BUM-HMM computational pipeline

Alina Selega

October 23, 2016

This vignette provides an example workflow for running the Beta-Uniform Mixture hidden Markov model (BUM-HMM) computational pipeline [1]. BUM-HMM provides a statistical framework for computing per-nucleotide posterior probabilities of modification from the reactivity scores obtained in an RNA structure probing experiment such as SHAPE [2] or ChemModSeq [3].

1 Data format

The pipeline requires three data sets: the coverage, the number of times that the reverse transcriptase dropped off, and the rate of this drop-off for each nucleotide position of the RNA molecule of interest. (The reverse transcriptase drop-off reflects the modification of a site by the chemical probe.) The coverage and the drop-off counts are the data obtained in an experiment and the drop-off rate r at each nucleotide is computed as the ratio between the drop-off count k and the coverage n:

$$r = \frac{k}{n}$$

All data sets are expected to be in the matrix format, with rows corresponding to nucleotide positions and columns corresponding to experimental replicates (all control replicates followed by all treatment replicates). The key strength of BUM-HMM is accounting for the variability of the data and thus it requires data sets available in multiple replicates. The three matrices for coverage, drop-off counts, and drop-off rates should only hold numerical values and no information about the chromosome or the nucleotide position. For transcriptome-wide experiments, the data over different chromosomes should be concatenated row-wise in a single matrix.

We provide a data set obtained in a structure probing experiment on 18S ribosomal RNA with the DMS chemical probing agent, available in triplicates.

This data set has three matrices `covFile`, `docFile`, and `dorFile`, representing the coverage, drop-off counts, and drop-off rates for each of the 1800 nucleotides of the 18S molecule. The data set has 3 control and 3 treatment experimental replicates, which are arranged column-wise with control replicates first (columns 1-3) and treatment replicates following (columns 4-6). The data set also has a string 'sequence' with the corresponding genetic sequence, required for correcting the sequence-dependent bias.

```
library(BUMHMM)
pos = 300
paste0("coverage = ", covFile[pos, 1], "; drop-off count = ", docFile[pos, 1],
       "; drop-off rate = ", signif(dorFile[pos, 1], 3), "; base = ",
       substr(sequence, pos, pos))

## [1] "coverage = 813073; drop-off count = 1837; drop-off rate = 0.00226; base = A"
```

Thus, we see that the 300th nucleotide A had coverage of 813073 in the first control experimental replicate, of which the reverse transcriptase stopped at that nucleotide 1837 times, giving it a drop-off rate of 0.00226.

```
paste0("coverage = ", covFile[pos, 4], "; drop-off count = ", docFile[pos, 4],
       "; drop-off rate = ", signif(dorFile[pos, 4], 3), "; base = ",
       substr(sequence, pos, pos))

## [1] "coverage = 1640501; drop-off count = 4844; drop-off rate = 0.00295; base = A"
```

In the first treatment experimental replicate, that is, in the presence of a chemical probe, the coverage and drop-off count at that position were higher but the drop-off rate remained similar, 0.00295.

2 The overview of pipeline

The logic of structure probing experiments associates the accessibility of a nucleotide with structural flexibility, i.e. double-stranded nucleotides or those otherwise protected (e.g. by a protein interaction) will not be available for an interaction with the chemical reagent. In contrast, those nucleotides that are located in flexible regions, could be chemically modified by the reagent and will therefore be at the positions at which the reverse transcriptase drops off. Thus, these nucleotides are expected to have a high drop-off rate.

To distinguish the true positives from noisy observations, BUM-HMM compares the drop-off rates at each nucleotide position between the control experimental replicates:

$$log\left(\frac{r_{C_i}}{r_{C_j}}\right)$$

If the drop-off rates r_{C_i} and r_{C_j} are similar in a pair of replicates (C_i and C_j), the above log-ratio will be close to 0, indicating little variability. In contrast, different drop-off rates would result in a large log-ratio (in absolute value). Computing these per-nucleotide log-ratios for all pairs of control experimental replicates defines a *null distribution* which quantifies the amount of noise observed in control conditions. Anything within that range could be simple noise. (Note that due to a log transform, drop-off rates $r = 0$ are not allowed.)

We now compare the drop-off rates between all pairs of control and treatment experimental replicates:

$$log\left(\frac{r_{T_i}}{r_{C_j}}\right)$$

The assumption is that a flexible nucleotide will have a much larger drop-off rate in a treatment experiment (T_i) compared to control conditions (C_j), and thus generate a large log-ratio for this pair of replicates. By comparing treatment-contol log-ratios to the null distribution in control conditions, we can find those nucleotides that demonstrate differences in drop-off rate larger than those that can be expected by chance.

