Using `tileHMM` for ChIP–on–Chip Analysis

Peter Humburg, David Bulger, and Glenn Stone

August 18, 2008

1 Introduction

This document provides an introduction to the `tileHMM` package. A relatively small simulated dataset is used to illustrate the main concepts and work flow of the package; starting with the raw data the necessary steps for pre-processing and analysis are explained. The final output of the analysis presented here is a file in GFF format summarising ChIP enriched regions. While the `simChIP` data provided with the `tileHMM` package is much smaller than a typical tiling array dataset some of the examples below may take some time to execute, especially on older machines.

1.1 Requirements

Some additional R packages are required to run the examples in this document.

- The packages `affy` and `st` are required for pre-processing.
- The package `geneplotter` is used for some of the plotting.

Both `affy` and `geneplotter` are part of the bioconductor project and can be obtained from `http://www.bioconductor.org/`, `st` is available from CRAN (`http://cran.r-project.org/`).

1.2 Data

```
> library(tileHMM)
> data(simChIP)
```

The dataset `simChIP` is used throughout this document. It is part of the data used for a simulation study in [3] and is based on data published by [6]. The data consists of measurements for 210828 probes under three different treatments with four replicates each.

Input Genomic DNA.

H3 ChIP sample using an antibody against histone H3.

H3K27me3 ChIP sample using an antibody against lysine 27 trimethylation at histone H3.

2 Identifying ChIP Regions

In this section we will carry out an analysis to identify H3K27me3 enriched regions. For this analysis we use the H3 sample as control for the H3K27me3 treatment. We will ignore the genomic input for now.

```
> h3k27.data <- as.matrix(simChIP[, 7:14])
```

2.1 Pre-processing

We start by log transforming and quantile normalising the data.

```
> library(affy)
> h3k27.data <- log(h3k27.data, 2)
> h3k27.norm <- normalize.quantiles(h3k27.data)
```

Now we can look at some diagnostic plots. Here we restrict ourselves to simple scatter plots (Figure 1), of course real data should be inspected more carefully before any analysis is carried out. The scatter plots do not reveal any major problems although it should be noted that there appear to be some differences between samples. This is most apparent when comparing the scatter plots for samples 2 and 4.

The next step is to summarise probe measurements into probe statistics. For this we use the "shrinkage t" statistic [5]. The `tileHMM` package provides the function `shrinkt.st` for this purpose.

```
> h3k27.stat <- shrinkt.st(h3k27.norm, c(rep(2, 4), rep(1, 4)),
+     verbose = 0)
```

This provides us with a single value for each probe. Before we move on to build a hidden Markov model to analyse these data we take a look at the distribution of the probe statistics (Figure 2).

2.2 Initial Parameter Estimates

To identify ChIP enriched regions in the data we use a two state hidden Markov model with t distributions to model observations. Initial parameter estimates can be obtained from the data using `hmm.setup`. This requires information about the average probe density and fragment size. Here we will assume that the distance between probe centres is 35bp and that ChIP

