Getting started with BaMORC

BaMORC quickstart guide

The goal of this document is to get you up and running with BaMORC as quickly as possible. BaMORC is designed as Bayesian Model Optimized Reference Correction (BaMORC) Method for Assigned and Unassigned Protein NMR Spectra. For a detailed explanation of the algorithm please refer to "Automatic 13C chemical shift reference correction for unassigned protein NMR spectra".

There are two important parts to BaMORC: the `bamorc()`, reference correction function for assigned 13C protein NMR spectra, and the `unassigned_bamorc()`, reference correction function for unassigned 13C protein NMR spectra. In the first section, you'll learn about the basics of running both of the functions. In the second section, you will dive into the basics of input data process, and in third section, you'll learn a little more on functions of the behind-the-scenes of the BaMORC algorithm.

BaMORC basics

To make a correction, first load BaMORC, then call `bamorc()` or `unassigned_bamorc()` with correct arguments:

```
library(BaMORC)
```

For assigned protein NMR spectra:

Here we will using the built-in data to demonstrate the arguments that will be passed in the `bamorc()`.

```
## Arguments:
sequence = paste(RefDB_data$carbonDat[[1]]$AA, collapse = "")
secondary_structure = paste(RefDB_data$carbonDat[[1]]$SS, collapse = "")
chemical_shifts_input = RefDB_data$carbonDat[[1]][, c(4,5)]
from= -5
to = 5

## Running bamorc() function:
bamorc(sequence, secondary_structure, chemical_shifts_input, from=-5, to=5)
#> [1] 0.0142443
```

- `sequence`: the sequence of protein of interest with single-letter convention.
- `secondary_structure`: the secondary structure information of the protein with with single-letter convention.
- `chemical_shifts_input`: <data frame n by 2> the carbon 13 chemical shift of the protein.
- `from` and `to`: the upper- and lower-bound of optimization range. Value assigned to `from` must be lower than `to`.

The length of the `sequence` and `secondary_structure` should be the same, if not pleasue assign `secondary_structure = NULL`, however, the peaklist groups number could be more or less than the length of the sequence. Since commonly the output of the spectra should have a little deviate from the sequence information.

Printing an argument object gives you some useful information: the actual format, the size, the object type, and if it's a string.

For unassigned protein NMR spectra:

Next we will using the built-in data to demonstrate the arguments that will be passed in the `unassigned_bamorc()`. The output will be slight different each time runs due to the randomness of the optimization.

```
## Arguments:
sequence = "RPAFCLEPPYAGPGKARIIRYFYNAAAGAAQAFVYGGVRAKRNNFASAADALAACAAA"
sample_data_generator(input_type = "ssc_sample") # this will generate a temperary sample file and later
file_path = "./bpti_HNcoCACB.txt" # temperary sample file path.

## Running unassigned_bamorc() function:
unassigned_bamorc(peakList_file_loc = file_path, sequence = sequence, secondary_structure = NULL, from =
#> Your job will be submitted with the following parameters:
#> format: seq
#> skipPDB: on
#> file:
#> seq: RPAFCLEPPYAGPGKARIIRYFYNAAAGAAQAFVYGGVRAKRNNFASAADALAACAAA
#> email:
#> name:
#> Created JPred job with jobid: jp_0592K5v
#> You can check the status of the job using the following URL: http://www.compbio.dundee.ac.uk/jpred4/
#> Your job status will be checked with the following parameters:
#> Job id: jp_0592K5v
#> Get results: TRUE
#> Saving results to: /Users/bill/Documents/GitHub/BaMORC/inst/doc/jp_0592K5v/jp_0592K5v
#> JPred is done.
#> The predicted secondary structure is: CCCCCCCCCCCCCCCCBBBBBBBCCCCCBBBBBBBCCCCCCCCCHHHHHHHHHHCCC
#> Running grouping algorithm
#> docker run -v /Users/bill/Documents/GitHub/BaMORC/inst/doc/bpti_HNcoCACB.txt:/ssc/test.txt -t mosele
#> Running reference correction algorithm.
#> [1] 0.0007908663

## Delete the temperary sample file.
unlink("./bpti_HNcoCACB.txt")
```

The data processing:

The data passed in both above functions should be pre-processed. Luckly, we provided a varity of helper functions. I'll show you how to process the data as following.