3 Selecting pairs of nucleotides

The first step of the pipeline selects pairs of nucleotide positions for computing log-ratios and the positions for which the posterior probabilities of modification will be computed. This is implemented with the function `selectNuclPos`. The function takes the `covFile` and `docFile` matrices, the numbers of control and treatment experimental replicates in the data set (`Nc` and `Nt`, correspondingly), and a user-specified coverage threshold `t`. Nucleotides with coverage $n < t$ will not be considered. Posterior probabilities are computed for nucleotides with minimum allowed coverage in all experimental replicates and a non-zero drop-off count in at least one treatment replicate.

In our data set, we have 3 control and 3 treatment replicates, so if we set the minimum allowable coverage to 1, we can make the following function call:

```
Nc = 3
Nt = 3
t = 1
nuclSelection <- selectNuclPos(covFile, docFile, Nc, Nt, t)
```

The function `selectNuclPos` returns a list with three elements:

- `analysedC` is a list where each element corresponds to a control-control replicate comparison. Each element holds indices of nucleotides that have coverage $n >= 1$ and a drop-off count $k > 0$ in both replicates of that comparison.
- `analysedCT` is a list where each element corresponds to a treatment-control replicate comparison. Again, each element holds indices of nucleotides that have coverage $n >= 1$ and a drop-off count $k > 0$ in both replicates of that comparison.
- `computePosteriors` is a list where each element corresponds to a treatment replicate. Each element holds indices of nucleotides that have coverage $n >= 1$ in all replicates and a drop-off count $k > 0$ in that experimental replicate.

```
length(nuclSelection$analysedC[[1]])

## [1] 1713

length(nuclSelection$analysedCT[[1]])

## [1] 1723
```

In this case, we select 1713 nucleotides for the first control-control comparison and 1723 for the first treatment-control comparison.

4 Scaling the drop-off rates across replicates

Because BUM-HMM works with data collected in multiple experimental replicates, it is important to ensure that the drop-off rates do not differ dramatically between different replicates. Thus, the second step of the pipeline scales the drop-off rates of all nucleotides selected for pair-wise comparisons (in other words, those nucleotides that will be considered in the analysis) to have a common median value. This is implemented with a function `scaleDOR`, which requires the output of the function `selectNuclPos` (described above) and returns an updated matrix of drop-off rates `dorFile` where the drop-off rates of each replicate (i.e. values in each column) are scaled so that the selected drop-off rates have the same median in all columns:

```
## Medians of original drop-off rates in each replicate
apply(dorFile, 2, median)

## [1] 0.001005071 0.001025182 0.003684315 0.001608314 0.001302076 0.003380141

dorFile <- scaleDOR(dorFile, nuclSelection, Nc, Nt)
## Medians of scaled drop-off rates in each replicate
apply(dorFile, 2, median)

## [1] 0.001618859 0.001611737 0.001707502 0.001641554 0.001708537 0.001719018
```

After scaling, medians are much more similar across replicates (they are not exactly equal when computed this way as not all nucleotides were selected for different pair-wise comparisons.)

5 Computing stretches of nucleotide positions

The next step in the BUM-HMM modelling approach enforces a smoothness assumption over the state of nucleotides: chemical modification does not randomly switch along the chromosome, rather, continuous stretches of RNA are either flexible or not. This is captured with a hidden Markov model (HMM) with binary latent states, corresponding to the true state of each nucleotide: modified or unmodified.

The observations of the HMM are the p-values associated with each nucleotide arising from treatment-control comparisons. Modelling p-values directly and having prior expectations about the log-ratios arising from the comparisons involving modified nucleotides enabled us to model the emission distribution with a Beta-Uniform mixture model. Further details can be found in [1].

To run HMM, we compute uninterrupted stretches of nucleotides for which the posterior probabilities are to be computed. This is achieved with the function `computeStretches`, which requires a sorted list of all nucleotide positions for which posterior probabilities are required. This list can be obtained from the third element of `nuclSelection`, the output of the `selectNuclPos` function. This element holds indices of such nucleotides selected from each treatment replicate; a union of those gives all nucleotides for which the model can compute the probability of modification.

```
allNucleotides <- Reduce(union, nuclSelection$computePosteriors)
stretches <- computeStretches(sort(allNucleotides))
```

The function returns a list where each element corresponds to an uninterrupted stretch; its first element indicates the start index and the second element - the end index of the stretch. HMM will be run separately on each stretch.

```
head(stretches)

## [[1]]
## [1]    1 1747
##
## [[2]]
## [1] 1749 1800
```

We will compute posterior probabilities for all nucleotides but one, which is at the 1748th position.

