BUMHMM: Probabilistic computational pipeline for modelling RNA structure probing data

Alina Selega[1]

Modified: October 25, 2016. Compiled: June 15, 2024

Contents

1 Introduction

RNA structure is known to be a key regulator of many important mechanisms, such as RNA stability, transcription, and mRNA translation. RNA structural regulatory elements are interrogated with chemical and enzymatic structure probing [1]. In these experiments, a chemical agent reacts with the RNA molecule in a structure-dependent way, cleaving or otherwise modifying its flexible parts. These modified positions can then be detected, providing valuable structural information that can be used for structure prediction [2].

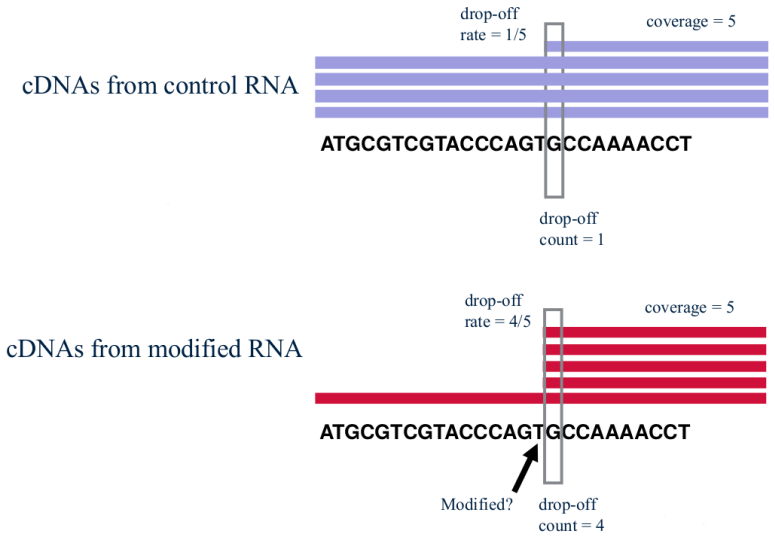
Specifically, chemical modification terminates the reverse transcription reaction, resulting in the reverse transcriptase (RT) dropping off at the modified positions. These positions of drop-off can be then mapped back to the reference sequence. However, the challenge lies

[1] alina.selega@ed.ac.uk or alina.selega@gmail.com

in the stochasticity of this process as the RT can also drop off randomly. To address this, a complementary control experiment is routinely performed to monitor random RT drop-offs when no reagent is used.

Let us consider a toy example of data obtained in a paired-end sequencing structure probing experiment (Fig. 1). We'll focus on a particular nucleotide G and analyse the data from a control experiment (with no reagent added) and a treatment experiment (with RNAs modified by the reagent). In control conditions, we mapped 5 fragments overlapping with the nucleotide G, one of which also terminated at that position. Thus, this nucleotide had a coverage of 5 and a drop-off count of 1 (the number of times the RT dropped off immediately after this position), giving it a *drop-off rate* of $\frac{1}{5}$ (formally defined below). In treatment conditions, more fragments terminated at this position and we measured a drop-off rate of $\frac{4}{5}$. This seems to suggest that the next nucleotide T has been modified by the reagent and perhaps corresponds to a flexible site within the molecule. However, would our conclusion remain the same had we observed a higher drop-off rate in control conditions to start with? In fact, how high would this control drop-off rate have to be for us to dismiss the drop-off rate of $\frac{4}{5}$ as a noisy measurement of randrom drop-off rather than an indication of real modification?

Figure 1: Toy example of structure probing data.



This question reinforces the need for deciding statistically whether the drop-off rate in treatment conditions is significantly higher than the drop-off rate in control. To do this, we must understand how much noise can be expected in control conditions. If the treatment drop-off rate is outside of this range of drop-off rate variability, then we could deem it as significantly higher.

We developed Beta-Uniform Mixture hidden Markov model (`BUM-HMM`) [3], a statistical framework for modelling reactivity scores from an RNA structure probing experiment such as SHAPE [4] or ChemModSeq [5]. `BUM-HMM` implements the intuition outlined above by utilising data from multiple experimental replicates and quantifying the variability of the RT drop-offs. `BUM-HMM` also provides empirical strategies to correct intrinsic biases in the data. It generates a probabilistic output measuring the probability of modification for each nucleotide transcriptome-wide.

