Local Score Package

Sébastien Déjean - Sabine Mercier - Sebastian Simon - David Robelin

2025-04-18

Contents

Introduction

Main purpose: given a numerical sequence of numerical scores, find the maximum sum subsequence value among all possible subsequences. This value is called the local Score.

This package provides functionalities for two main tasks: 1- Calculating the local Score of a given score sequence or, of a given component sequence and a given scoring scheme. 2- Calculating the statistical relevance (p-value) of a given local Score, associated to a given sequence length and a given distribution for the model.

Also, the package deals with sub-optimal local scores, that is all strictly positive sum subsequences. Since the second version of this package, it can also calculates the p-value of a sub-optimal local scores given its position in sequential order.

This second version of the package deals with a model of independent and identically distributed (I.I.D.) and markov dependent sequences.

If your in a hurry, the section "In brief" presents the main functions and there typical uses. Details and other useful functions are presented in the remaining of this vignette.

In brief

In all the examples below, the score expectation should strictly negative, so that the local score has a sense.

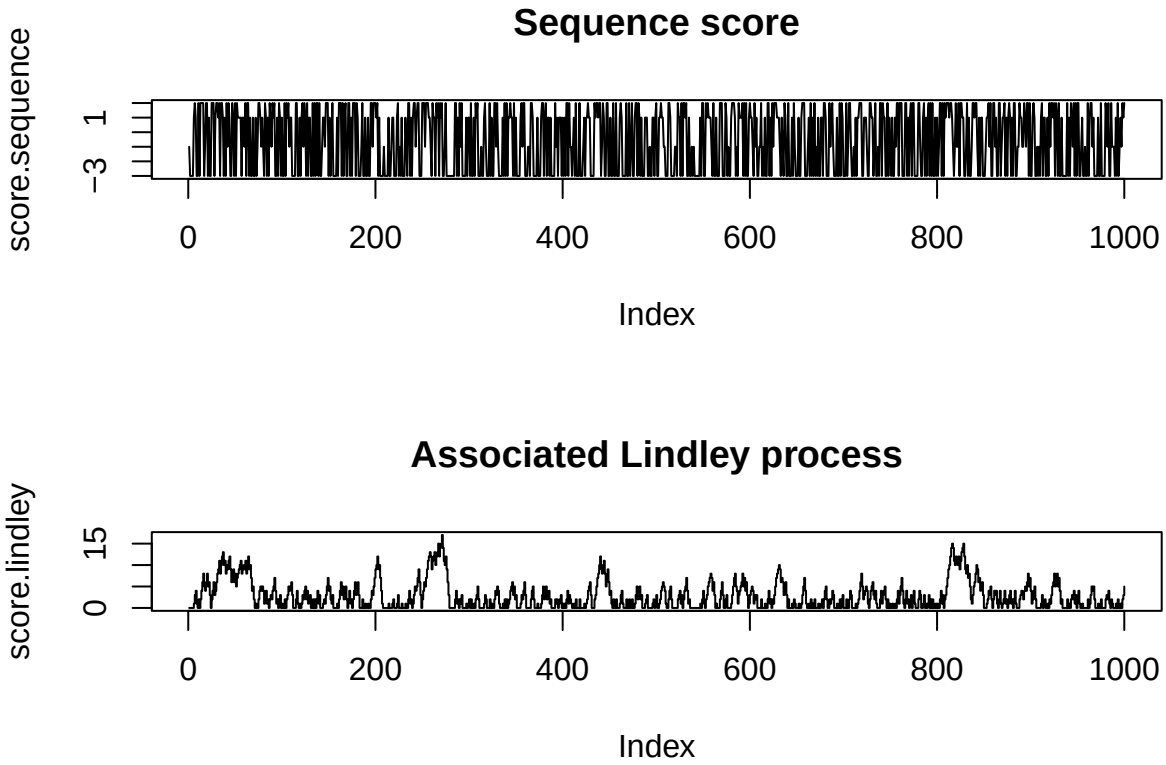
Case of I.I.D. integer score sequence

Generating an I.I.D. score sequence for this example

```
score.values <- -3:2                                        # Possible score values
score.probability <- c(0.4, 0.0, 0.1, 0.0, 0.2, 0.3)        # Associated score probabilities
score.expectation <- sum(score.values * score.probability) # Score expectation
print(score.expectation)
#> [1] -0.5
n <- 1000                                                  # sequence size
score.sequence <- sample(score.values, n, replace = TRUE, prob = score.probability)
```

Graphical representation via the Lindley process

```
score.lindley <- lindley(score.sequence)
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(2,1))
plot(score.sequence, typ = 'l', main = "Sequence score")
plot(score.lindley, typ = 's', main = "Associated Lindley process")
```




```
par(oldpar)
```

Finding the maximal sum subsequence and all strictly positive sum subsequences

```
segments.sequence <- localScoreC(score.sequence)
localScore.sequence <- segments.sequence$localScore # Local score and position of the segment
print(localScore.sequence)
#> value begin   end
#>    17   250   271
subLocalScore.sequence <- segments.sequence$suboptimalSegmentScores # suboptimal local scores and posit
print(head(subLocalScore.sequence))
#>   value begin end
#> 1     4     6   8
#> 2     2    11  11
#> 3     8    13  16
#> 4    13    25  37
#> 5     1    72  72
#> 6     5    75  77
```

Calculating p-value of local score

```
# Exact p-value (computational limitation, see help(daudin))
daudin(localScore.sequence["value"],
       sequence_length = n,
       score_probabilities = score.probability,
       sequence_min = min(score.values),
       sequence_max = max(score.values))
#> [1] 0.8932085
# Karlin and Dembo approximation (n big)
karlin(localScore.sequence["value"],
       sequence_length = n,
       score_probabilities = score.probability,
       sequence_min = min(score.values),
       sequence_max = max(score.values))
#> [1] 0.8633377
# Improved Karlin and Dembo approximation (computational limitation, see help(mcc))
mcc(localScore.sequence["value"],
       sequence_length = n,
       score_probabilities = score.probability,
       sequence_min = min(score.values),
       sequence_max = max(score.values))
#> [1] 0.8656326
```

Case of markov integer score sequence

Generating a markov score sequence for this example

```
transitionMatrix <- matrix(c(0.2, 0.3, 0.5,
                             0.3, 0.4, 0.3,
                             0.2, 0.4, 0.4), byrow = TRUE, ncol = 3)
score.values <- c(-3, -1, 2)
row.names(transitionMatrix) <- score.values
score.stationary.distribution <- stationary_distribution(transitionMatrix)
score.expectation <- sum(score.values * score.stationary.distribution)
print(score.expectation)
#> [1] -0.3168317
# Generating example markov sequence of score:
n <- 10000
score.sequence <- transmatrix2sequence(matrix = transitionMatrix,
                                       length = n,
                                       score = score.values)
head(score.sequence)
#> [1]  2  2  2  2 -1 -1
```

Finding the maximal sum subsequence and all strictly positive sum subsequences

```
segments.sequence <- localScoreC(score.sequence)
localScore.sequence <- segments.sequence$localScore # Local score and position of the segment
```

```r
print(localScore.sequence)
#> value begin   end
#>    45  2886  2949
subLocalScore.sequence <- segments.sequence$suboptimalSegmentScores # suboptimal local scores and posit
print(head(subLocalScore.sequence))
#>   value begin end
#> 1     8     1   4
#> 2    15    11  19
#> 3     4    43  44
#> 4     2    50  50
#> 5     6    56  61
#> 6     2    67  67
```

Calculating p-value of local score

```r
# Exact p-value (computational limitation, see help(exact_mc))
exact_mc(local_score = localScore.sequence["value"],
         m = transitionMatrix,
         sequence_length = n,
         prob0 = score.stationary.distribution)
#> [1] 0.212366
```

Case of various dependent structure sequence via Monte-Carlo approach

A monte-Carlo function is implemented in this package and allows to compute the p-value of the local score in case of diverse model generating the score sequence. We present an example of use here.

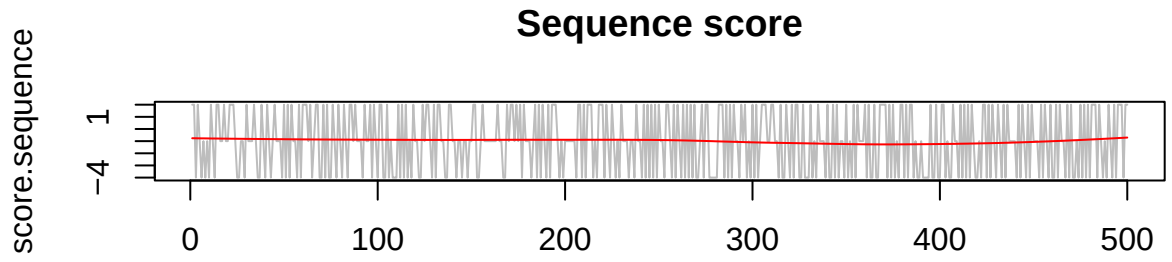
Generating a complex dependent structure score sequence for this example

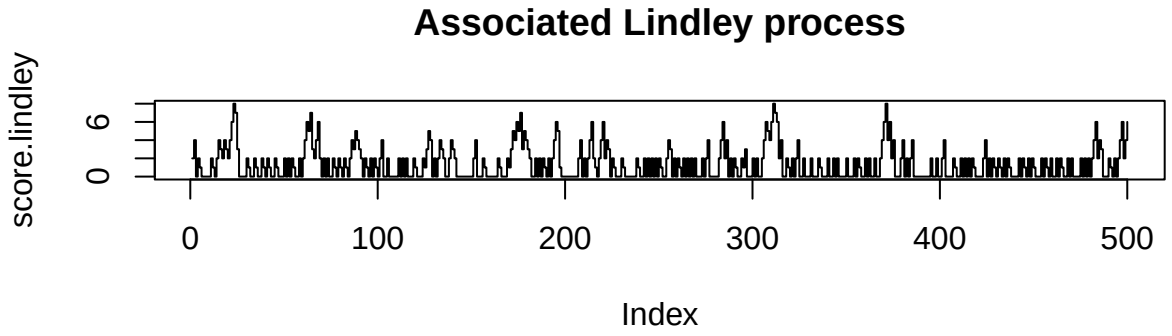
```r
# Some sort of drifting markov sequence generator
sequence.generator <- function(n, P1, P2, score_values) {
  nstate <- dim(P1)[1]
  sequence.sim <- rep(NA, n)
  sequence.sim[1] <- sample(1:nstate, 1, prob = stationary_distribution(P1))
  for (i in 2:n) {
    P <- (n - i) / (n - 1) * P1 + (i - 1) / (n - 1) * P2
    sequence.sim[i] <- sample(1:nstate, 1, prob = P[sequence.sim[i - 1],])
  }
  return(score_values[sequence.sim])
}

P1 <- matrix(c(0.2, 0.3, 0.5,
               0.3, 0.4, 0.3,
               0.2, 0.4, 0.4), byrow = TRUE, ncol = 3)
P2 <- matrix(c(0.2, 0.1, 0.7,
               0.6, 0.4, 0.0,
               0.8, 0.2, 0.2), byrow = TRUE, ncol = 3)
score.values <- c(-4, -1, 2)
n <- 500
```

```r
score.sequence <- sequence.generator(n, P1, P2, score.values)
head(score.sequence)
#> [1]  2  2 -4  2 -1 -4
mean(score.sequence) # Expected to be strictly negative
#> [1] -1
```