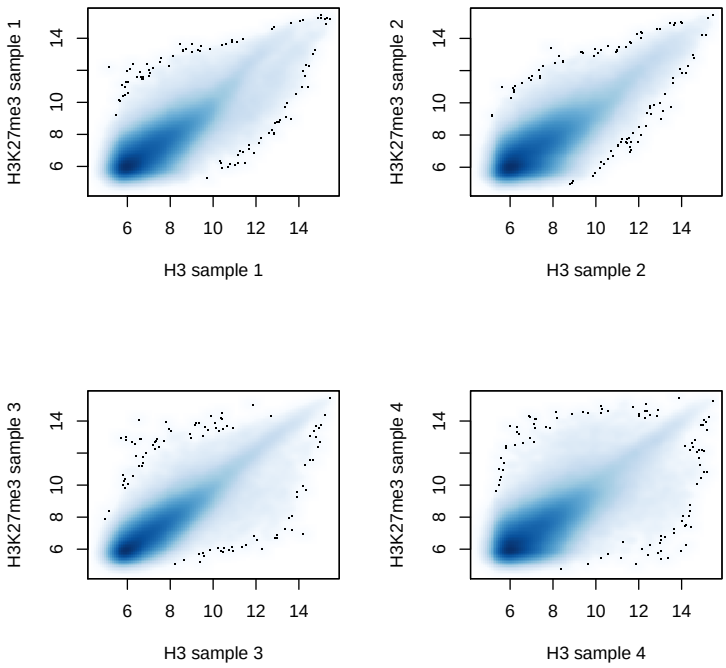

Figure 1: Pairwise scatter plots of the quantile normalised data

fragments are about 1kb long. The function `hmm.setup` accepts arguments `probe.region` and `frag.size` to set these parameters, we will use the defaults. A good choice for these parameters is useful but not crucial as they will be optimised later. We also provide names to identify the two states of our model. For the estimation of transition probabilities it is necessary to indicate which state should represent ChIP regions. Since `hmm.setup` automatically assigns the distribution with the smaller location parameter to the first state the second state will correspond to enriched regions.

```
> hmm.init <- hmm.setup(h3k27.stat,
+     state = c("non-enriched","enriched"), pos.state = 2)
```

We inspect the resulting model to ensure that the parameter estimates are reasonable. We can also plot the density functions of the two states (Figure 3).

```
> print(hmm.init)
```

```
An object of class "contHMM"
Slot "transition.matrix":
```

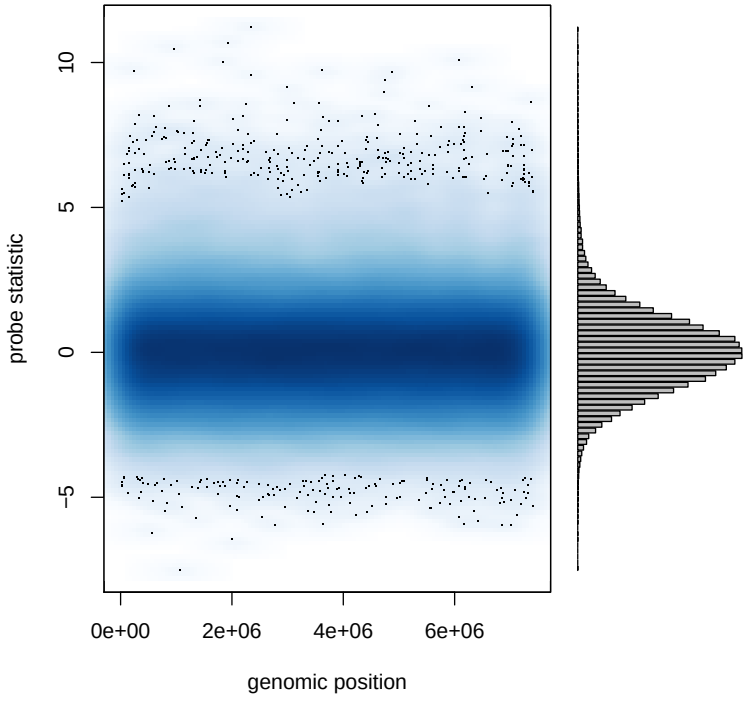

Figure 2: Distribution of probe statistic along the 'chromosome' with marginal histogram. Larger probe statistics indicate probes that are likely to be inside a ChIP-region.

```
             non-enriched   enriched
non-enriched     0.981676 0.01832392
enriched         0.035000 0.96500000

Slot "emission":
$`non-enriched`
An object of class "tDist"
Slot "components":
         weight       mean  variance df
mean.pos      1 -0.9716776 0.6438807  9


$enriched
An object of class "tDist"
Slot "components":
         weight     mean  variance df
mean.neg      1 1.144570 0.9099603  9
```

```
Slot "init":
[1] 0.6563659 0.3436341

> plot(hmm.init)
```