The BUMHMM package implements the functionality of the BUM-HMM model. This vignette provides an example workflow for using the BUMHMM package on the structure probing data set for the yeast ribosomal RNA 18S, obtained in an experiment with random priming and paired-end sequencing (available in the Gene Expression Omnibus under accession number GSE52878).

2 Data format

The BUMHMM pipeline requires three data sets for all nucleotide positions:

- the coverage (or the number of reads overlapping with this position),
- the drop-off count (or the number of times the RT dropped off at the next nucleotide),
- and the drop-off rate at this position.

The coverage and the drop-off counts are the data obtained in a structure probing experiment with paired-end sequencing. The drop-off rate r at each nucleotide position is computed as the ratio between its drop-off count k and the coverage n: $r = \frac{k}{n}$. Such data sets can be easily stored in a *SummarizedExperiment* object, commonly used to represent data from sequencing-based experiments such as RNA-Seq.

The key strength of the BUM-HMM model is accounting for the biological variability of the data and thus, it requires data sets available in multiple replicates. The data set `se` provided with this package (accession number GSE52878) is available in triplicates and was obtained in a structure probing experiment on the 18S ribosomal RNA using the DMS chemical probing agent [6].

```
suppressPackageStartupMessages({
    library(BUMHMM)
    library(Biostrings)
    library(SummarizedExperiment)
})
se
```

```
## class: SummarizedExperiment
## dim: 1800 6
## metadata(0):
## assays(3): coverage dropoff_count dropoff_rate
## rownames: NULL
## rowData names(1): nucl
## colnames(6): C1 C2 ... T2 T3
## colData names(1): replicate
```

We see that 18S has 1,800 nucleotides (represented as rows in `se`) and that the data set has 6 replicates: 3 control experiments followed by 3 treatment experiments (represented as columns in `se`). The assays correspond to different data sets, namely, the coverage, drop-off count, and drop-off rate information for each nucleotide. One could quickly access the coverage information for control experimental replicates (labelled 'C1', 'C2', 'C3') as follows:

```
controls <- se[, se$replicate == "control"]
head(assay(controls, 'coverage'))
```

```
##          C1     C2     C3
## [1,] 382943 276113 135209
```

```
## [2,] 399211 280523 137706
## [3,] 403519 284622 140211
## [4,] 404242 287589 144656
## [5,] 404463 288348 147167
## [6,] 404506 288445 147687
```

We also provide the associated genomic sequence of 18S, accessible through the `rowData` function which stores information about the rows, in this case corresponding to the nucleobases:

```
rowData(controls)[1:4,]
```

```
## Views on a 1800-letter DNAString subject
## subject: TATCTGGTTGATCCTGCCAGTAGTCATATGCTT...TTTCCGTAGGTGAACCTGCGGAAGGATCATTA
## views:
##       start end width
##   [1]     1   1     1 [T]
##   [2]     2   2     1 [A]
##   [3]     3   3     1 [T]
##   [4]     4   4     1 [C]
```

Similarly, the function `colData` stores the description of the columns, which correspond to the experimental replicates:

```
colData(controls)
```

```
## DataFrame with 3 rows and 1 column
##      replicate
##    <character>
## C1     control
## C2     control
## C3     control
```

For transcriptome-wide experiments, the data over different chromosomes should be concatenated row-wise.

To briefly illustrate the data set, let us examine the 300th nucleotide:

```
pos <- 300
assay(controls, 'coverage')[pos, 1]
```

```
##     C1
## 813073
```

```
assay(controls, 'dropoff_count')[pos, 1]
```

```
##   C1
## 1837
```

```
assay(controls, 'dropoff_rate')[pos, 1]
```

```
##         C1
## 0.00225933
```

We see that it had coverage of 813073 in the first control experimental replicate, of which the reverse transcription randomly terminated at that position 1837 times, giving it a drop-off rate of 0.00226.

```
treatments <- se[, se$replicate == "treatment"]
assay(treatments, 'coverage')[pos, 1]
```

```
##      T1
## 1640501
```

```
assay(treatments, 'dropoff_count')[pos, 1]
```

```
##   T1
## 4844
```

```
assay(treatments, 'dropoff_rate')[pos, 1]
```

```
##          T1
## 0.002952757
```

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

3 The overview of pipeline

The logic of structure probing experiments associates the binding accessibility of a nucleotide with its structural flexibility, i.e. double-stranded nucleotides or those otherwise protected (e.g. by a protein interaction) will not be available for interaction with the chemical reagent. In contrast, those nucleotides located in flexible parts of the molecule, could be chemically modified by the reagent and will therefore correspond to the positions at which the RT drops off. Thus, we expect the nucleotides immediately downstream from the modification sites within the transcript to have a high drop-off rate in the presence of a reagent; higher than what we observe in control conditions.