Graphical representation via the Lindley process

```r
score.lindley <- lindley(score.sequence)
x <- 1:n
lw1 <- loess(score.sequence ~ x)
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(2,1))
{{{plot(score.sequence, typ = 'l', col = "grey", main = "Sequence score")
lines(x, lw1$fitted[x], col = "red")}}}
plot(score.lindley, typ = 's', main = "Associated Lindley process")
```






```r
par(oldpar)
```

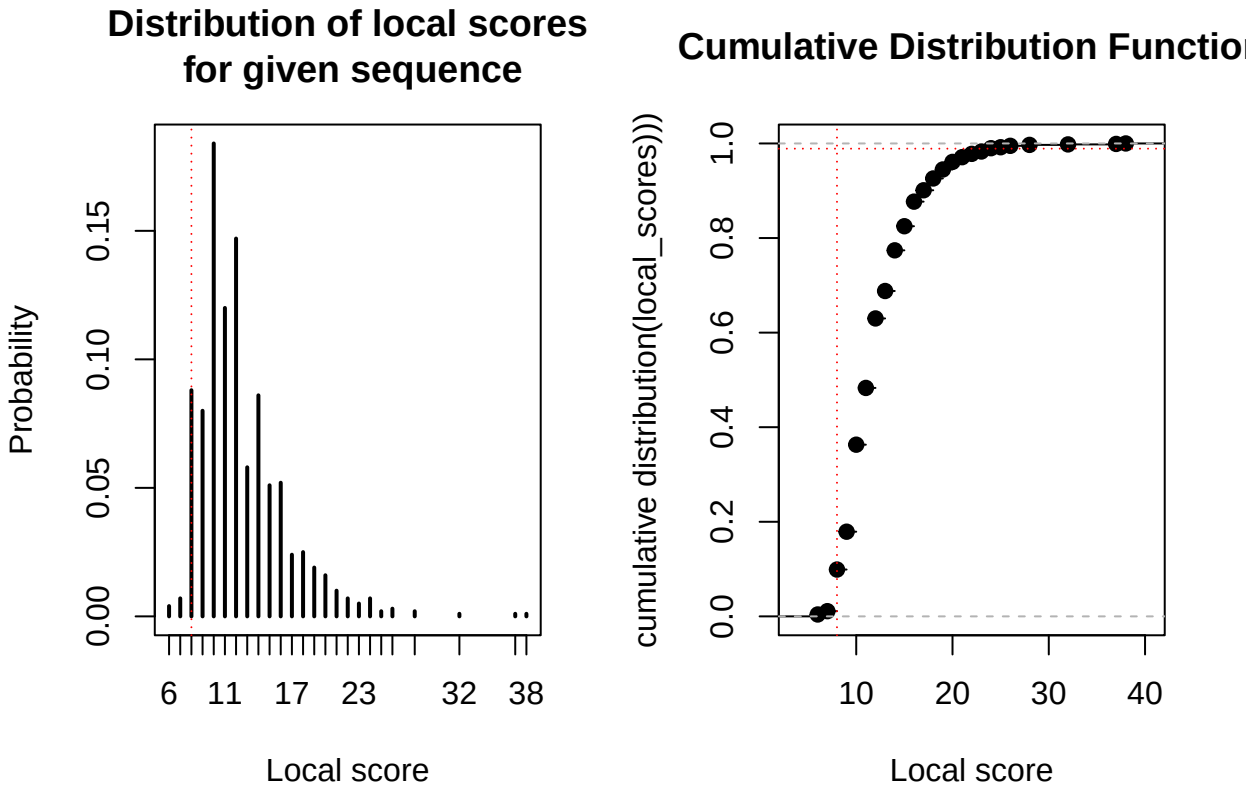
Finding the maximal sum subsequence and all strictly positive sum subsequences

```r
segments.sequence <- localScoreC(score.sequence)
localScore.sequence <- segments.sequence$localScore # Local score and position of the segment
print(localScore.sequence)
```

```
#> value begin   end
#>     8    14    23
subLocalScore.sequence <- segments.sequence$suboptimalSegmentScores # suboptimal local scores and posit
print(head(subLocalScore.sequence))
#>   value begin end
#> 1     4     1   2
#> 2     2     4   4
#> 3     2    11  11
#> 4     8    14  23
#> 5     2    30  30
#> 6     2    34  34
```

Calculating p-value of local score by Monte Carlo simulation

```
p.value <- monteCarlo(local_score = localScore.sequence["value"],
           FUN = function(n, P1, P2, score_values) {
             return(sequence.generator(n, P1, P2, score_values))
             },
           n = n, P1 = P1, P2 = P2, score_values = score.values,  #generative function parameters
           plot = TRUE,
           numSim = 1000 # low number here to compute the vignette in a "short" time
          )
```




```
print(p.value)
#> p_value
#>   0.989
```

Case of ith excursion (sequential order)

Below is a markov example, but also available for i.i.d. case.

Generating a markov score sequence for this example

```
transitionMatrix <- matrix(c(0.2, 0.3, 0.5,
                             0.3, 0.4, 0.3,
                             0.2, 0.4, 0.4), byrow = TRUE, ncol = 3)
score.values <- c(-3, -1, 2)
row.names(transitionMatrix) <- score.values
score.stationary.distribution <- stationary_distribution(transitionMatrix)
score.expectation <- sum(score.values * score.stationary.distribution)
print(score.expectation)
#> [1] -0.3168317
# Generating example markov sequence of score:
n <- 10000
score.sequence <- transmatrix2sequence(matrix = transitionMatrix,
                                       length = n,
                                       score = score.values)
```

Finding the maximal sum subsequence and all strictly positive sum subsequences

```
segments.sequence <- localScoreC(score.sequence)
subLocalScore.sequence <- segments.sequence$suboptimalSegmentScores
```

Calculating p-value of ith excursion

```
iVisualExcursion <- 1 #first strictly positive excursion
iSegment <- subLocalScore.sequence[iVisualExcursion,]
print(iSegment)
#>   value begin end
#> 1     6     2   4
# determining i in the sense of Karlin and Dembo (1990) (i >= iVisualExcursion)
i <- sum(segments.sequence$RecordTime <= iSegment$begin)
print(i)
#> [1] 2
proba_theoretical_ith_excursion_markov(iSegment$value, score.values, transitionMatrix, score.values, i =
#> $proba_q_i_geq_a
#> [1] 0.119621
#>
#> $P_alpha
#>        -3        -1         2
#> 0.1226966 0.1192796 0.1180610
```

Local Score computation methods

First defined in *Karlin and Altschul (1990)*, it represents the value of the highest scoring segment in a sequence of scores (formula H_n below). It corresponds to the highest cumulated subsequence sum amongst all possible subsequences (independent of length). For the score to be relevant, the expectation of the sequence should be negative. Thus, for a sequence of interest, the possible score of sequence components should be positive and negative for the local score have to be meaningful.

$$H_n = max_{1 \leq i \leq j \leq n} \sum_{l=i}^{j} X_l$$

A first example: function "localScoreC()''

Let us assume a score function taking its values in [-2, -1, 0, 1, 2]. A sample score sequence of length 100 could be

```
library(localScore)
help(localScore)
ls(pos = 2)
#>  [1] "Aeso"
#>  [2] "CharSequence2ScoreSequence"
#>  [3] "CharSequences2ScoreSequences"
#>  [4] "HydroScore"
#>  [5] "LongSeq"
#>  [6] "MidSeq"
#>  [7] "MySeqList"
#>  [8] "RealScores2IntegerScores"
#>  [9] "SJSyndrome"
#> [10] "SJSyndrome.data"
#> [11] "Seq1093"
#> [12] "Seq219"
#> [13] "Seq31"
#> [14] "SeqListSCOPe"
#> [15] "ShortSeq"
#> [16] "aeso.data"
#> [17] "automatic_analysis"
#> [18] "daudin"
#> [19] "dico"
#> [20] "exact_mc"
#> [21] "karlin"
#> [22] "karlinMonteCarlo"
#> [23] "karlinMonteCarlo_double"
#> [24] "karlin_parameters"
#> [25] "lindley"
#> [26] "loadMatrixFromFile"
#> [27] "loadScoreFromFile"
#> [28] "localScoreC"
#> [29] "localScoreC_double"
#> [30] "localScoreC_int"
#> [31] "maxPartialSumd"
#> [32] "mcc"
#> [33] "monteCarlo"
```

```
#> [34] "monteCarlo_double"
#> [35] "proba_theoretical_first_excursion_iid"
#> [36] "proba_theoretical_ith_excursion_iid"
#> [37] "proba_theoretical_ith_excursion_markov"
#> [38] "recordTimes"
#> [39] "scoreSequences2probabilityVector"
#> [40] "sequences2transmatrix"
#> [41] "stationary_distribution"
#> [42] "transmatrix2sequence"
mySeq <- sample(-2:2, 100, replace = TRUE, prob = c(0.5, 0.3, 0.05, 0.1, 0.05))
mySeq
#>   [1] -1 -2 -2 -2 -2 -1 -1 -1  2 -2 -1  1 -2  1 -2 -1 -1 -2 -1  1 -2 -2  1 -2 -1
#>  [26] -1 -1 -2 -2 -1 -2 -2 -2 -1 -1 -1  1 -1  0  1 -2 -1 -1  1 -2  1  0 -2 -1  1
#>  [51] -2 -2 -1 -2 -2 -2 -2 -1 -2 -2  1 -1 -1 -1  2 -2 -1 -2 -2  0 -1 -1 -2  1 -2
#>  [76]  1  2 -2 -1 -1 -2 -2 -2 -1 -1 -2 -2  1 -1 -2 -2 -2 -2 -1 -2 -2 -1 -1 -2 -1
scoreSequenceExpectation <- sum(c(-2:2)*c(0.5, 0.3, 0.05, 0.1, 0.05))
scoreSequenceExpectation
#> [1] -1.1
localScoreC(mySeq)
#> $localScore
#> value begin   end
#>     3    76    77
#>
#> $suboptimalSegmentScores
#>    value begin end
#> 1      2     9   9
#> 2      1    12  12
#> 3      1    14  14
#> 4      1    20  20
#> 5      1    23  23
#> 6      1    37  37
#> 7      1    40  40
#> 8      1    44  44
#> 9      1    46  46
#> 10     1    50  50
#> 11     1    61  61
#> 12     2    65  65
#> 13     1    74  74
#> 14     3    76  77
#> 15     1    88  88
#>
#> $RecordTime
#>  [1]   0   1   2   3   4   5   6   7   8  11  13  15  16  17  18  19  21  22  24
#> [20]  25  26  27  28  29  30  31  32  33  34  35  36  41  42  43  45  48  49  51
#> [39]  52  53  54  55  56  57  58  59  60  63  64  67  68  69  71  72  73  75  80
#> [58]  81  82  83  84  85  86  87  90  91  92  93  94  95  96  97  98  99 100
```

The result is a maximum score and for which subsequence it has been found: the starting position "[" and the end of it ("]"). It also yields all other subsequences with a score equal or less and their positions in the `$suboptimalSegmentScores` matrix. Note that those subsequences do not have common positions. "Stopping Times" are local minima in the cumulated sum of the sequence and correspond to the beginning of excursions (potential segments of interest). The "end" of the segment is the position realizing the (sub)-maximum of the segments of the sequence.