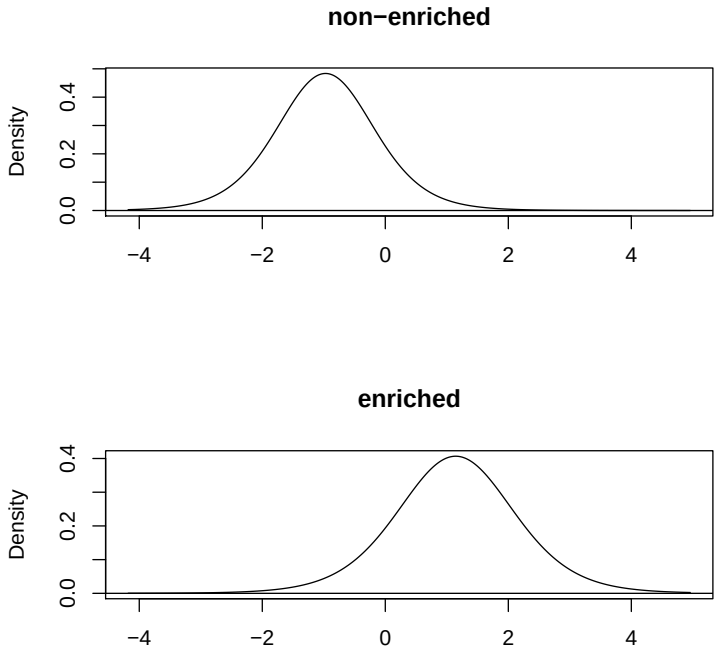

Figure 3: Initial estimate for emission distributions.

2.3 Parameter Optimisation

At this stage it is necessary to consider another aspect of tiling array data. An underlying assumption of our model is that all probes are spaced out equally along the genome. While this is approximately true for the majority of the tiled genome sequence, there are some large gaps between probes. It is questionable whether the relationship between neighboring probes that is implied by the model holds in these cases. To ensure that these gaps are not distorting the result we will split our sequence of probe statistics into sub-sequences whenever the gap between two probes is larger than `max.gap`.

These simulated data were generated such that it contains several gaps of 1kb between probes.

```
> max.gap <- 999
> h3k27.gap <- diff(simChIP[["position"]]) > max.gap
> gap.idx <- which(h3k27.gap)
```

Now `gap.idx` contains the index of the last probe in each of the subsequences except the last. With this information it is straightforward to extract subsequences that are guaranteed not to contain any gap larger than `max.gap`.

```
> start <- c(1, gap.idx + 1)
> end <- c(gap.idx, length(h3k27.stat))
> h3k27.lst <- mapply(function(s, e, data) data[s:e], start, end,
+     MoreArgs = list(h3k27.stat))
```

We will now use a maximum likelihood approach to optimise the parameters of our model. In a first step the initial parameter estimates are improved by Viterbi training [4] and are then further optimised with the EM algorithm [2, 1]. The package `tileHMM` provides the function `viterbiEM` for this purpose.

```
> hmm.opt <- viterbiEM(hmm.init, h3k27.lst, df = 9)
```

This uses five iterations of Viterbi training to improve initial parameter estimates. The new estimates are then used as starting point for 15 iterations of the EM algorithm. The number of iterations for each of the two algorithms can be adjusted via the `max.iter` argument of `viterbiEM` but 5 and 15 should be sufficient in most cases. Here we are restricting the degrees of freedom for the t distributions of both states to 9 using the `df` argument. It is possible to estimate the required degrees of freedom from the data but this is time consuming, may not give good results and should generally be avoided. Using the argument `verbose` the amount of status messages produced during the model fitting procedure can be controlled. Above we used the default setting `verbose = 0`. If you want more feedback try a higher level of verbosity. For example using `verbose = 2` would produce something like

```
Viterbi training: 5 iterations

Number of iterations: 5
Log likelihood of best model: -356120.327537388
Last change in log likelihood: -15.7415983500541

EM algorithm: 15 iterations
```

```
Number of iterations: 7
Log likelihood of final model: -354114.807307941
Last change in log likelihood: 0.00472418434219435
```

Note that this indicates that the EM algorithm converged after only 7 iterations. Convergence is determined by the change in log likelihood between iterations. The threshold below which changes are considered insignificant can be set via the argument `eps`. The default we used for this example is 0.01.