To quantify the variability in drop-off rate measured in control conditions, the `BUM-HMM` method compares the drop-off rates at each nucleotide position between two *control* experimental replicates, C_i and C_j:

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

If the drop-off rates r_{C_i} and r_{C_j} are similar in a pair of control replicates, the above log-ratio will be close to 0, indicating little to no variability in drop-off rate. In contrast, different drop-off rates will 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 how much variability in drop-off rate we can observe between two experimental replicates simply by chance, in the absence of any reagent. (Note that due to a log transform, the drop-off rates $r = 0$ are not allowed.)

We now compute this log-ratio between the drop-off rates in all pairs of *treatment* and *control* experimental replicates, T_i and C_j:

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

We expect the neighbour of a modified nucleotide to have a much larger drop-off rate in a treatment experiment compared to control conditions, generating a large log-ratio for this pair of experimental replicates. By comparing each treatment-control log-ratio to the null distribution, we can find those nucleotide positions that demonstrate differences in drop-off rate larger than what can be expected by chance.

The next section goes through the steps of the `BUMHMM` pipeline using the provided data set `se` as an example.

4 BUMHMM pipeline steps

4.1 Selecting pairs of nucleotides

We first need to select nucleotide positions in each experimental replicate for which we will compute the log-ratios. This is implemented with the function `selectNuclPos`. This function requires the coverage and drop-off count information stored in `se`, the numbers of control and treatment experimental replicates (`Nc` and `Nt`, correspondingly), and a user-specified coverage threshold `t`. Nucleotides with coverage $n < t$ will not be considered.

In our data set, we have 3 control and 3 treatment replicates, so if we set the minimum allowed coverage as $t = 1$, we can make the following function call:

```
Nc <- Nt <- 3
t <- 1
nuclSelection <- selectNuclPos(se, Nc, Nt, t)
List(nuclSelection)
```

```
## List of length 2
## names(2): analysedC analysedCT
```

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

- `analysedC` is a list where each element corresponds to a control-control replicate comparison. Each element holds indices of nucleotides that have coverage $n >= t$ and a drop-off count $k > 0$ in both replicates of that comparison. Thus, each element stores those nucleotide positions for which we can compute the log-ratio for the corresponding pair of control replicates.
- `analysedCT` is a list where each element corresponds to a treatment-control replicate comparison. Again, each element holds indices of nucleotides that have coverage $n >= t$ and a drop-off count $k > 0$ in both replicates of that comparison.

The pairwise control replicate comparisons are enumerated with the function `combn` from the `utils` package:

```
t(combn(Nc, 2))
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    1    3
## [3,]    2    3
```

Thus, the first element of `analysedC` corresponds to comparing the control replicate 1 to control replicate 2. The comparisons between treatment and control replicates are computed similarly.

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

```
## [1] 1713
```

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

```
## [1] 1723
```

We select 1713 nucleotide positions for computing the log-ratios for the first control-control comparison and 1723 positions for the first treatment-control comparison.