Another example with missing score values.

```
library(localScore)
mySeq <- sample(c(-3,2,0,1,5), 100, replace = TRUE, prob = c(0.5, 0.3, 0.05, 0.1, 0.05))
head(mySeq)
#> [1] -3 -3  2 -3  0  2
localScoreC(mySeq)
#> $localScore
#> value begin   end
#>    18    66    74
#>
#> $suboptimalSegmentScores
#>    value begin end
#> 1      2     3   3
#> 2      4     6   7
#> 3      1    14  14
#> 4     10    19  24
#> 5      1    40  40
#> 6      2    43  43
#> 7      6    45  52
#> 8      2    55  55
#> 9      1    57  57
#> 10     7    61  62
#> 11    18    66  74
#> 12     6    95  99
#>
#> $RecordTime
#>  [1]  0  1  2  4  9 10 12 13 15 16 17 34 35 36 37 38 39 41 42 44 56 58 59 60 65
#> [26] 94
```

Remark: in the case where the local score is realized by more that one segment with the same starting position, the shortest one is chosen. In the example below, the local score of the `seq2` sequence is 6 and there are two segments starting at position 5 which realized it: the first finished at position 6, and the second finished at position 8. In this case, the function `localScoreC` retains the shortest segment.

```
seq1 <- c(1,-2,3,1,-1,2)
seq2 <- c(1,-2,3,1,-1,2,-1,1)
localScoreC(seq1)
#> $localScore
#> value begin   end
#>     5     3     6
#>
#> $suboptimalSegmentScores
#>   value begin end
#> 1     1     1   1
#> 2     5     3   6
#>
#> $RecordTime
#> [1] 0 2
localScoreC(seq2)
#> $localScore
#> value begin   end
#>     5     3     6
#>
```

```
#> $suboptimalSegmentScores
#>   value begin end
#> 1     1     1   1
#> 2     5     3   6
#>
#> $RecordTime
#> [1] 0 2
```

Example with real scores

```
score_reels <- c(-1, -0.5, 0, 0.5, 1)
proba_score_reels <- c(0.2, 0.3, 0.1, 0.2, 0.2)
sample_from_model <- function(score.sple, proba.sple, length.sple) {
  sample(score.sple,
    size = length.sple, prob = proba.sple, replace = TRUE
  )
}
seq.essai <- sample_from_model(score.sple = score_reels, proba.sple = proba_score_reels,
                               length.sple = 10)
localScoreC(seq.essai)
#> $localScore
#> value begin   end
#>     2     8     9
#>
#> $suboptimalSegmentScores
#>   value begin end
#> 1     1     1   5   5
#> 2     2     2   8   9
#>
#> $RecordTime
#> [1] 0 4 7
```

Example of alphabetical sequence associated to a scoring function

```
# Loading a fasta protein
data(Seq219)
Seq219
#> [1] "MSGLSGPPARRGPFPLALLLLFLLGPRLVLAISFHLPINSRKCLREEIHKDLLVTGAYEISDQSGGAGGLRSHLKITDSAGHILYSKEDATKGKF
# or using your own fasta sequence
#MySeqAA_P49755 <- as.character(read.table(file="P49755.fasta",skip=1)[,1])
#MySeqAA_P49755
# Loading a scoring function
data(HydroScore)
?HydroScore
# or using your own scoring function
# HydroScoreKyte<- loadScoreFromFile("Kyte1982.txt")
# Transforming the amino acid sequence into a score sequence with the score function
# in HydroScore file
SeqScore_P49755 <- CharSequence2ScoreSequence(Seq219, HydroScore)
```

```
head(SeqScore_P49755)
#> [1]  2 -1  0  4 -1  0
length(SeqScore_P49755)
#> [1] 219
# Computing the local score
localScoreC(SeqScore_P49755)
#> $localScore
#> value begin   end
#>    52    14    38
#>
#> $suboptimalSegmentScores
#>    value begin end
#> 1      5     1   4
#> 2      2     9   9
#> 3     52    14  38
#> 4     17   121 124
#> 5      7   138 139
#> 6      4   144 144
#> 7      4   147 147
#> 8      4   149 149
#> 9      4   151 151
#> 10     4   154 154
#> 11     4   157 157
#> 12     9   161 162
#> 13     2   175 175
#> 14    43   186 208
#>
#> $RecordTime
#>  [1]   0  10  11  13 120 136 137 143 146 152 153 156 159 160 170 171 172 173 174
#> [20] 176 177 178 179 180 181 182 183 184 185
```

Note that the scoring function (here `HydroScore`) could be read from a dedicated file using 'loadScoreFromFile()"'. See Section "File format" for more details.

p-Value computation methods

There are different methods available to establish the statistical significance, also called p-value, of the local score depending on the length of the sequence and the score expectation. This value describes the probability to encounter a given local score for a given score distribution or higher for a given sequence length. Therefore it allows to determinate if the local score in question is significant or could have been obtained by chance. Since the second version of this package, two probabilistic models for the sequence are available: - Identically and Independently Distributed Variables model (I.I.D.) - Markovian model

For an Identically and Independently Distributed Variables model (I.I.D.), the main function of the packages to calculate the p-value of the local score are: `daudin()`, `mcc()`, `karlin()`, `monteCarlo()` and `KarlinMonteCarlo()`, each of them correspond to diverse probability methods found in the literature. In the Markovian model case, the main functions are `exact_mc()` and `monteCarlo()`. The following sections illustrate and detail the context and use of these functions. In the I.I.D. case, note than we advise to use the exact method `daudin()` if possible, then `mcc()`, then `karlin()`. In any case, a more computational intensive Monte Carlo approach is always possible (`monteCarlo()` and `KarlinMonteCarlo()`), and even allow to take into account more complex sequence models using specialized random generation functions.

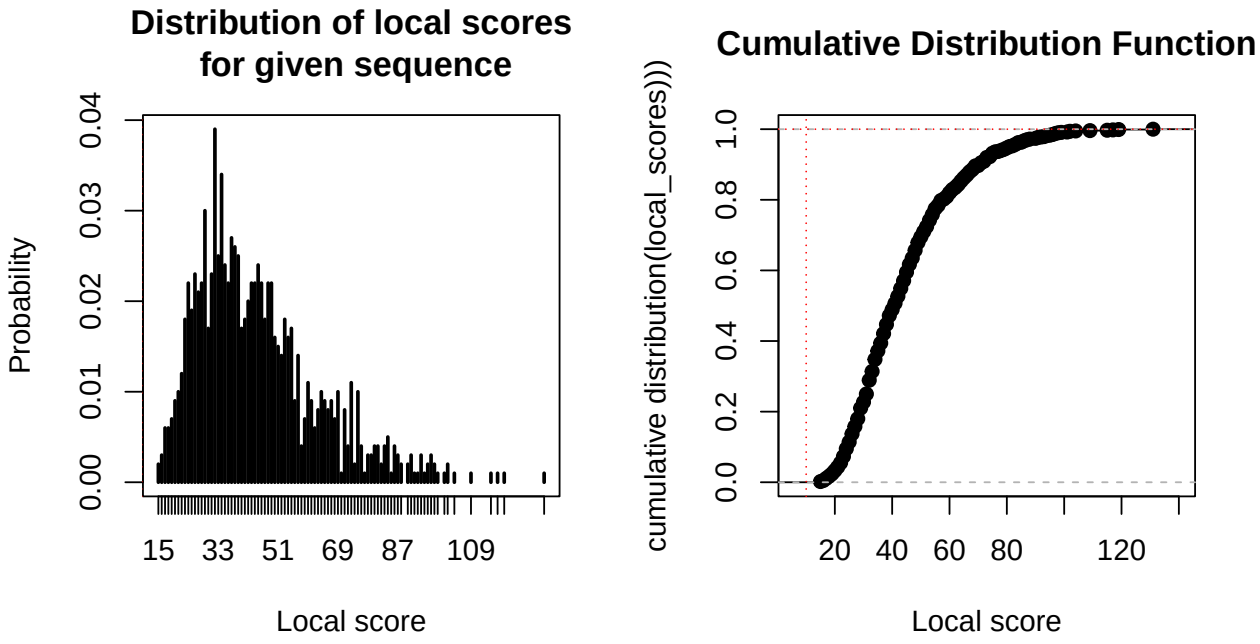
Simulating computation: functions "monteCarlo()"

The function `monteCarlo()` simulates a number of score sequences similar (same distribution) to the one having yielded the local score in question. Therefore, it requires a function that produces such sequences and the parameters used by this function. See the help page and the following example to use the empirical distribution of a given sequence, but any other function, such as `rbinom()` or custom functions, are valid too.

In the following example we search the probability to obtain a local score of 10 in the sequence we created in the previous section and which serves as a blueprint for the score sequences to produce. The return value is the p-value of the local score for the given score sequence. A plot of the distribution of all local scores simulated and the cumulative distribution function are displayed. These plots can be hidden by setting the argument `plot` to `FALSE`. The number of sequences simulated in our example is 1000, a default value, and can be changed by setting the argument "numSim" to an appropriate value.

Note that the cumulative distribution function plot indicates $P(LocalScore \leq \cdot)$ and so the corresponding p-value equals 1 minus the cumulative distribution function value.

```
monteCarlo(local_score = 10, FUN = function(x) {
  return(sample(x = x, size = length(x), replace = TRUE))
}, x = SeqScore_P49755)
```