6 Bias correction

Using a transcriptome-wide data set, we identified sequence and coverage as factors that influence log-ratios in control conditions, that is, in the absence of any reagent. We would therefore like to transform the log-ratios such that these biases are eliminated and the performed comparisons are not confounded.

6.1 Coverage bias

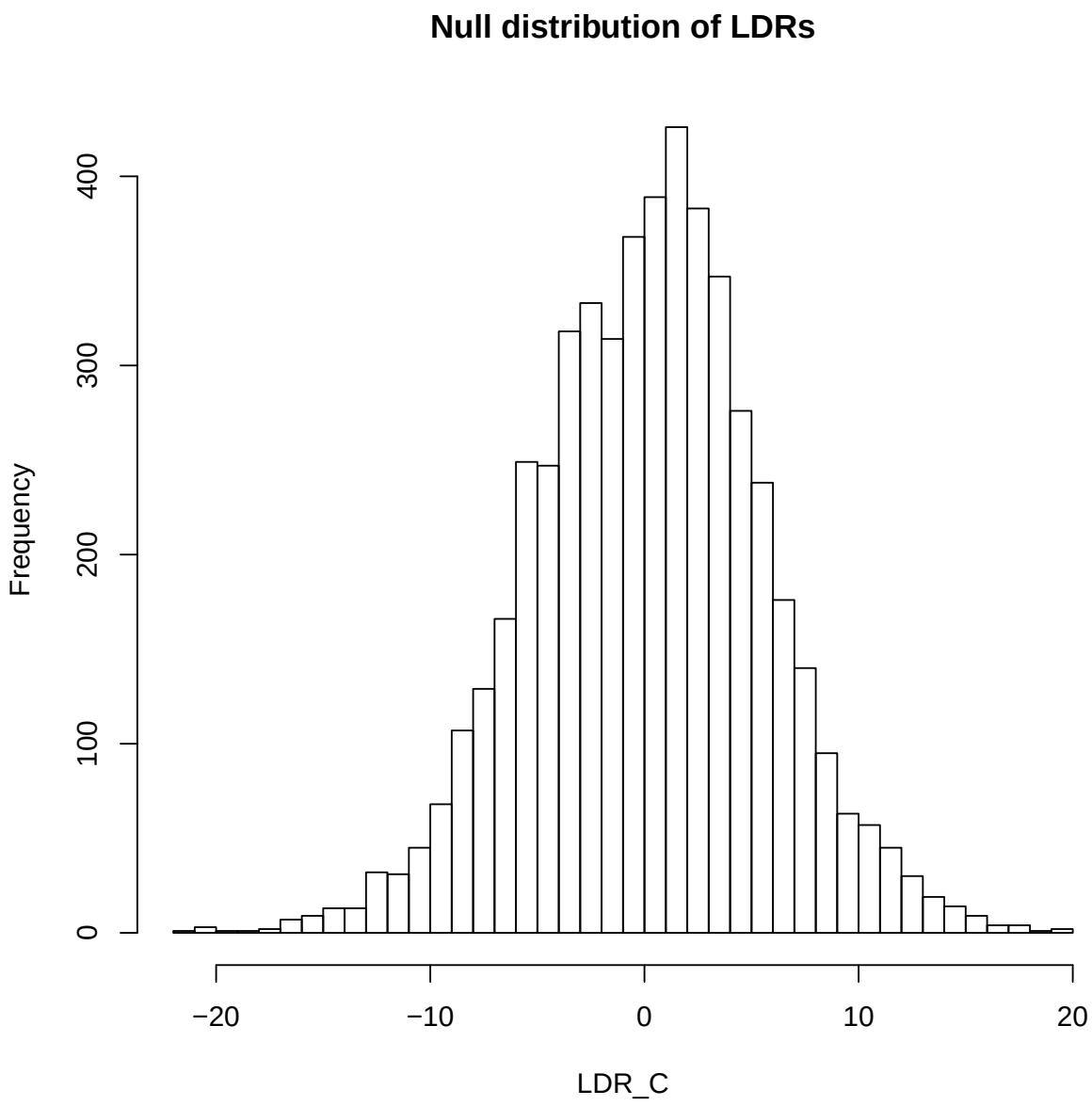
The coverage bias is addressed by a variance stabilisation strategy, implemented by the `stabiliseVariance` function. This function aims to find a functional relationship between the LDRs in the null distribution and the average coverage in the corresponding pair of control replicates. This relationship is modelled with the assumption that the drop-off count is a binomially distributed random variable (see [1] for details) and is fitted to the data with a non-linear least squares technique. For efficiency (i.e. running on transciptome-wide data sets), the mean coverage values and LDRs are split into bins and average values across each bin are fitted. Then, all LDRs (both for control-control and treatment-control comparisons) are transformed accordingly, so that the dependency on the coverage is eliminated.