The sample generating functions:

There are three file reading functions within the BaMORC package: `read_raw_file()`, `read_NMRSTAR_file()`, and `read_DB_File()`. Even though, these functions should handle a wide range of the input files, however, a good file formation are highly recommanded. You can check the sample file using the `sample_data_generator()` with following code:

- Delimiter of white space:

  ```
  input_type = "ws"
  sample_data_generator(input_type = input_type)
  head(read_raw_file("sample_input_ws.txt", delim = "ws"))
  ```

```
#> Parsed with column specification:
#> cols(
#>   X1 = col_double(),
#>   X2 = col_double()
#> )
#> [[1]]
#> [1] "NHQDHNNFQTLPYVPCSTCEGNLACLSLCHIE"
#>
#> [[2]]
#> # A tibble: 32 x 2
#>       X1    X2
#>    <dbl> <dbl>
#>  1  NA    38.6
#>  2  NA    28.8
#>  3  56.3  30.4
#>  4  54.4  41.3
#>  5  NA    28.8
#>  6  55.1  38.9
#>  7  NA    38.8
#>  8  NA    39.3
#>  9  NA    30.3
#> 10  62.1  69.8
#> # ... with 22 more rows
unlink("sample_input_ws.txt")
```

- Delimiter of comma:

```
input_type = "csv"
sample_data_generator(input_type = input_type)
head(read_raw_file("sample_input.csv", delim = "comma"))
#> Parsed with column specification:
#> cols(
#>   X1 = col_double(),
#>   X2 = col_double()
#> )
#> Warning in rbind(names(probs), probs_f): number of columns of result is not
#> a multiple of vector length (arg 2)
#> Warning: 1 parsing failure.
#> row # A tibble: 1 x 5 col     row col   expected  actual    file                expected   <int>
#> [[1]]
#> [1] "NHQDHNNFQTLPYVPCSTCEGNLACLSLCHIE"
#>
#> [[2]]
#> # A tibble: 33 x 2
#>       X1    X2
#>    <dbl> <dbl>
#>  1  NA    38.6
#>  2  NA    28.8
#>  3  56.3  30.4
#>  4  54.4  41.3
#>  5  NA    28.8
#>  6  55.1  38.9
#>  7  NA    38.8
#>  8  NA    39.3
#>  9  NA    30.3
#> 10  62.1  69.8
#> # ... with 23 more rows
unlink("sample_input.csv")
```

- Delimiter of semicolon:

```
input_type = "sc"
sample_data_generator(input_type = input_type)
head(read_raw_file("sample_input_sc.txt", delim = "semicolon"))
#> Parsed with column specification:
#> cols(
#>   X1 = col_double(),
#>   X2 = col_double()
#> )
#> [[1]]
#> [1] "NHQDHNNFQTLPYVPCSTCEGNLACLSLCHIE"
#>
#> [[2]]
#> # A tibble: 32 x 2
#>        X1    X2
#>     <dbl> <dbl>
#>  1  NA    38.6
#>  2  NA    28.8
#>  3  56.3  30.4
#>  4  54.4  41.3
#>  5  NA    28.8
#>  6  55.1  38.9
#>  7  NA    38.8
#>  8  NA    39.3
#>  9  NA    30.3
#> 10  62.1  69.8
#> # ... with 22 more rows
unlink("sample_input_sc.txt")
```

The reading functions will return sequence and chemical shifts:

- `read_raw_file()`: parses user-provided file in customed format as show above examples.
- `read_NMRSTAR_file()`: parses user-provided file in BMRB Star 2/3 format.

```
## Download a BMRB file
library(BMRBr)
bmrb_download(4020, output_dir = "./")
#> [1] "Downloading..."
#> Processing: bmr4020[1] "Downloaded: bmr4020"

## Read in BMRB file and procec
file_path = "bmr4020.str"
head(read_NMRSTAR_file(file_path))
#> [1] "Please note down the ID:f3237b66edd8a96e4d78c580092fe5f2"
#> [1] "ID will expire on  2018-10-24 13:41:24"
#> [[1]]
#>  [1] "N" "H" "Q" "D" "H" "N" "N" "F" "Q" "T" "L" "P" "Y" "V" "P" "C" "S"
#> [18] "T" "C" "E" "G" "N" "L" "A" "C" "L" "S" "L" "C" "H" "I" "E"
#>
#> [[2]]
#>      CA   CB
#> 1    NA 38.6
#> 12   NA 28.8
#> 23 56.3 30.4
#> 27 54.4 41.3
#> 28   NA 28.8
#> 29 55.1 38.9
#> 30   NA 38.8
#> 31   NA 39.3
#> 32   NA 30.3
#> 2  62.1 69.8
#> 3    NA 42.0
#> 4  63.2 31.9
#> 5    NA 39.2
#> 6  60.7 33.6
#> 7    NA 32.2
#> 8    NA 40.2
#> 9  60.8 62.3
#> 10   NA 70.1
#> 11   NA 35.2
#> 13 56.8 27.5
#> 14 45.4   NA
#> 15 54.4 39.6
#> 16 58.5 41.7
#> 17 55.3 17.5
#> 18 59.3 33.8
#> 19 57.4 41.8
#> 20 59.6 63.8
#> 21 57.3 43.8
#> 22 56.9 43.9
#> 24   NA 30.4
#> 25 62.0 38.9
#> 26 55.9 29.6

## Delete downloaded BMRB file
unlink("./bmr4020.str")
```