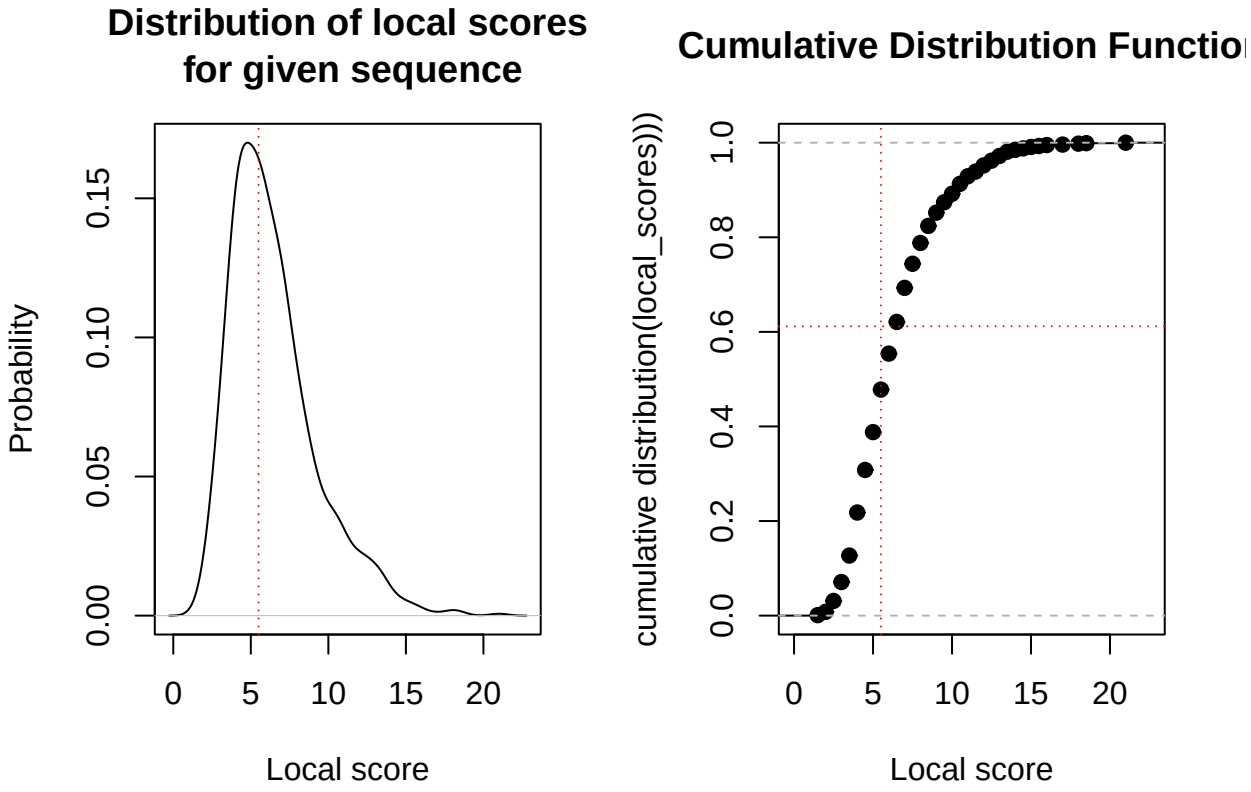



```
#> p_value
#>       1
```

The use of this method depends of computing power, number of simulations, implementation of the simulating function and the length of the sequence. For long sequences, you may prefer the next method which combines simulating and approximated methods and called `KarlinMonteCarlo()` (see next section).

```
# Example
score_reels <- c(-1, -0.5, 0, 0.5, 1)
proba_score_reels <- c(0.2, 0.3, 0.1, 0.2, 0.2)
sample_from_model <- function(score.sple, proba.sple, length.sple) {
```

```
  sample(score.sple,
    size = length.sple, prob = proba.sple, replace = TRUE
  )
}
monteCarlo(5.5,
  FUN = sample_from_model, plot = TRUE, score.sple = score_reels, proba.sple = proba_score_reels,
  length.sple = 100, numSim = 1000
)
```



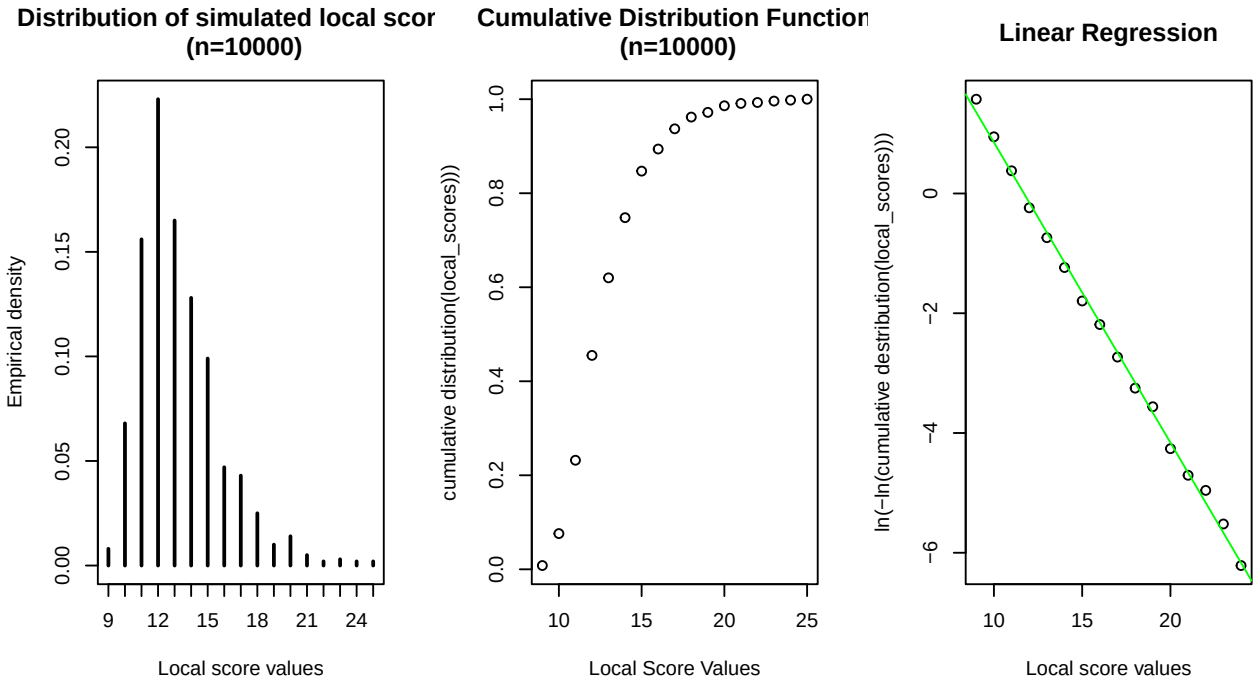

```
#> p_value
#>   0.612
```

A mixed method: functions "karlinMonteCarlo()"

The function `karlinMonteCarlo()` also uses a function supplied by the user to do simulations. However, it does not deduce directly the p-value from the cumulative distribution function. However, this function is used to estimate the parameters of the Gumbel distribution of *Karlin and al.* approximation. Thus, it is suited for long sequences. Note: `simulated_sequence_length` is the length of the simulated sequences. This value must correspond to the length of sequences yielded by FUN. The value of `n` is the sequence length for which we want to compute the p-value. If `n` is too large function `MonteCarlo()` could be too much time consuming. Using`karlinMonteCarlo()` with a smaller sequence length `simulated_sequence_length` allows to extract the parameters and then to apply them to the sequence value `n`.

```
fu <- function(n, size, prob, shift) {
  rbinom(size = size, n = n, prob = prob) + shift
}
karlinMonteCarlo(12,
```

```
  FUN = fu, n = 10000, size = 8, prob = 0.2, shift = -2,
  sequence_length = 1000000, simulated_sequence_length = 10000
)
```

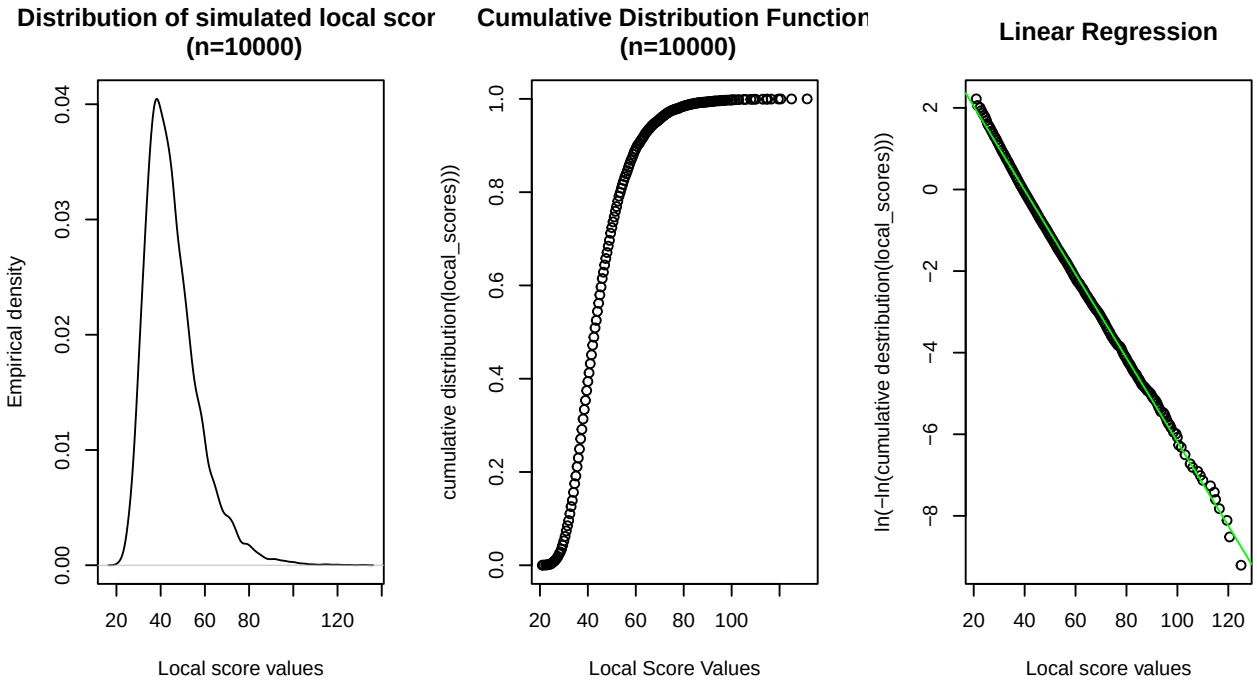



```
#> $p_value
#> [1] 1
#>
#> $`K*`
#> [1] 0.03534759
#>
#> $lambda
#> [1] 0.5014632
```

If not specified otherwise, the function produces three graphes, two of them like the function `monteCarlo()` and the third a representation of $\ln(-\ln(cf))$, with cf being the cumulated function, showing the linear regression in green color providing the parameters for the Gumbel distribution K^* and λ.

Example for real scores:

```
score_reels <- c(-1.5, -0.5, 0, 0.5, 1.5)
proba_score_reels <- c(0.2, 0.3, 0.1, 0.2, 0.2)
fu <- function(score.sple, proba.sple, length.sple) {
  sample(score.sple,
    size = length.sple, prob = proba.sple, replace = TRUE
  )
}
karlinMonteCarlo(85.5,
  FUN = fu, score.sple = score_reels, proba.sple = proba_score_reels,
                length.sple = 10000, numSim = 10000, sequence_length = 100000,
  simulated_sequence_length = 10000
)
```




```
#> $p_value
#> [1] 0.09622426
#>
#> $`K*`
#> [1] 0.00600362
#>
#> $lambda
#> [1] 0.1028218
```

Exact method for integer scores: function "daudin()''

The exact method calculates the p-value exploiting the fact of the "stopped" Lindley process, stopped at value the local score is a Markov process. Therefore, an exact p-value can be retrieved. The complexity of matrix multiplication involved being $> O(n^2)$, the method can be unsuited for sequences of either great length, $n \geq 10^4$ for example could take too much time for computation, or dispersed scores. Note that the exact method requires integer scores.

```
daudin(
  local_score = 15, sequence_length = 500, score_probabilities =
    c(0.2, 0.3, 0.3, 0.0, 0.1, 0.1), sequence_min = -3, sequence_max = 2
)
#> [1] 0.0004119255
```

This function is based on: *S. Mercier and J.J. Daudin 2001: "Exact distribution for the local score of one i.i.d. random sequence"*