4.2 Scaling the drop-off rates across replicates

Because BUMHMM works with data collected in multiple experimental replicates, it is important to ensure that the drop-off rates do not differ dramatically between replicates. Thus, the second step of the pipeline scales the drop-off rates of nucleotides selected for pairwise comparisons to have a common median value. This is implemented with a function `scaleDOR`, which requires the data container `se`, the output of the function `selectNuclPos` (described above), and the numbers of replicates. It returns the updated drop-off rates such that the selected positions have the same median drop-off rate in all replicates:

```
## Medians of original drop-off rates in each replicate
apply(assay(se, 'dropoff_rate'), 2, median)
```

```
##          C1          C2          C3          T1          T2          T3
## 0.001005071 0.001025182 0.003684315 0.001608314 0.001302076 0.003380141
```

```
## Scale drop-off rates
assay(se, "dropoff_rate") <- scaleDOR(se, nuclSelection, Nc, Nt)

## Medians of scaled drop-off rates in each replicate
apply(assay(se, 'dropoff_rate'), 2, median)
```

```
##          C1          C2          C3          T1          T2          T3
## 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 most, but not all nucleotides were selected for the pairwise comparisons.)

4.3 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 state corresponding to the true state of each nucleotide: modified or unmodified.

The observations of the HMM are the empirical p-values associated with each nucleotide. These p-values arise from comparing the treatment-control log-ratios corresponding to each nucleotide position with the null distribution:

$$p\text{-value} = 1 - \text{closest percentile of null distribution}$$

If the difference between the drop-off rates in treatment and control replicates (as measured by the treatment-control log-ratio) is well within the range of the drop-off rate variability that we observed in control conditions (as summarised by the null distribution), then this log-ratio will get assigned a fairly large p-value. However, those log-ratios that are much larger than most values in the null distribution will be close to its right side (e.g. 90^{th} percentile). They will then receive a small p-value ($1-0.9=0.1$ in this case). Thus, the p-value can be thought of as a probability for the treatment-control log-ratio to belong to the null distribution. We are interested in those log-ratios that are unlikely to belong to it as they could indicate the real RT drop-off signal. Note that we expect the drop-off rate in treatment conditions to be higher than in control, which is why we restrict our attention to the right side of the null distribution.

Modelling p-values directly enabled us to define the emission distribution of the HMM as a Beta-Uniform mixture model. Briefly, the unmodified state of a nucleotide corresponds to the null hypothesis and the associated p-values are modelled with the Uniform distribution. In the modified state we expect to see large log-ratios and small associated p-values, which are modelled with a Beta distribution. Further details and theoretical justifications can be found in [3].

To run the HMM, we compute uninterrupted stretches of nucleotides for which the posterior probabilities are to be computed. Posterior probabilities will be computed for those nucleotides with at least the minimum allowed coverage in all experimental replicates and a non-zero drop-off count in at least one treatment replicate. This is achieved with the function `computeStretches`, which takes `se` and the threshold `t` as parameters.

```
stretches <- computeStretches(se, t)
```

The function returns an *IRanges* object where each element corresponds to a stretch of nucleotides and each stretch is at least 2 nucleotides long. HMM will be run separately on each stretch.

```
head(stretches)
```

```
## IRanges object with 2 ranges and 0 metadata columns:
##           start       end     width
##       <integer> <integer> <integer>
##   [1]         1      1747      1747
##   [2]      1749      1800        52
```

```
assay(se, 'dropoff_count')[1748,]
```

```
## C1 C2 C3 T1 T2 T3
##  0  0  0  0  0  0
```

On this data set, we will compute posterior probabilities for all nucleotides but one, which is at the 1748th position. This is because at this position, all treatment replicates (and in fact, all replicates) had a drop-off count of 0.

4.4 Bias correction

Using a transcriptome-wide data set, we identified sequence and coverage as factors that influence log-ratios in control conditions [3]. We would therefore like to transform the log-ratios such that these biases are eliminated and the performed comparisons are not confounded.

4.4.1 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 log-ratios 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 [3] for details) and is fitted to the data with a non-linear least squares technique. Then, all log-ratios (both for control-control and treatment-control comparisons) are transformed accordingly so that this dependency on coverage is eliminated or at least reduced.

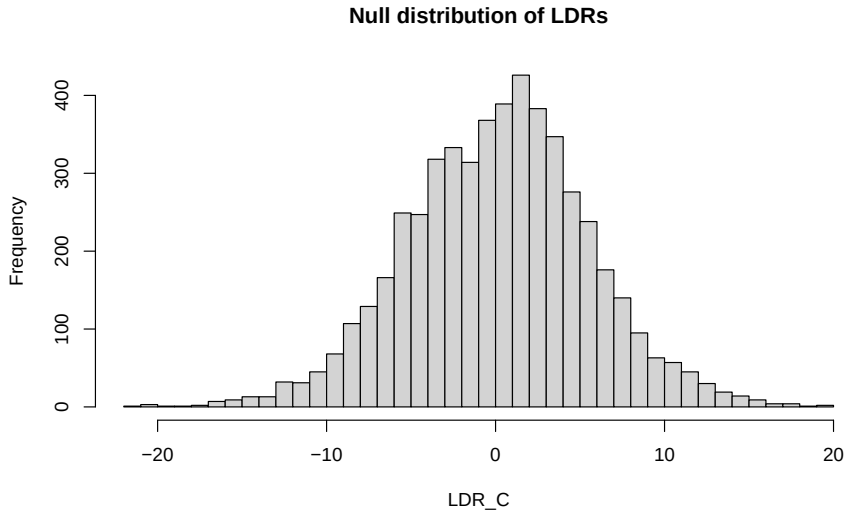
The function requires the data container `se`, the positions of nucleotides selected for pairwise comparisons, and the numbers of replicates. It returns a list with two elements (`LDR` stands for "log drop-off rate ratio"):

- `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 nucleotide positions and columns – to a pairwise comparison. Thus, `LDR_C` has as many columns as there are control-control comparisons (3 comparisons for 3 control replicates) and `LDR_CT` has as many columns as treatment-control comparisons (9 comparisons for 3 control and 3 treatment replicates).