The function requires the coverage and drop-off rate matrices, the positions of nucleotides selected for pairwise comparisons, and the numbers of replicates. The LDR null distribution is computed by using the values at the nucleotide positions selected for control-control comparisons. Then, the parameters are fitted as described above, the LDRs selected for all treatment-control comparisons are computed, and both

them and the null distribution are transformed accordingly. Finally, the function returns a list with two elements:

- `LDR_C` is a matrix with transformed log-ratios for control-control comparisons.
- `LDR_CT` is a matrix with transformed log-ratios for treatment-control comparisons. Both matrices have rows corresponding to nucleotides and columns to a pair-wise comparison.

```
varStab <- stabiliseVariance(covFile, dorFile, nuclSelection$analysedC,
                        nuclSelection$analysedCT, Nc, Nt)
LDR_C <- varStab$LDR_C
LDR_CT <- varStab$LDR_CT

hist(LDR_C, breaks = 30, main = 'Null distribution of LDRs')
```



6.2 Sequence bias

The sequence-dependent bias is addressed by computing different null distributions of LDRs for different patterns of nucleotides. One could consider trinucleotide patterns, reflecting an assumption that the previous and next neighbours of a nucleotide could affect its accessibility; patterns of other lengths cold also be considered. The function `nuclPerm` returns a vector of all permutations of four nucleobases (A, T, G, and C) of length n:

```
nuclNum <- 3
patterns <- nuclPerm(nuclNum)
patterns

##  [1] "AAA" "AAC" "AAG" "AAT" "ACA" "ACC" "ACG" "ACT" "AGA" "AGC" "AGG" "AGT" "ATA" "ATC"
## [15] "ATG" "ATT" "CAA" "CAC" "CAG" "CAT" "CCA" "CCC" "CCG" "CCT" "CGA" "CGC" "CGG" "CGT"
## [29] "CTA" "CTC" "CTG" "CTT" "GAA" "GAC" "GAG" "GAT" "GCA" "GCC" "GCG" "GCT" "GGA" "GGC"
## [43] "GGG" "GGT" "GTA" "GTC" "GTG" "GTT" "TAA" "TAC" "TAG" "TAT" "TCA" "TCC" "TCG" "TCT"
## [57] "TGA" "TGC" "TGG" "TGT" "TTA" "TTC" "TTG" "TTT"
```

Considering trinucleotide patterns will result in computing 64 different null distributions of LDRs, each corresponding to one pattern. To do this, we first need to find all occurrences of each pattern within the sequence. This is implemented with the function `findPatternPos`, which takes the list of patterns, a string containing the sequence, and a parameter indicating whether we are dealing with sense (+) or anti-sense (-) DNA strand.

```
nuclPosition <- findPatternPos(patterns, sequence, '+')
patterns[[1]]

## [1] "AAA"

head(nuclPosition[[1]])

##      start end
## [1,]    39  41
## [2,]    84  86
## [3,]   103 105
## [4,]   179 181
## [5,]   180 182
## [6,]   217 219
```

In the above example, pattern AAA appears on the positions indicated above. For each pattern, we would like to collect the LDRs at the middle nucleotide to compute a pattern-specific null distribution. These middle positions can be computed as follows:

```
nuclPosition <- lapply(nuclPosition, function(x)
                x[, 1] + (ceiling(nuclNum / 2) - 1))
head(nuclPosition[[1]])

## [1]  40  85 104 180 181 218
```

7 Computing posterior probabilities with HMM

We are now ready to run the HMM and compute posterior probabilities of modification on our example data set. Due to the short length of the 18S molecule, we will be omitting the sequence bias-correcting step (which is mostly designed for transcriptome studies). Instead, we will use all nucleotide positions:

```
nuclPosition = list()
nuclPosition[[1]] = 1:nchar(sequence)
## First position
nuclPosition[[1]][1]

## [1] 1

## Last position
nuclPosition[[1]][length(nuclPosition[[1]])]

## [1] 1800
```

We are going to run the HMM on all uninterrupted stretches of nucleotides. However, it is possible to only select stretches of interest, e.g. those overlapping with particular genes.