How to use the exact method for real scores

The exact method requires integer score to be used. For real scores, a homothetic transformation can be considered to be able to use the exact method. This transformation is theoretically validated to still

provide an exact probability. The only existing drawback of the homothetic transformation is that the computation time is increasing. We can check in the following example that the p-value computation using the exact method with an homothetic transformation corresponds, approximately, to the real local score p-value computed with the Monte Carlo method.

```
score_reels <- c(-1, -0.5, 0, 0.5, 1)
proba_score_reels <- c(0.2, 0.3, 0.1, 0.2, 0.2)
sample_from_model <- function(score.sple, proba.sple, length.sple) {
  sample(score.sple,
    size = length.sple, prob = proba.sple, replace = TRUE
  )
}
seq.essai <- sample_from_model(score.sple = score_reels, proba.sple = proba_score_reels,
                               length.sple = 100)
localScoreC(seq.essai)
#> $localScore
#> value begin   end
#>     5    44    72
#>
#> $suboptimalSegmentScores
#>    value begin end
#> 1    0.5     2   2
#> 2    1.0    10  10
#> 3    1.0    13  14
#> 4    2.5    19  32
#> 5    0.5    37  37
#> 6    1.0    39  40
#> 7    5.0    44  72
#> 8    1.0    83  83
#> 9    2.0    89  90
#> 10   1.0    95  95
#> 11   1.0   100 100
#>
#> $RecordTime
#>  [1]  0  1  6  7  8  9 18 36 38 42 43 87 88 93 94 98
C <- 10 # homothetic coefficient
localScoreC(as.integer(C*seq.essai))
#> $localScore
#> value begin   end
#>    50    44    72
#>
#> $suboptimalSegmentScores
#>    value begin end
#> 1      5     2   2
#> 2     10    10  10
#> 3     10    13  14
#> 4     25    19  32
#> 5      5    37  37
#> 6     10    39  40
#> 7     50    44  72
#> 8     10    83  83
#> 9     20    89  90
#> 10    10    95  95
#> 11    10   100 100
```

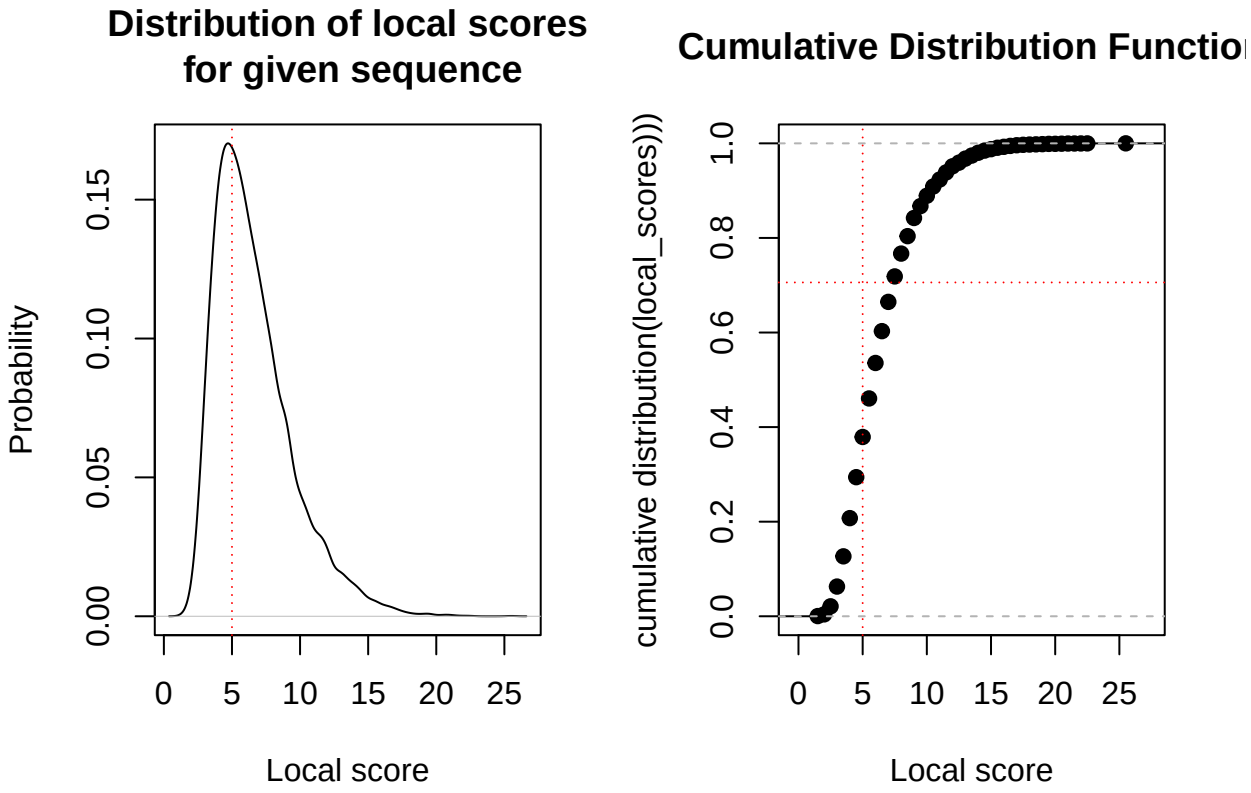
```
#>
#> $RecordTime
#>  [1]  0  1  6  7  8  9 18 36 38 42 43 87 88 93 94 98
```

We can check that the local score of the sequence which has been homothetically transformed is multiplied by the same coefficient. We are going to compute the p-value. For this we need to create the integer score vector and its corresponding distribution from the ones dedicated to real scores:

```
RealScores2IntegerScores(score_reels, proba_score_reels, coef = C)
#> $ExtendedIntegerScore
#>  [1] -10  -9  -8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8
#> [20]   9  10
#>
#> $ProbExtendedIntegerScore
#> -10  -9  -8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8   9
#> 0.2 0.0 0.0 0.0 0.0 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.2 0.0 0.0 0.0 0.0
#>  10
#> 0.2
M.s.r <- RealScores2IntegerScores(score_reels, proba_score_reels, coef = C)$ExtendedIntegerScore
M.s.prob <- RealScores2IntegerScores(score_reels, proba_score_reels, coef = C)$ProbExtendedIntegerScore
M.SL <- localScoreC(as.integer(C * seq.essai))$localScore[1]
M.SL
#> value
#>    50
pval.E <- daudin(
  local_score = M.SL, sequence_length = length(seq.essai), score_probabilities = M.s.prob,
  sequence_min = -10, sequence_max = 10
)
pval.E
#> [1] 0.7044225
```

Let us compare the result of the exact method with homothetical transformation with MonteCarlo method for the initial sequence and real scores

```
SL.real <- localScoreC(seq.essai)$localScore[1]
SL.real
#> value
#>     5
pval.MC <- monteCarlo(
  local_score = SL.real, FUN = sample_from_model,
  score.sple = score_reels, proba.sple = proba_score_reels,
  length.sple = length(seq.essai), plot = TRUE, numSim = 10000
)
```




```
pval.MC
#> p_value
#>  0.7059
```

We can check that the two probabilities are close: 0.7044225 for the exact method and 0.7059 for Monte Carlo's one. The difference comes from the fact that the Monte Carlo method produces an approximation.

Approximate method of Karlin *et al.*: function "karlin()''

The method of Karlin uses the local score's distinctive cumulative distribution following a law of Gumbel to approximate the p-value. It is suited for large and very large sequences as the approximation is asymptotic with the sequence length and so more accurate for large sequences ; and secondly very large sequence case can be too much time and space consuming for the exact method whereas Karlin *et al.* method does not depend to the sequence length for the computational criteria. The average score must be non positive.

```
score.v <- -2:1
score.p <- c(0.3, 0.2, 0.2, 0.3)
sum(score.v*score.p)
#> [1] -0.5
karlin(
  local_score = 14, sequence_length = 100000, sequence_min = -2, sequence_max = 1,
  score_probabilities = c(0.3, 0.2, 0.2, 0.3)
)
#> [1] 0.4171019
karlin(
  local_score = 14, sequence_length = 1000, sequence_min = -2, sequence_max = 1,
  score_probabilities = c(0.3, 0.2, 0.2, 0.3)
```

```
)
#> [1] 0.00538289
```

We verify here that the same local score value 14 is more usual for a longer sequence.

```
# With missing score values
karlin(
  local_score = 14, sequence_length = 1000, sequence_min = -3, sequence_max = 1,
  score_probabilities = c(0.3, 0.2, 0.0, 0.2, 0.3)
)
#> [1] 0.001034394
```

This function is based on: Karlin *et al.* 1990: *"Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes"*

The function `Karlin()` is dedicated for integer scores. For real ones the homothetic solution presented for the exact method can also be applied.

An improved approximate method: function "mcc()''

The function `mcc()` uses an improved version of the Karlin's method to calculate the *p*-value. It is suited for sequences of length upper or equal to several hundreds. Let us compare the three methods on the same case.

```
mcc(
  local_score = 14, sequence_length = 1000, sequence_min = -3, sequence_max = 2,
  score_probabilities = c(0.2, 0.3, 0.3, 0.0, 0.1, 0.1)
)
#> [1] 0.002011779

daudin(
  local_score = 14, sequence_length = 1000, score_probabilities =
    c(0.2, 0.3, 0.3, 0.0, 0.1, 0.1), sequence_min = -3, sequence_max = 2
)
#> [1] 0.001988438
karlin(
  local_score = 14, sequence_length = 1000, sequence_min = -3, sequence_max = 2,
  score_probabilities = c(0.2, 0.3, 0.3, 0.0, 0.1, 0.1)
)
#> [1] 0.002007505
```

We can observe than the improved approximation method with `mcc()` gives a *p*-value equal to 0.0020118 which is more accurate than the one of `karlin()` function equal to 0.0020075 compared to the exact method which computation equal to 0.0019884.

This function is based on the work of S. Mercier, D. Cellier and D. Charlot 2003 *"An improved approximation for assessing the statistical significance of molecular sequence features''.*

An automatic method: function "automatic_analysis()''

This function is meant as a support for the inexperienced user. Since the use of methods for *p*-value requires some understanding on how these methods work, this function automatically selects an adequate methods based on the sequence given.