```
varStab <- stabiliseVariance(se, nuclSelection, Nc, Nt)
LDR_C <- varStab$LDR_C
LDR_CT <- varStab$LDR_CT

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



The histogram shows the null distribution after the transformation.

4.4.2 Sequence bias

The sequence-dependent bias is addressed by computing different null distributions of log-ratios for different sequence patterns of nucleotides. One could consider sequences of three nucleotides, reflecting the assumption that the immediate neighbours of a nucleotide on both sides could affect its accessibility; patterns of other lengths could 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"
## [13] "ATA" "ATC" "ATG" "ATT" "CAA" "CAC" "CAG" "CAT" "CCA" "CCC" "CCG" "CCT"
## [25] "CGA" "CGC" "CGG" "CGT" "CTA" "CTC" "CTG" "CTT" "GAA" "GAC" "GAG" "GAT"
## [37] "GCA" "GCC" "GCG" "GCT" "GGA" "GGC" "GGG" "GGT" "GTA" "GTC" "GTG" "GTT"
## [49] "TAA" "TAC" "TAG" "TAT" "TCA" "TCC" "TCG" "TCT" "TGA" "TGC" "TGG" "TGT"
## [61] "TTA" "TTC" "TTG" "TTT"
```

Considering patterns of length $n = 3$ will result in computing 64 different null distributions of log-ratios, each corresponding to one sequence 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 (e.g. a `DNAString` object), and a parameter indicating whether we are dealing with sense (+) or anti-sense (-) DNA strand. For transcriptome-wide experiments, when searching for pattern occurrences within the genomic sequence on the anti-sense strand (sense strand sequence is expected by the function as the second parameter), the patterns will be converted to complementary sequence.

```
## Extract the DNA sequence
sequence <- subject(rowData(se)$nucl)
sequence
```

```
## 1800-letter DNAString object
## seq: TATCTGGTTGATCCTGCCAGTAGTCATATGCTTGT...GGTTTCCGTAGGTGAACCTGCGGAAGGATCATTA
```

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

```
## [1] "AAA"
```

```
head(nuclPosition[[1]])
```

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

The function returns a list with an element corresponding to each pattern generated by `nuclPerm`. Each element holds the indices of the middle nucleotide of this pattern's occurrence in the genomic sequence. Thus, we will separately consider the drop-off rates of the nucleotides that occur in the context of each pattern. For instance, the null distribution specific to the pattern "AAA" will be constructed from the log-ratios for the nucleotide positions 40, 85, 104, 180, 181, 218 etc.

4.5 Computing posterior probabilities with HMM

We are now ready for the final step of the pipeline which computes posterior probabilities of modification with the HMM. Due to the short length of the 18S molecule (only 1,800 nucleotides), we will be omitting the sequence bias-correcting step, which is primarily designed for transcriptome studies. Instead, we will use all nucleotide positions for constructing a single null distribution quantifying the drop-off rate variability. The `nuclPosition` list should therefore have one element corresponding to the single stretch that we will run the HMM on. This stretch contains all nucleotide positions:

```
nuclPosition <- list()
nuclPosition[[1]] <- 1:nchar(sequence)

## Start of the stretch
nuclPosition[[1]][1]

## [1] 1

## End of the stretch
nuclPosition[[1]][length(nuclPosition[[1]])]

## [1] 1800
```

The HMM is run separately on all stretches of nucleotides (of which we have only one in this particular case). However, in transcriptome studies it could be useful to only select some stretches of interest, e.g. those overlapping with particular genes.

The function `computeProbs` computes the posterior probabilities of modification for all nucleotides in the specified stretches. The function requires matrices with transformed log-ratios `LDR_C` and `LDR_CT`, the numbers of replicates, the strand indicator, the lists of positions for computing the null distribution(-s) (stored in `nuclPosition`) and pairwise comparisons (stored in `nuclSelection`), and the stretches which to run the HMM on:

```
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 `computeProbs` compares log-ratios to the null distribution(-s) and computes empirical p-values. These are then passed as observations to the HMM, which computes posterior probabilities for each selected nucleotide of being in the unmodified (first column in `posteriors`) and modified state (second column in `posteriors`).