The function `computeProbs` computes the posterior probabilities of modification for all nucleotides, selected in the list `nuclSelection$computePosteriors` returned by the `selectNuclPos` function. It requires matrices with transformed LDRs `LDR_C` and `LDR_CT`, the numbers of replicates `Nc` and `Nt`, the strand indicator, the list of positions addressing the sequence bias `nuclPosition`, the lists of nucleotide positions selected for pair-wise comparisons stored in `nuclSelection`, and the list of stretches on which to run the HMM:

```
posteriors <- computeProbs(LDR_C, LDR_CT, Nc, Nt, '+', nuclPosition,
                           nuclSelection$analysedC, nuclSelection$analysedCT,
                           stretches)

## Computing quantiles of null distributions...
## Computing empirical p-values...
## Computing posteriors...
```

The function first compares LDRs arising from treament-control comparisons with the null distribution (of the corresponding nucleotide pattern if multiple patterns are used) by computing p-values, defined as $1 - q$, for q being the closest percentile to the given LDR.

This way, LDRs reflecting large differences in drop-off rates between treatment and control (larger than those observed by chance) will be larger than most values in the null distribution and will thus get a small p-value. These p-values are then passed to the HMM and posterior probabilities are computed for each selected nucleotide of being in the unmodified (first column in `posteriors`) and the modified state (second column in `posteriors`).

We see that the first few nucleotides have larger probabilities of being unmodified by the chemical probe.

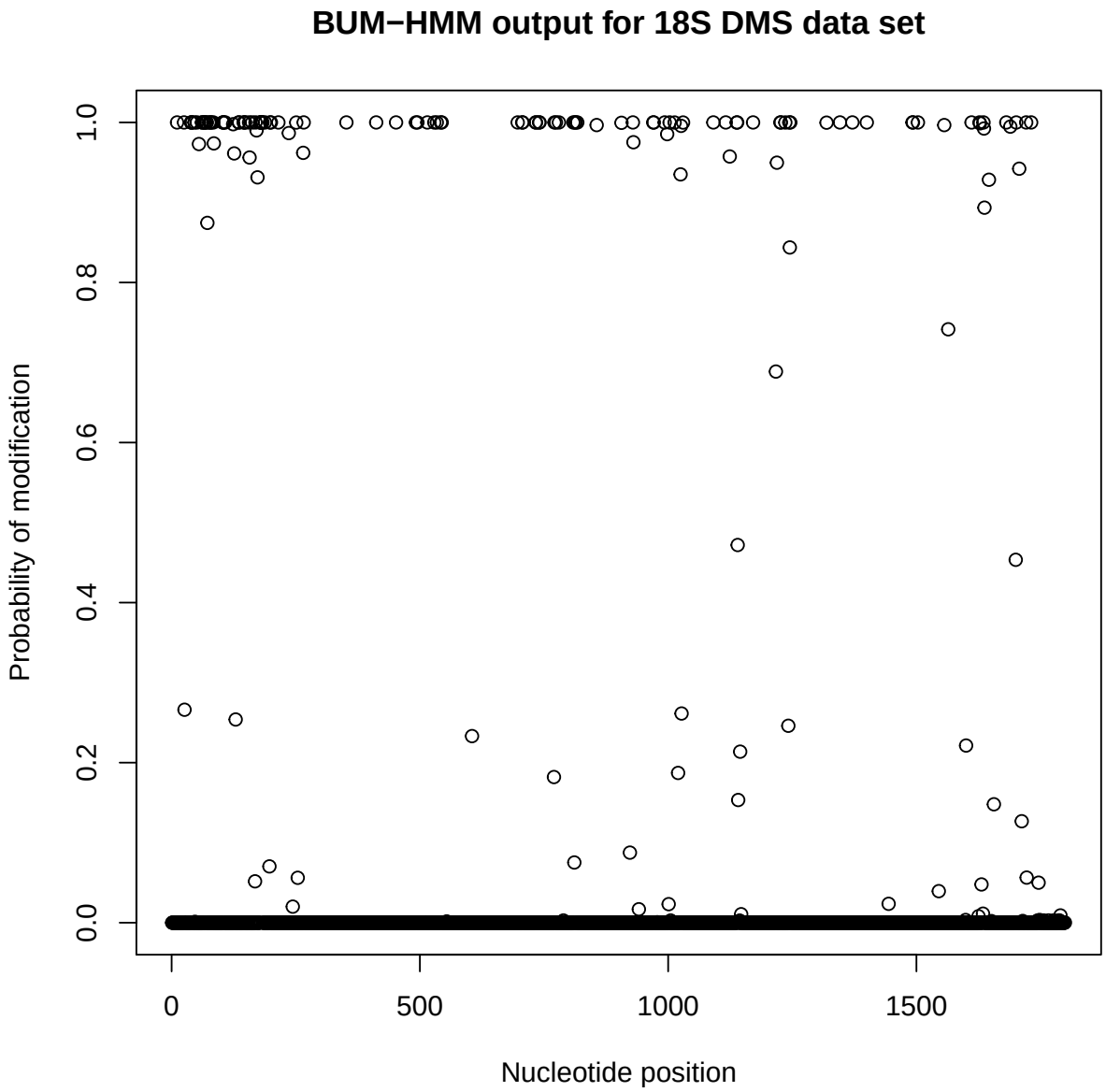
```
head(posteriors)

##      [,1]         [,2]
## [1,]    1 7.410487e-56
## [2,]    1 8.033753e-95
## [3,]    1 3.823135e-42
## [4,]    1 2.361604e-37
## [5,]    1 1.626158e-27
## [6,]    1 1.584615e-38
```