There are different use-case scenarios for this function. One can just put a sequence and the model (Markov chains or identically and independently distributed).

```
automatic_analysis(sequences = list("x1" = c(1,-2,2,3,-2, 3, -3, -3, -2)), model = "iid")
#> $x1
#> $x1$`p-value`
#> [1] 0.4750898
#>
#> $x1$`method applied`
#> [1] "Exact Method Daudin et al"
#>
#> $x1$localScore
#> $x1$localScore$localScore
#> value begin   end
#>     6     3     6
#>
#> $x1$localScore$suboptimalSegmentScores
#>   value begin end
#> 1     1     1   1
#> 2     6     3   6
#>
#> $x1$localScore$RecordTime
#> [1] 0 2 9
```

In the upper example, the sequence is short, so the exact method is adapted. Here is another example. As the sequence is much more longer, the asymptotic approximation of Karlin *et al.* can be used. This is possible because the average score is negative. If not the `MonteCarlo` method could have been preferred by the function.

```
score <- c(-2, -1, 0, 1, 2)
proba_score <- c(0.2, 0.3, 0.1, 0.2, 0.2)
sum(score*proba_score)
#> [1] -0.1
sample_from_model <- function(score.sple, proba.sple, length.sple) {
  sample(score.sple,
    size = length.sple, prob = proba.sple, replace = TRUE
  )
}
seq.essai <- sample_from_model(score.sple = score, proba.sple = proba_score, length.sple = 5000)
MyAnalysis <- automatic_analysis(
  sequences = list("x1" = seq.essai),
  distribution = proba_score, score_extremes = c(-2, 2), model = "iid"
)$x1
MyAnalysis$"p-value"
#> [1] 0.5482626
MyAnalysis$"method applied"
#> [1] "Asymptotic Method Karlin et al"
MyAnalysis$localScore$localScore
#> value begin   end
#>    42  3400  3751
```

For real score, achieved the homothetic transformation before. If not, the results could not be correct.

```r
score_reels <- c(-1, -0.5, 0, 0.5, 1)
proba_score_reels <- c(0.2, 0.3, 0.1, 0.2, 0.2)
sample_from_model <- function(score.sple, proba.sple, length.sple) {
  sample(score.sple,
    size = length.sple, prob = proba.sple, replace = TRUE
  )
}
seq.essai <- sample_from_model(score.sple = score_reels, proba.sple = proba_score_reels, length.sple =

# Homothetie
C <- 10
RealScores2IntegerScores(score_reels,proba_score_reels, coef=C)
#> $ExtendedIntegerScore
#>  [1] -10  -9  -8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8
#> [20]   9  10
#>
#> $ProbExtendedIntegerScore
#> -10  -9  -8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8   9
#> 0.2 0.0 0.0 0.0 0.0 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.2 0.0 0.0 0.0 0.0
#>  10
#> 0.2
M.s.r <- RealScores2IntegerScores(score_reels, proba_score_reels, coef = C)$ExtendedIntegerScore
M.s.prob <- RealScores2IntegerScores(score_reels, proba_score_reels, coef = C)$ProbExtendedIntegerScore
# The analysis
MyAnalysis <- automatic_analysis(
  sequences = list("x1" = as.integer(C * seq.essai)), model = "iid",
  distribution = M.s.prob, score_extremes = range(M.s.r)
)
MyAnalysis$x1$"p-value"
#> [1] 0.7287704
MyAnalysis$x1$"method applied"
#> [1] "Exact Method Daudin et al"

# Without the homothety, the function gives a wrong result
# MyAnalysis2 <- automatic_analysis(sequences = list("x1" = seq.essai), model = "iid")
# MyAnalysis2$x1$"p-value"
# MyAnalysis2$x1$"method applied"
```

Whenever there is no distribution given, these is learned from the sequence(s) given. The sequence(s) must be passed as **named list**. If there are more than one sequence, all sequences are treated. A progress bar informs the user about the progress made. If the sequence(s) passed are not score sequences but sequence(s) of letters, a file picker dialog pops up. Here, the user can choose a file containing a mapping from letters to scores. The format is csv (view section "File Formats" for details) and allows also to provide a distribution that is loaded concurrently to the score. One can also chose not to provide sequences at all. In this case, the first dialog that pops up is for selection of a FASTA file (view section "File Formats" for details). Either way, the user has little influence on the method applied to find the p-value. One can change the argument `method_limit` to modify the threshold for the use of exact and approximating methods or supply a function for simulation methods which will result in the use of a simulation method. In this case, make sure to provide a suitable `simulated_sequence_length` argument, as for long sequences, the method `monteCarloKarlin` will be used.

Markovian model of the sequence : function `exact_mc()`

A markovian dependency of the components of the sequence can be taken into account to calculate the p-value of the local score. Note that memory usage and time computation can be too large for a high local score value and high score range, as this method computation needs to allocate a square matrix of size localScoreˆ(range(score_values)). This matrix is then exponentiated to `sequence_length`. So, be aware to only use this function for values respecting your hardware configuration. As an alternative, the `monteCarlo()` function can be used. See an example of use below. Note that the function `stationary_distribution()` calculates the stationary distribution of the markovian sequence. The probability of the first score can be specified. If not, the stationary distribution is used meaning that the markovian is supposed to be at the stationary state.

```
scoreValues <- c(-2, -1, 2)
mTransition <- matrix(c(0.2, 0.3, 0.5, 0.3, 0.4, 0.3, 0.2, 0.4, 0.4), byrow = TRUE, ncol = 3)
initialProb <- stationary_distribution(mTransition)
exact_mc(local_score = 50, m = mTransition, sequence_length = 100,
        score_values = scoreValues, prob0 = initialProb)
#> [1] 0.001486195
```

Note that if you name the transition matrix rows with the score values, you can omit the score_values parameter. Example below.

```
scoreValues <- c(-2, -1, 2)
mTransition <- matrix(c(0.2, 0.3, 0.5, 0.3, 0.4, 0.3, 0.2, 0.4, 0.4), byrow = TRUE, ncol = 3)
rownames(mTransition) <- scoreValues
initialProb <- stationary_distribution(mTransition)
exact_mc(local_score = 50, m = mTransition, sequence_length = 100)
#> [1] 0.001486195
```

Last, note the existence of the function `transmatrix2sequence()` which simulate a markovian sequence given a transition matrix and a initial score value in option. This can be useful in combination with `monteCarlo()` function. Example:

```
MyTransMat <-
  matrix(c(
    0.3, 0.1, 0.1, 0.1, 0.4, 0.2, 0.2, 0.1, 0.2, 0.3, 0.3, 0.4, 0.1, 0.1, 0.1, 0.3, 0.3, 0.1, 0.0, 0.3,
    0.1, 0.1, 0.2, 0.3, 0.3
  ), ncol = 5, byrow = TRUE)

MySeq.CM <- transmatrix2sequence(matrix = MyTransMat, length = 150, score = -2:2)
MySeq.CM
#>   [1] -1  2  2  2 -2 -1  2  0  0 -2 -2 -2  2  1 -2  2  2 -2 -2  0  1 -1 -1 -1  2
#>  [26]  1 -1 -1  2  1 -1 -1 -2 -2 -1 -1 -2  1  2  2  0 -1 -2  2  1 -1  0 -2  2  1
#>  [51] -1 -1 -1 -1  0 -1  2  2  2  1 -2 -2  0  1 -2 -2  1  2  0  1  2  1 -1  2  1
#>  [76]  2  1 -1 -2  2 -1 -1  2  2  2  2  0 -1 -1 -1  2 -1  1  0 -1  2  1  2  1  2
#> [101]  2  1 -1 -2  2  2 -2 -2  2 -1  0 -2  2  2  1 -2 -2 -1  2  0 -2  1 -1 -1  2
#> [126]  2  2  0  2  2  0  0 -2  2  0 -1 -2  2  2  1  0 -2  2  1  0 -1 -2 -2  0 -2
AA.CM <- automatic_analysis(sequences = list("x1" = MySeq.CM), model = "markov")
AA.CM
#> $x1
#> $x1$`p-value`
#> [1] 0.4748838
#>
```

```
#> $x1$`method applied`
#> [1] "Exact Method"
#>
#> $x1$localScore
#> $x1$localScore$localScore
#> value begin   end
#>    34    67   144
#>
#> $x1$localScore$suboptimalSegmentScores
#>   value begin end
#> 1     6     2   4
#> 2     5    13  17
#> 3     4    25  30
#> 4     5    38  40
#> 5     7    57  60
#> 6    34    67 144
#>
#> $x1$localScore$RecordTime
#> [1]  0  1 12 24 34 35 36 37
```

With MonteCarlo method the local score approximate p-value is also not significant.

```
Ls.CM <- AA.CM$x1$localScore[[1]][1]
monteCarlo(
  local_score = Ls.CM,
  FUN = transmatrix2sequence, matrix = MyTransMat,
  length=150, score = -2:2,
  plot = FALSE, numSim = 10000
)
#> p_value
#>  0.4048
```

Other Functions

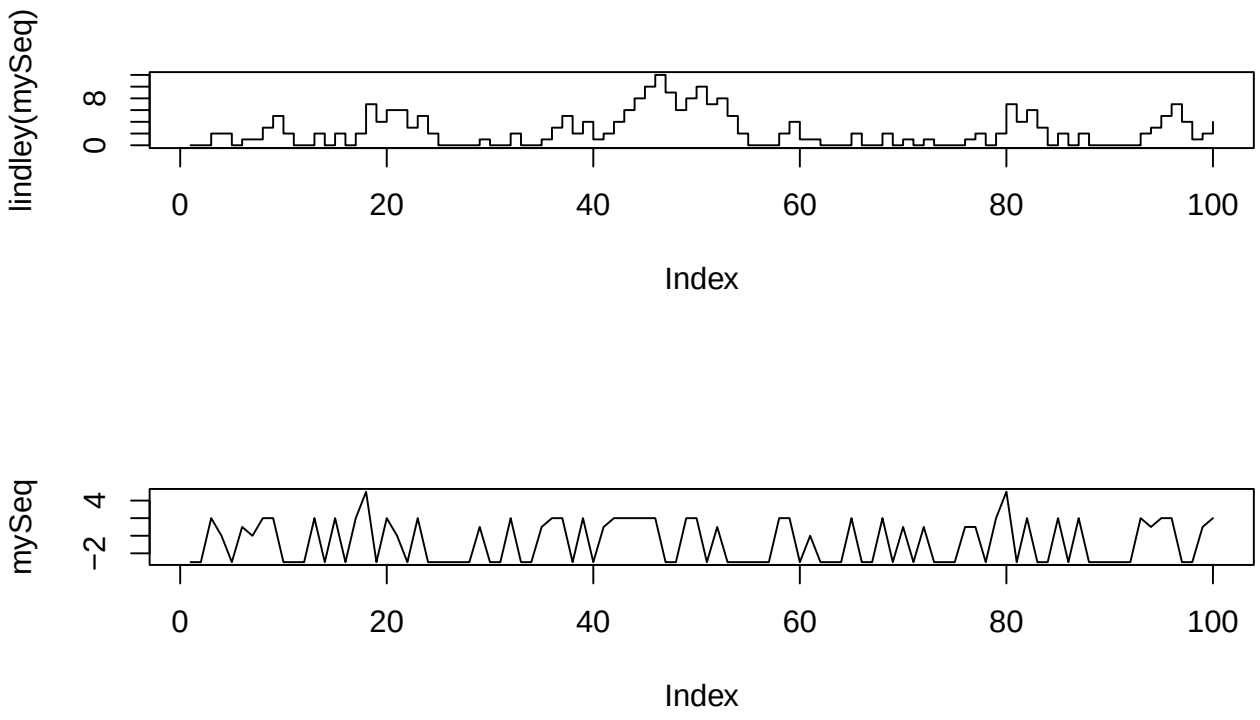
The package provides a number of auxiliary functions for loading or creating sequences for different models.