Again we can inspect the parameters of our model. This time they should be more meaningful. Compare the parameter estimates to the initial ones above. Figure 4 is a plot of the newly estimated emission distributions.

```
> print(hmm.opt)

An object of class "contHMM"
Slot "transition.matrix":
             non-enriched    enriched
non-enriched   0.99841633 0.001583671
enriched       0.03694841 0.963051591

Slot "emission":
$`non-enriched`
An object of class "tDist"
Slot "components":
     weight        mean variance df
[1,]      1 -0.06256513 1.299466  9


$enriched
An object of class "tDist"
Slot "components":
     weight     mean variance df
[1,]      1 2.229261 2.053111  9



Slot "init":
[1] 0.95449261 0.04550739

> plot(hmm.opt)
```

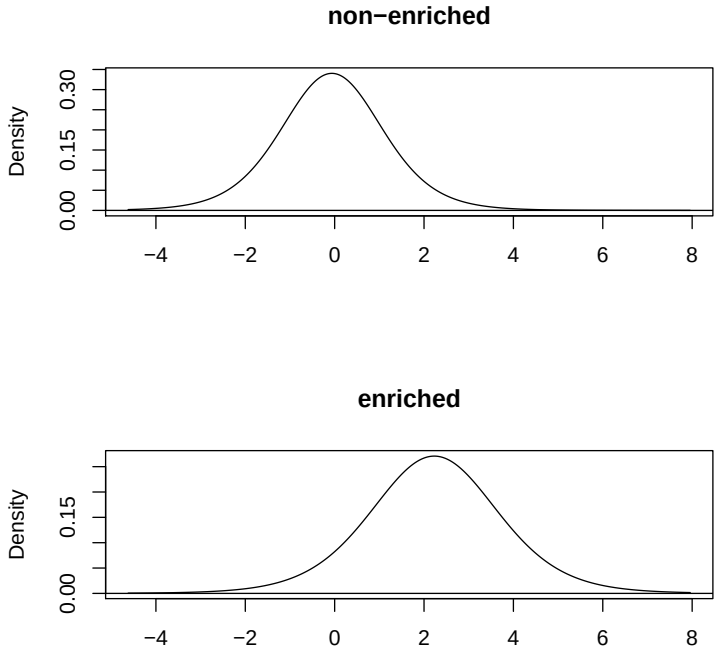

Figure 4: Emission distributions of optimised HMM.

2.4 Calling Probes

Now that we have fitted our model we are ready to identify enriched probes. One way of doing this is to determine the most likely state for each individual probe. This can be achieved with the function `posterior` which calculates the posterior probability for each state.

```
> post <- lapply(h3k27.lst, posterior, hmm.opt)
> state.seq <- lapply(post, apply, 2, which.max)
> state.seq <- states(hmm.opt)[c(state.seq, recursive = TRUE)]
```

This provides us with a single sequence of states to explain the observed probe statistics. Before we move on to combining these individual probe calls into enriched and non-enriched regions we can use the probe level information to produce a slightly different plot of the result.

```
> ratio <- sum(state.seq == "enriched")/length(state.seq)
> hist(h3k27.stat, breaks = 100, probability = TRUE, main = "",
+     xlab = "probe statistic")
> plot(hmm.opt@emission$enriched, new.plot = FALSE, lty = 1, lwd = 2,
```

```
+     col = 4, weight = ratio)
> plot(hmm.opt@emission$"non-enriched", new.plot = FALSE, lty = 2,
+     lwd = 2, col = 2, weight = 1 - ratio)
> legend("topright", legend = c("enriched", "non-enriched"), lty = 1:2,
+     lwd = 2, col = c(4, 2))
```

This produces the plot in Figure 5. It shows a histogram of the probe statistics with superimposed density functions of emission distributions. The density functions are scaled according to the relative frequency of probe calls from the corresponding states.