Because the positions of a transcript that were modified in the experiment are the ones at which the reverse transcriptase drops off, the last positions of the corresponding cDNA fragments that are detected and for which we consequently increment the drop-off counts are just upstream of the modification sites. Thus, we need to shift all of our probabilities up by one position:

```
shifted_posteriors <- matrix(, nrow=dim(posteriors)[1], ncol=1)
shifted_posteriors[1:(length(shifted_posteriors) - 1)] <-
                  posteriors[2:dim(posteriors)[1], 2]
```

We can now plot our probabilities to see the output of BUM-HMM for the structure probing data obtained with DMS probe for the yeast ribosomal RNA 18S.

```
plot(shifted_posteriors, xlab = 'Nucleotide position',
     ylab = 'Probability of modification',
     main = 'BUM-HMM output for 18S DMS data set')
```



We see that most nucleotides are predicted to be in the unmodified state (and thus could either be double-stranded or protected by a protein interaction). However, the model also identified modified regions, which should in theory correspond to accessible parts of the molecule.

We also provide an option to optimise the shape parameters of the Beta distribution, which defines the emission model of the HMM for the modified state, with the EM algorithm. To do this, the `computeProbs` function should be called with the last parameter `optimise` set to the desired tolerance. Once the previous and the current estimates of the parameters are within this tolerance, the EM algorithms stops (unless it already reached the maximum number of iterations before that). Further details can be found in [1].

```
posteriors <- computeProbs(LDR_C, LDR_CT, Nc, Nt, '+', nuclPosition,
                           nuclSelection$analysedC, nuclSelection$analysedCT,
                           stretches, 0.001)
```

Normally, the last parameter is set to a default value of `NULL`. We discovered during our experiments that this optimisation appeared vulnerable to local minima. Thus, the default version of the BUM-HMM pipeline does not currently use it.

References

[1] A. Selega, C. Sirocchi, I. Iosub, S. Granneman, and G. Sanguinetti, "Robust statistical modeling improves sensitivity of high-throughput rna structure probing experiments.," *Nature Methods*, in press.

[2] R. C. Spitale, P. Crisalli, R. A. Flynn, E. A. Torre, E. T. Kool, and H. Y. Chang, "Rna shape analysis in living cells," *Nature chemical biology*, vol. 9, no. 1, pp. 18–20, 2013.

[3] R. D. Hector, E. Burlacu, S. Aitken, T. Le Bihan, M. Tuijtel, A. Zaplatina, A. G. Cook, and S. Granneman, "Snapshots of pre-rrna structural flexibility reveal eukaryotic 40s assembly dynamics at nucleotide resolution," *Nucleic acids research*, p. gku815, 2014.