Lindley Process: to visualize optimal and suboptimal segments

This function takes a score sequence and shows the Lindley process associated, illustrating the operation of the local score algorithm. Plotting the Lindley process provides a view of the potential optimal segments and their comparison between them, and offers a good representation of potential signal along the sequence, better alternative than a sliding window approach as it captures both a punctual high score value and a succession of small positive score values.

```
set.seed(1)
mySeq <- sample(c(-3,2,0,1,5), 100, replace = TRUE, prob = c(0.5, 0.3, 0.05, 0.1, 0.05))
lindley(mySeq)
#>   [1]  0  0  2  2  0  1  1  3  5  2  0  0  2  0  2  0  2  7  4  6  6  3  5  2  0
#>  [26]  0  0  0  1  0  0  2  0  0  1  3  5  2  4  1  2  4  6  8 10 12  9  6  8 10
#>  [51]  7  8  5  2  0  0  0  2  4  1  1  0  0  0  2  0  0  2  0  1  0  1  0  0  0
```

```
#>  [76]  1  2  0  2  7  4  6  3  0  2  0  2  0  0  0  0  0  2  3  5  7  4  1  2  4
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(2,1))
plot(lindley(mySeq), type = "s")
plot(mySeq,typ = 'l')
```




```
par(oldpar)
```

Record times: gives the record times of a sequence

This function takes a score sequence and return a vector with the record times defined as follow: $K_0 = 0$, and $K_{i+1} := inf\{k > K_i : S_k - S_{K_i} < 0\}$, for $i \geq 0$.

```
set.seed(1)
mySeq <- sample(c(-3,2,0,1,5), 100, replace = TRUE, prob = c(0.5, 0.3, 0.05, 0.1, 0.05))
mySeq
#>   [1] -3 -3  2  0 -3  1  0  2  2 -3 -3 -3  2 -3  2 -3  2  5 -3  2  0 -3  2 -3 -3
#>  [26] -3 -3 -3  1 -3 -3  2 -3 -3  1  2  2 -3  2 -3  1  2  2  2  2  2 -3 -3  2  2
#>  [51] -3  1 -3 -3 -3 -3 -3  2  2 -3  0 -3 -3 -3  2 -3 -3  2 -3  1 -3  1 -3 -3 -3
#>  [76]  1  1 -3  2  5 -3  2 -3 -3  2 -3  2 -3 -3 -3 -3 -3  2  1  2  2 -3 -3  1  2
recordTimes(mySeq)
#>  [1]  1  2  5 11 12 14 16 25 26 27 28 30 31 33 34 55 56 57 62 63 64 66 67 69 71
#> [26] 73 74 75 78 86 88 89 90 91 92
```

Score Loading Function

`loadScoreFromFile()` reads a csv-file returning a named list where names correspond to the first file column and values to the second file column. If a third column is available within the file, it will be read and can be

accessed by name, too. An example is given in the first case study. The function is reading a header line by default, so be careful that you have one otherway the first score will be missed.

Empirical distribution: function "scoreSequences2probabilityVector()"

`scoreSequences2probabilityVector()` takes in a list of score sequences and returns the resulting empirical distribution from the minimal to the maximal score value of all sequences. Thus,

```
seq1 <- sample(7:8, size = 10, replace = TRUE)
seq2 <- sample(2:3, size = 15, replace = TRUE)
l <- list(seq1, seq2)
r <- scoreSequences2probabilityVector(l)
r
#>    2    3    4    5    6    7    8
#> 0.24 0.36 0.00 0.00 0.00 0.24 0.16
length(r)
#> [1] 7
```

returns a vector of length 7, even if there are only 4 distinct unique values present in the list. Very useful for the use in any non-simulating method of p-value.

Case study

Medium sequence

Sequence extracted from https://www.uniprot.org/uniprot/P49755/

```
data(Seq219)
data(HydroScore)
SeqScore <- CharSequence2ScoreSequence(Seq219, HydroScore)
n <- length(SeqScore)
n
#> [1] 219
```

Local score computation and parameter model setings

```
LS <- localScoreC(SeqScore)$localScore[1]
LS
#> value
#>    52
```

Parameter model setings

```
prob <- scoreSequences2probabilityVector(list(SeqScore))
prob
#>         -5         -4         -3         -2         -1          0          1
#> 0.06392694 0.25570776 0.02283105 0.03652968 0.15068493 0.06849315 0.00000000
#>          2          3          4          5
#> 0.08675799 0.07762557 0.17808219 0.05936073
```

Exact method

```
time.daudin <- system.time(
  res.daudin <- daudin(
    local_score = LS, sequence_length = n,
       score_probabilities = prob,
       sequence_min = min(SeqScore),
    sequence_max = max(SeqScore)
  )
)
res.daudin
#> [1] 0.2654051
```

Approximated method

The call of the function `karlin()` is similar to the one of `daudin()`.

```
time.karlin <- system.time(
  res.karlin <- karlin(
    local_score = LS, sequence_length = n,
       score_probabilities = prob,
       sequence_min = min(SeqScore),
    sequence_max = max(SeqScore)
  )
)
res.karlin
#> [1] 0.2028498
```

The two p-values are different because the sequence length n is equal 219. It is not enough large to have a good approximation with the approximated method.

Improved approximation

The call of the function 'mcc()' is still the same.

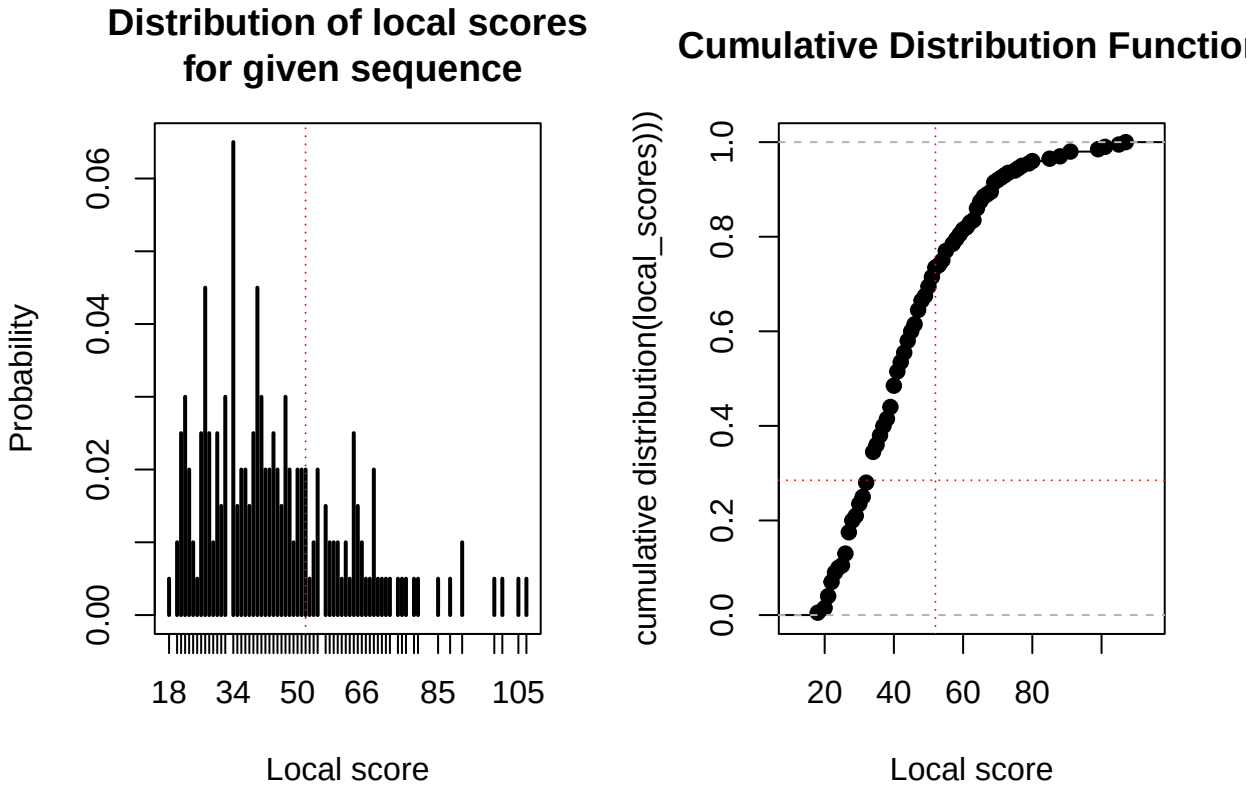
```
time.mcc <- system.time(
  res.mcc <- mcc(
    local_score = LS, sequence_length = n,
       score_probabilities = prob,
       sequence_min = min(SeqScore),
    sequence_max = max(SeqScore)
  )
)
res.mcc
#> [1] 0.214167
```

We can verify here that this approximation is more accurate than the one of Karlin *et al.* for sequences of length of several hundred components.

Monte Carlo

Let us do a very quick empirical computation with only 200 repetitions.

```
time.MonteCarlo1 <- system.time(
  res.MonteCarlo1 <- monteCarlo(
    local_score = LS,
    FUN = function(x) {
      return(sample(
        x = x, size = length(x),
        replace = TRUE
      ))
    },
    x = SeqScore, numSim = 200
  )
)
```

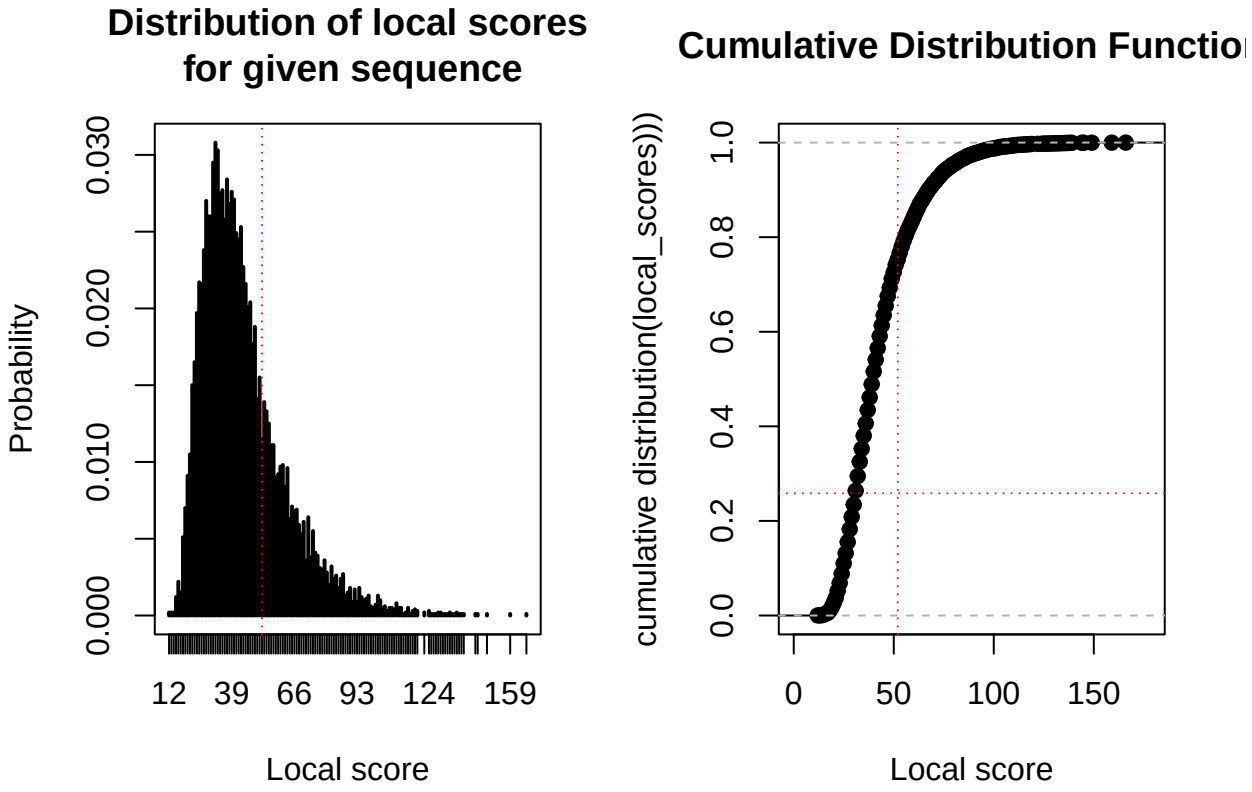



```
res.MonteCarlo1
#> p_value
#>   0.285
```