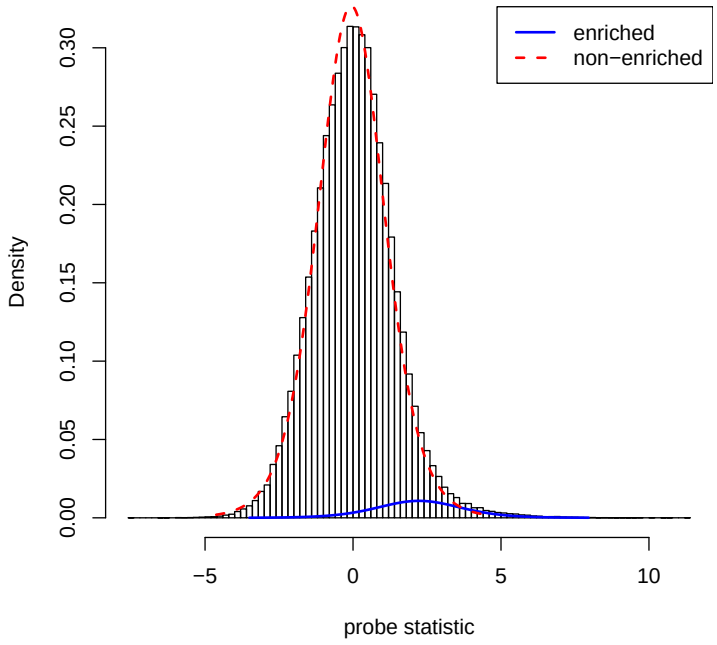

Figure 5: Probe statistics and emission distributions of optimised HMM. The density functions for both states are scaled according to the relative number of probes assigned to the corresponding state.

2.5 Calling Regions

We are now ready to combine individual probe calls into regions. The identified ChIP enriched regions are then summarised in a GFF file. The first step is to identify continuous runs of probes that were called 'enriched'. The function `region.position` provides an easy way to do just that.

```
> regions.idx <- region.position(state.seq, region = "enriched")
```

Now `regions.idx` is a matrix with the index of the first probe in each enriched region in the first row and the index of the last probe in the second row. The number of columns in this matrix tells us how many enriched regions our model identified. In this case there are 275.

The next step is to convert probe indices into genomic coordinates. Using the information about probe positions provided with the data we can do this easily.

```
> regions.pos <- matrix(simChIP[regions.idx, 2],
+     nrow = 2, ncol = dim(regions.idx)[2])
```

At this point there are a few things about the identified ChIP regions that we should consider. It is important to note that we used the complete sequence of probe calls to identify the regions of interest rather than the set of subsequences that we created earlier. This is fine in some cases but it means that enriched regions from neighbouring subsequences may be joined into one region. This may not be desired in which case regions have to be called for each subsequence. This can be done in the same way as above.

Once regions of interest are identified there are several possible post-processing steps that should be considered. We could join neighbouring enriched regions that are close together to avoid the fragmentation of enriched regions caused by some analysis methods. The model used for the analysis here is very sensitive and is not expected to fragment enriched regions into many smaller ones [3]. Thus, we are not recommending this type of post-processing. It is, however, possible that the analysis produced some small spurious regions. We can try to identify and remove these regions by looking for short regions with low posterior probability of enrichment. Here we will use the mean probability of all probes in the enriched region as a score for that region.

```
> post.enriched <- lapply(post,"[",2,)
> post.enriched <- exp(c(post.enriched, recursive = TRUE))
> region.score <- apply(regions.idx, 2,
+     function(reg, post) mean(post[reg[1]:reg[2]]), post.enriched)
```

We also calculate the length of all enriched regions identified by our analysis.

```
> region.len <- apply(regions.pos, 2, diff)
```

A plot of this information is presented in Figure 6. Regions shorter than 400bp with an average posterior probability of less than 0.8 are marked with a red '×' , these are low confidence regions that we may want to remove. The function `remove.short` can be used to remove these suspect regions.

