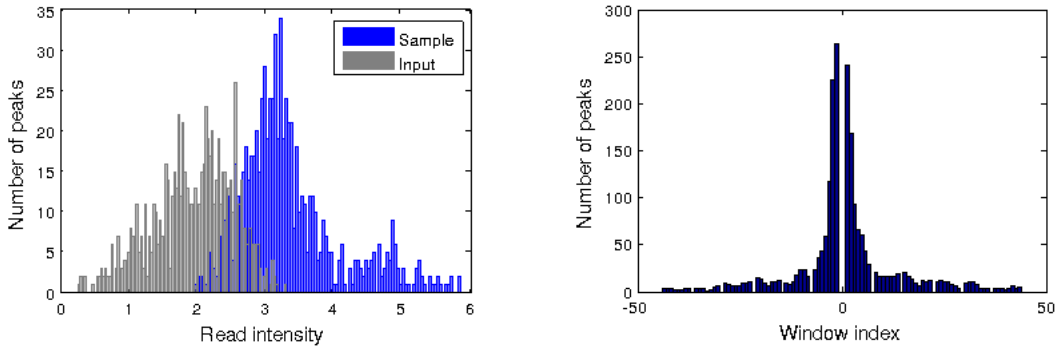


Fig. 3. Histogram of posterior probabilities for all candidate windows.

Windows with posterior probabilities over 0.95 are selected and consecutive windows are further merged into larger peaks using function "**window2peak()**". (An example of the distributions of read intensity and relative distance to TSS of identified peaks is shown in Fig. 4.)

```
peak_threshold = 0.95;%probability threshold for peak detection
%***************merge windows into peaks********************
ChIP_BIT_peaks = window2peak(partition_promoters, peak_threshold);
```



(a) Read intensity of detected peaks (b) Relative distance of detected peaks

Fig. 4. Distributions of read intensity and relative distance to TSS of identified peaks.

Finally, a peak file, "**ChIP_BIT_Chr1_peaks.txt**", is created to store all detected peaks detected.

```
chromosome  start_point end_point   gene_symbol Sample_Reads_intensity
      Input_Reads_intensity   Posterior_probability
1     229694243   229695242   ABCB10      3.140179    2.156729    0.947231
1     1243270     1243869     ACAP3       3.216313    2.193865    0.950682
1     226374224   226375023   ACBD3       3.266675    2.092388    0.967518
1     159169803   159170002   ACKR1       3.133405    1.659656    0.974730
1     55012807    55013806    ACOT11      2.934070    2.106528    0.967670
1     6457027     6457426     ACOT7       3.082827    1.762697    0.935282
```

```
1     120434948   120435347   ADAM30      3.061520    1.134501    0.983762
1     167883865   167884064   ADCY10      3.293018    1.819366    0.992985
1     202928301   202928500   ADIPOR1     3.521880    2.694488    0.905599
1     244616637   244623836   ADSS        3.488231    1.762697    0.954942
1     50488427    50488626    AGBL4       2.953868    1.479771    0.991095
1     15911006    15911405    AGMAT       2.763043    1.577070    0.975546
…
```

As an extension, if ChIP-seq profiles of two TFs are available, a MATLAB script "**ChIP_BIT_two_TFs_main.m**" will be optionally used to predict co-regulated target genes using function "**coregulated_gene_prediction()**".

```
coregulated_genes=coregulated_gene_predction(TF1_partition_promoters,
TF2_partition_promoters);
```

III. ChIP-seq data analysis using R version of ChIP-BIT

If the R script of ChIP-BIT is used, sample and input ChIP-seq data sets need to be separately preprocessed using the stand along binary file "**PeakSeq_first_pass_intensity_output**" as follows:

```
$ mkdir Sample_chr1_reads
$ mkdir Input_chr1_reads

$ samtools view PBX1_chr1.bam | ./PeakSeq_first_pass_intensity_output -
preprocess SAM stdin Sample_chr1_reads
$ samtools view Input_chr1.bam | ./PeakSeq_first_pass_intensity_output -
preprocess SAM stdin Input_chr1_reads

$ ./PeakSeq_first_pass_intensity_output -peak_select config.dat
```

Then, a file "**Candidate_regions_chr1.txt**" including all candidate regions will be created. We put "**Candidate_regions.txt**" and gene annotation file "**hg19_RefSeq.txt**" under the same folder with the R script of ChIP-BIT and use file "**ChIP_BIT_signle_TF_demo.R**" to perform the remaining Steps 4 and 5 as well as the final peak calling.

In our demo case, we have already generated "**Candidate_regions_chr1.txt**" from chromosome 1 of PBX1 and its input ChIP-seq data for users to test the main functions of ChIP-BIT.

In this R script, names and formats of input and output variables of major functions are consistent with the MATLAB functions descried previously. We have tested both R scripts (R 3.2.2 64 bit) and MATLAB package (MATLAB 2012b 64 bit) under Ubuntu 12.04. Their results are highly consistent, as shown in Table 1.

Table 1. ChIP-BIT results comparison of R version and Matlab version.

	R version	Matlab version
Number of candidate regions	24331	24331
Number of candidate windows	12448	12454
Number of significant windows (>0.95)	1191	1184
Number of significant windows (>0.9)	2090	2090
Number of significant windows (>0.85)	3631	3623
Number of significant peaks using windows >0.9	735	735