5 BUMHMM output

We see that the model assigns very large probabilities to the first few nucleotides to be unmodified by the chemical probe.

```
head(posteriors)

##      [,1]         [,2]
```

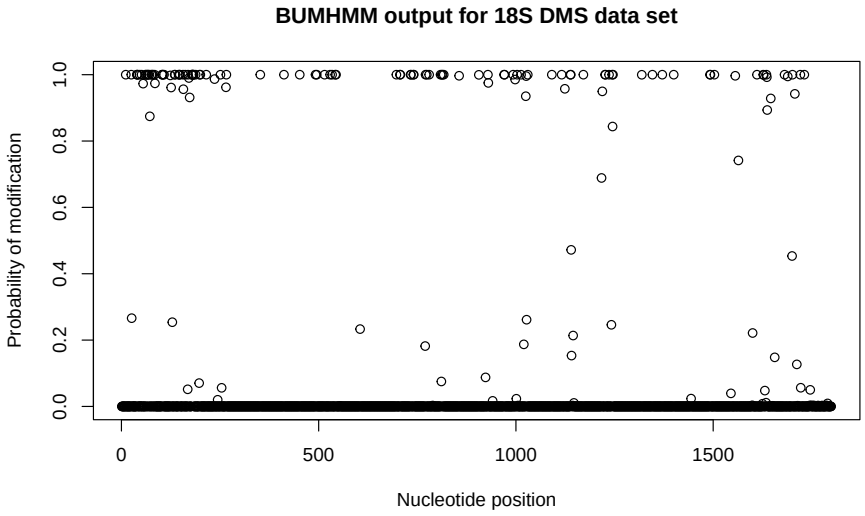
```
## [1,]    1 7.410487e-56
## [2,]    1 8.033753e-95
## [3,]    1 3.823135e-42
## [4,]    1 2.361604e-37
## [5,]    1 1.626158e-27
## [6,]    1 1.584615e-38
```

As the modified positions within a transcript are the ones at which the reverse transcriptase drops off, the last position of the corresponding cDNA fragment that is detected and for which we consequently increment the drop-off count is the nucleotide next to the modification site. Thus, we need to shift 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 the probabilities to see the `BUMHMM` output for the DMS structure probing data for the yeast rRNA 18S.

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



We see that most nucleotides are predicted to be in the unmodified state, having the probability of modification close to 0 (and thus could possibly be double-stranded or protected by a protein interaction). However, the model also identified some modified regions, which could correspond to accessible parts of the molecule. It should be noted that the DMS probe preferentially reacts with "A" and "C" nucleotides, which effectively makes only a subset of the structural state of the molecule accessible to probing.

The `BUMHMM` package also provides an option to optimise the shape parameters of the Beta distribution, which defines the HMM emission model for the modified state. To optimise parameters with the EM algorithm, the `computeProbs` function should be called with the last

parameter `optimise` set to the desired tolerance. Once the previous and 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 [3].

```
## Call the function with the additonal tolerance parameter
posteriors <- computeProbs(LDR_C, LDR_CT, Nc, Nt, '+', nuclPosition,
                           nuclSelection$analysedC, nuclSelection$analysedCT,
                           stretches, 0.001)
```

By default, the last parameter is set to `NULL`. During our experiments, we discovered that this optimisation appeared vulnerable to local minima. Thus, the current version of the `BUMHMM` pipeline does not use this optimisation.