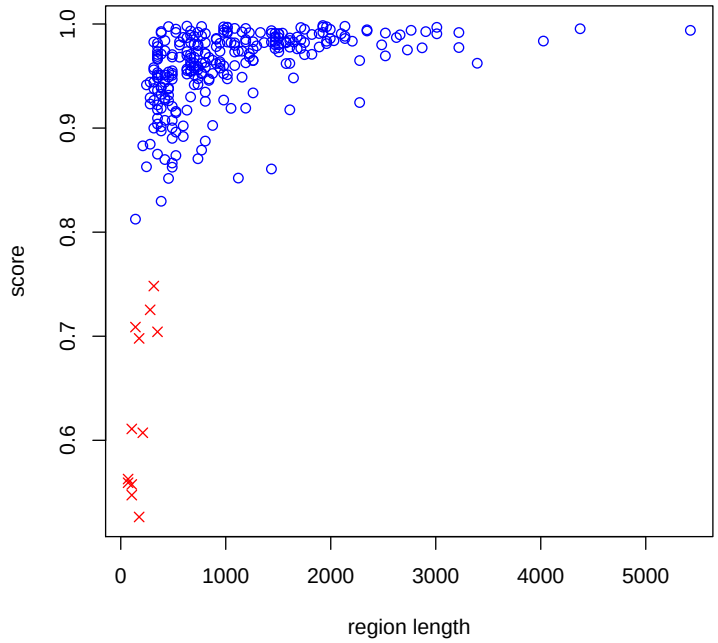

Figure 6: Length and score of enriched regions. Regions shorter than 400bp with a score of less than 0.8 are indicated by red ×s. These are low confidence regions that may have to be removed.

```
> regions.clean <- remove.short(regions.idx, post.enriched,
+     simChIP[ , 1:2], min.length = 400, min.score = 0.8)
```

Note that `remove.short` expects regions to be identified in terms of probe indices, not genomic coordinates (which is why we used `regions.idx` instead of `regions.pos`). Some care should be taken in choosing the parameters for this post-processing step to avoid excluding large numbers of regions that really are enriched but are for some reason difficult to detect, which may result in short regions with low scores. Where possible, information about the size of ChIP fragments should be used to inform parameter choices.

Now that we have identified ChIP enriched regions we can save the information about these regions to a GFF formatted file.

```
> gff <- reg2gff(regions.clean, post.enriched, simChIP[, 1:2],
+     file = "simRegions.gff")
```

This creates a GFF file called '`simRegions.gff`' with an entry for each enriched region. The score for each region is again calculated by summarising

the posterior probabilities of all probes inside the region. The function used to calculate this summary is determined by the `score.fun` argument which defaults to `mean`. Other functions like `median` and `max` may be useful as well. It may be desirable to employ a more complex scoring function. Any function that accepts a single numeric argument and returns a scalar value will work, although the usefulness of the resulting score can vary substantially. The contents of the fields containing meta information can be set via the arguments of `reg2gff`, see the help page for more details on this.

The GFF file can then be used for further down stream analysis and visualisation, e.g., in a genome browser.

References

[1] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occuring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, **41**(1):164-171, 1970.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood for incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, **39**(1), 1977.

[3] P. Humburg, D. Bulger, and G. Stone. Parameter estimation for robust HMM analysis of ChIP-chip data. *BMC Bioinformatics*, **9**:343, 2008.

[4] B-H. Juang and L. R. Rabiner. A segmental k-means algorithm for estimating parameters of hidden Markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **38**(9):1639-1641, 1990.

[5] R. Opgen-Rhein and K. Strimmer. Accurate ranking of differentially expressed genes by a distribution-free shrinkage approach. *Statistical applications in Genetics and Molecular Biology*, **6**(1):Article 9, 2007.

[6] X. Zhang, O. Clarenz, S. Cokus, Y. V. Bernatavichute, J. Goodrich, and S. E. Jacobsen. Whole-genome analysis of histone H3 lysine 27 trimethylation in Arabidopsis. *PLoS Biology*, **5**(5), May 2007.