- `read_DB_File()`: parses file from BMRB by a given entry ID.

```
id = 4022
output <- read_DB_File(id)
head(output[[1]])
#> [1] "A" "S" "P" "D" "W" "G"
head(output[[2]])
#>       CA   CB
#> 1   51.9 19.4
#> 112 57.5 63.3
#> 183 63.1 32.2
#> 194 54.6 41.3
#> 205 54.7 31.4
#> 216 45.4   NA
```

Estimating econdary structure from sequence:

One important part of the optmization for referencing correction value is through providing the secondary structue information basin protein sequence. This is done through the JPred and jpredapi.

```
protein_sequence <- "MQVWPIEGIKKFETLSYLPPLTVEDLLKQI"
secondary_structure <- jpred_fetcher(protein_sequence)
#> Your job will be submitted with the following parameters:
#> format: seq
#> skipPDB: on
#> file:
#> seq: MQVWPIEGIKKFETLSYLPPLTVEDLLKQI
#> email:
#> name:
#> Created JPred job with jobid: jp_w06BsSD
#> You can check the status of the job using the following URL: http://www.compbio.dundee.ac.uk/jpred4/
#> Your job status will be checked with the following parameters:
#> Job id: jp_w06BsSD
#> Get results: TRUE
#> Saving results to: /Users/bill/Documents/GitHub/BaMORC/inst/doc/jp_w06BsSD/jp_w06BsSD
#> JPred is done.
#> The predicted secondary structure is: CCCCCCCCCCCCCBBCCCCCCCHHHHHHCC

secondary_structure
#> [1] "CCCCCCCCCCCCCBBCCCCCCCHHHHHHCC"
```

Behind-the-scenes functions

The BaMORC algorithms relys minimize the difference betwee the actual relative cummulative frequence (RCF) of the protein sequence and estimated RCF from the chemical shifts information. The functions make the algorithm possible are showing below with examples, for detailed function descriptions, please see the reference.:

- `calculate_AA_Prob()`: returns the probability (density) for a certain type of amino acid based on a chi-squared statistics wtih 2 degrees of freedom.
- `calculate_chi_squared_stat()`: given a pair of alpha and beta carbons chemical shifts, this function will return a list of caculated chisquare statistics based on the combination of amino acid typings and secondary structures.
- `calculate_RCF()`: calculates the relative cummulative freqeucny of amino acid and secondary structure combination.

```
## Arguments:
sequence = paste(RefDB_data$carbonDat[[1]]$AA, collapse = "")
secondary_structure = paste(RefDB_data$carbonDat[[1]]$SS, collapse = "")

## Function:
calculate_RCF(sequence, secondary_structure)
#> [[1]]
#>  [1] "S-C" "I-C" "P-C" "C-B" "L-B" "L-B" "S-B" "P-C" "W-B" "S-B" "E-B"
#> [12] "W-B" "S-B" "D-C" "C-B" "S-B" "V-B" "T-C" "C-C" "K-B" "M-B" "R-B"
#> [23] "T-B" "R-B" "Q-B" "R-B" "M-B" "L-B" "K-B" "S-C" "L-C" "A-C" "E-C"
#> [34] "L-C" "D-C" "C-C" "N-C" "E-C" "D-C" "L-B" "E-B" "Q-B" "A-B" "E-B"
#> [45] "K-B" "C-B" "M-B" "L-B" "P-B" "E-B" "C-C" "P-C"
```

```
#>
#> [[2]]
#>    AA_SS       Freq
#> 1    A-B 0.01923077
#> 2    A-C 0.01923077
#> 3    C-B 0.05769231
#> 4    C-C 0.05769231
#> 5    D-C 0.05769231
#> 6    E-B 0.07692308
#> 7    E-C 0.03846154
#> 8    I-C 0.01923077
#> 9    K-B 0.05769231
#> 10   L-B 0.09615385
#> 11   L-C 0.03846154
#> 12   M-B 0.05769231
#> 13   N-C 0.01923077
#> 14   P-B 0.01923077
#> 15   P-C 0.05769231
#> 16   Q-B 0.03846154
#> 17   R-B 0.05769231
#> 18   S-B 0.07692308
#> 19   S-C 0.03846154
#> 20   T-B 0.01923077
#> 21   T-C 0.01923077
#> 22   V-B 0.01923077
#> 23   W-B 0.03846154
```

- `calculate_MSE()`: calculates mean squared error.

```
## chemicalShifts and aaFreq are predefined sample variables for demo purpose within the BaMORC Package
calculate_MSE(step_ca=1, step_cb=1, dat_cacb=chemicalShifts[, c(3,4)], aa_Freq=aaFreq)
#> [1] 0.0003151677
```