6 Session Info

This vignette was compiled using:

```
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24)
## Platform: x86_64-pc-linux-gnu
## Running under: Ubuntu 24.04 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.26.so;  LAPACK version 3.12.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## time zone: Etc/UTC
## tzcode source: system (glibc)
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] SummarizedExperiment_1.35.0 Biobase_2.65.0
##  [3] GenomicRanges_1.57.1        MatrixGenerics_1.17.0
##  [5] matrixStats_1.3.0           Biostrings_2.73.1
##  [7] GenomeInfoDb_1.41.1         XVector_0.45.0
##  [9] IRanges_2.39.0              S4Vectors_0.43.0
## [11] BiocGenerics_0.51.0         BUMHMM_1.29.0
##
## loaded via a namespace (and not attached):
##  [1] xfun_0.44              htmlwidgets_1.6.4      devtools_2.4.5
```

```
##  [4] remotes_2.5.0           lattice_0.22-6          vctrs_0.6.5
##  [7] tools_4.4.0             highr_0.11              Matrix_1.7-0
## [10] lifecycle_1.0.4         GenomeInfoDbData_1.2.12 compiler_4.4.0
## [13] stringr_1.5.1           tinytex_0.51            BiocStyle_2.33.1
## [16] httpuv_1.6.15           htmltools_0.5.8.1       sys_3.4.2
## [19] usethis_2.2.3           buildtools_1.0.0        yaml_2.3.8
## [22] later_1.3.2             crayon_1.5.2            urlchecker_1.0.1
## [25] ellipsis_0.3.2          cachem_1.1.0            DelayedArray_0.31.1
## [28] sessioninfo_1.2.2       abind_1.4-5             mime_0.12
## [31] gtools_3.9.5            digest_0.6.35           stringi_1.8.4
## [34] purrr_1.0.2             maketools_1.3.0         fastmap_1.2.0
## [37] grid_4.4.0              cli_3.6.2               SparseArray_1.5.8
## [40] magrittr_2.0.3          S4Arrays_1.5.1          pkgbuild_1.4.4
## [43] UCSC.utils_1.1.0        promises_1.3.0          rmarkdown_2.27
## [46] httr_1.4.7              memoise_2.0.1           shiny_1.8.1.1
## [49] evaluate_0.24.0         knitr_1.47              miniUI_0.1.1.1
## [52] profvis_0.3.8           rlang_1.1.4             Rcpp_1.0.12
## [55] xtable_1.8-4            glue_1.7.0              BiocManager_1.30.23
## [58] pkgload_1.3.4           jsonlite_1.8.8          R6_2.5.1
## [61] fs_1.6.4                zlibbioc_1.51.1
```

7 Acknowledgements

This package was developed at the the School of Informatics, University of Edinburgh with support from Sander Granneman and Guido Sanguinetti.

The manuscript describing the `BUM-HMM` computational model can be found in [3].

This study was supported in part by the grants from the UK Engineering and Physical Sciences Research Council, Biotechnology and Biological Sciences Research Council, and the UK Medical Research Council to the University of Edinburgh Doctoral Training Centre in Neuroinformatics and Computational Neuroscience (EP/F500385/1 and BB/F529254/1).

References

[1] Miles Kubota, Catherine Tran, and Robert C Spitale. Progress and challenges for chemical probing of rna structure inside living cells. *Nature chemical biology*, 11(12):933–941, 2015.

[2] Yang Wu, Binbin Shi, Xinqiang Ding, Tong Liu, Xihao Hu, Kevin Y Yip, Zheng Rong Yang, David H Mathews, and Zhi John Lu. Improved prediction of rna secondary structure by integrating the free energy model with restraints derived from experimental probing data. *Nucleic acids research*, page gkv706, 2015.

[3] Alina Selega, Christel Sirocchi, Ira Iosub, Sander Granneman, and Guido Sanguinetti. Robust statistical modeling improves sensitivity of high-throughput rna structure probing experiments. *Nature Methods*, 2016.

[4] Robert C Spitale, Pete Crisalli, Ryan A Flynn, Eduardo A Torre, Eric T Kool, and Howard Y Chang. Rna shape analysis in living cells. *Nature chemical biology*, 9(1):18–20, 2013.

[5] Ralph D Hector, Elena Burlacu, Stuart Aitken, Thierry Le Bihan, Maarten Tuijtel, Alina Zaplatina, Atlanta G Cook, and Sander Granneman. Snapshots of pre-rrna structural flexibility reveal eukaryotic 40s assembly dynamics at nucleotide resolution. *Nucleic acids research*, page gku815, 2014.

[6] Sandra E Wells, John MX Hughes, A Haller Igel, and Manuel Ares. Use of dimethyl sulfate to probe rna structure in vivo. *Methods in enzymology*, 318:479–493, 2000.