The p-value estimation is 0.285 which is around the exact value 0.2654051. Let us increase the number of repetition to be more accurate.

```
time.MonteCarlo2 <- system.time(
  res.MonteCarlo2 <- monteCarlo(
    local_score = LS,
    FUN = function(x) {
```

```
      return(sample(
        x = x, size = length(x),
        replace = TRUE
      ))
    },
    x = SeqScore, numSim = 10000
  )
)
```




```
res.MonteCarlo2
#> p_value
#>  0.2584
```

Result and time computation comparison

P-value

```
res.pval <- c(
  Daudin = res.daudin, Karlin = res.karlin, MCC = res.mcc,
  MonteCarlo1 = res.MonteCarlo1, MonteCarlo1 = res.MonteCarlo2
)
names(res.pval) <- c("Exact", "Approximation", "Improved appx", "MonteCarlo1", "MonteCarlo2")
res.pval
#>         Exact Approximation Improved appx   MonteCarlo1   MonteCarlo2
#>     0.2654051     0.2028498     0.2141670     0.2850000     0.2584000
```

Computation time

```
rbind(time.daudin, time.karlin, time.mcc,time.MonteCarlo1, time.MonteCarlo2)
#>                  user.self sys.self elapsed user.child sys.child
#> time.daudin          0.000        0   0.000          0         0
#> time.karlin          0.000        0   0.000          0         0
#> time.mcc             0.000        0   0.000          0         0
#> time.MonteCarlo1     0.062        0   0.062          0         0
#> time.MonteCarlo2     3.037        0   3.038          0         0
```

Time computation for Monte-Carlo method depends to the number of repetition. For medium sequences exact method is around 30 time more time consuming than the mcc method. Karlin's method is the fastest one but can be not accurate if the sequence are too short (here $n = 219$ a couple of hundred is not enough, a thousand may be preferred to be closest to convergence).

Short sequence

```
data(Seq31)
SeqScore.Short <- CharSequence2ScoreSequence(Seq31, HydroScore)
n.short <- length(SeqScore.Short)
n.short
#> [1] 31
```

Sequence of length 31. For short sequences, it is easier and usual to obtain a empirical positive expectation for the score. So the functions based on approximated methods can't be used and an error message is given.

```
SeqScore.S <- SeqScore.Short
LS.S <- localScoreC(SeqScore.S)$localScore[1]
prob.S <- scoreSequences2probabilityVector(list(SeqScore.S))

LS.S
#> value
#>    52
prob.S
#>         -5         -4         -3         -2         -1          0          1
#> 0.03225806 0.06451613 0.00000000 0.00000000 0.22580645 0.06451613 0.00000000
#>          2          3          4          5
#> 0.09677419 0.09677419 0.25806452 0.16129032
```

```
time.daudin <- system.time(
  res.daudin. <- daudin(
    local_score = LS.S, sequence_length = n.short,
       score_probabilities = prob.S,
       sequence_min = min(SeqScore.S),
    sequence_max = max(SeqScore.S)
  )
)

time.karlin <- system.time(
  res.karlin <- try(karlin(
    local_score = LS.S, sequence_length = n.short,
       score_probabilities = prob.S,
       sequence_min = min(SeqScore.S),
```

```
    sequence_max = max(SeqScore.S)
  ))
)
#> Error in eval(expr, envir) :
#>   [Invalid Input] Score expectation must be strictly negative.

time.mcc <- system.time(
  res.mcc <- try(mcc(
    local_score = LS.S, sequence_length = n.short,
       score_probabilities = prob.S,
       sequence_min = min(SeqScore.S),
    sequence_max = max(SeqScore.S)
  ))
)
#> Error in eval(expr, envir) :
#>   [Invalid Input] Score expectation must be strictly negative.

time.karlinMonteCarlo <- system.time(
res.karlinMonteCarlo <-
    karlinMonteCarlo(
      local_score = LS.S, plot = FALSE,
                  sequence_length = n.short,
                  simulated_sequence_length = 1000,
                  FUN = sample, x = min(SeqScore.S):max(SeqScore.S),
                  size = 1000, prob = prob.S, replace = TRUE,
      numSim = 10000
    )
)

time.MonteCarlo <- system.time(
  res.MonteCarlo <- monteCarlo(
    local_score = LS.S, plot = FALSE,
    FUN = function(x) {
      return(sample(
        x = x, size = length(x),
        replace = TRUE
      ))
    },
    x = SeqScore.S, numSim = 10000
  )
)
```

Results

```
res.pval <- c(Daudin = res.daudin, MonteCarlo = res.MonteCarlo)
names(res.pval) <- c("Daudin", "MonteCarlo")
res.pval
#>     Daudin MonteCarlo
#>  0.2654051  0.5873000
rbind(time.daudin, time.MonteCarlo)
#>                 user.self sys.self elapsed user.child sys.child
```

```
#> time.daudin         0.001    0.000   0.000          0         0
#> time.MonteCarlo     2.852    0.004   2.858          0         0
```

Here an example using another probability vector with non positive average score. In this example, the local score is very huge and realized by the whole sequence, the p-value is very low as confirmed by the exact method.

```
set.seed(1)
prob.bis <- dnorm(-5:5, mean = -0.5, sd = 1)
prob.bis <- prob.bis / sum(prob.bis)
names(prob.bis) <- -5:5
# Score Expectation
sum((-5:5)*prob.bis)
#> [1] -0.4999994

time.mcc <- system.time(
  res.mcc <- mcc(
    local_score = LS.S, sequence_length = n.short,
       score_probabilities = prob.bis,
       sequence_min = min(SeqScore.S),
    sequence_max = max(SeqScore.S)
  )
)

time.daudin <- system.time(
  res.daudin <- daudin(
    local_score = LS.S, sequence_length = n.short,
       score_probabilities = prob.bis,
       sequence_min = min(SeqScore.S),
    sequence_max = max(SeqScore.S)
  )
)

simu <- function(n, p) {
  return(sample(x = -5:5, size = n, replace = TRUE, prob = p))
}
time.MonteCarlo <- system.time(
res.MonteCarlo <-
    monteCarlo(
      local_score = LS.S, plot = FALSE,
      FUN = simu, n.short, prob.bis, numSim = 100000
    )
)
```

```
res.pval <- c(MCC=res.mcc,Daudin = res.daudin, MonteCarlo = res.MonteCarlo)
names(res.pval) <- c("MCC", "Daudin", "MonteCarlo")
res.pval
#>          MCC       Daudin   MonteCarlo
#> 0.000000e+00 1.306309e-33 0.000000e+00
rbind(time.mcc,time.daudin, time.MonteCarlo)
#>                 user.self sys.self elapsed user.child sys.child
#> time.mcc            0.001    0.000   0.000          0         0
#> time.daudin         0.000    0.000   0.001          0         0
```

```
#> time.MonteCarlo    28.959    0.009  28.982          0         0
```

For short sequences, exact method is fast, more precise and must be prefered.

Large sequence

```
data(Seq1093)
SeqScore.Long <- CharSequence2ScoreSequence(Seq1093, HydroScore)
n.Long <- length(SeqScore.Long)
n.Long
#> [1] 1093
SeqScore.Long <- CharSequence2ScoreSequence(Seq1093, HydroScore)
LS.L <- localScoreC(SeqScore.Long)$localScore[1]
LS.L
#> value
#>    65
prob.L <- scoreSequences2probabilityVector(list(SeqScore.Long))
prob.L
#>          -5          -4          -3          -2          -1           0           1
#> 0.07410796 0.20311070 0.02012809 0.07502287 0.21225984 0.07776761 0.00000000
#>           2           3           4           5
#> 0.07136322 0.09423605 0.14364135 0.02836231
sum(prob.L*as.numeric(names(prob.L)))
#> [1] -0.4638609
```

Sequence of length 1093 with a local score equal to 65. The average score is non positive so approximated methods can be used.

Results

```
res.pval.L <- c(res.daudin.L, res.mcc.L, res.karlin.L, res.karlinMonteCarlo.L$p_value,res.MonteCarlo.L)
names(res.pval.L) <- c("Daudin", "MCC", "Karlin", "KarlinMonteCarlo", "MonteCarlo")
res.pval.L
#>           Daudin              MCC           Karlin KarlinMonteCarlo
#>       0.07231933       0.07682238       0.07644314       0.07622407
#>       MonteCarlo
#>       0.07210000
```

```
rbind(
  time.daudin.L, time.karlin.L, time.mcc.L, time.karlinMonteCarlo.L,
  time.MonteCarlo.L
)
#>                          user.self sys.self elapsed user.child sys.child
#> time.daudin.L                0.001    0.000   0.001          0         0
#> time.karlin.L                0.000    0.000   0.001          0         0
#> time.mcc.L                   0.001    0.000   0.000          0         0
#> time.karlinMonteCarlo.L      3.957    0.002   3.961          0         0
#> time.MonteCarlo.L            3.983    0.001   3.985          0         0
```

Even for large sequences of several thousands, the exact method is still fast enough but it could become too much time consuming for a sequence data set with numerous sequences. The approximated methods must be preferred.

Several sequences

The function `automatic_analysis()` can analysis a named list of sequences. It choose the adequate method for each sequence. Here in the following example, the exact method is used for the short sequence, whereas an asymptotic method is used for the long one.

```
MySeqsList <- list(Seq31, Seq219, Seq1093)
names(MySeqsList) <- c("Q09FU3.fasta", "P49755.fasta", "Q60519.fasta")
MySeqsScore <- lapply(MySeqsList, FUN = CharSequence2ScoreSequence, HydroScore)
AA <- automatic_analysis(MySeqsScore, model = "iid")
AA$Q09FU3.fasta
#> $`p-value`
#> [1] 0.0005164345
#>
#> $`method applied`
#> [1] "Exact Method Daudin et al"
#>
#> $localScore
#> $localScore$localScore
#> value begin   end
#>    52     1    31
#>
#> $localScore$suboptimalSegmentScores
#>   value begin end
#> 1    52     1  31
#>
#> $localScore$RecordTime
#> [1] 0
AA$Q09FU3.fasta$`method applied`
#> [1] "Exact Method Daudin et al"
AA$Q60519.fasta$`method applied`
#> [1] "Exact Method Daudin et al"
```

We can observe differences between the p-value of the short sequence obtained in the case study for the only short sequence, and the one obtained with the automatic analysis. Note that the distribution vector of the scores used are different which induces a different p-value.

```
cbind(prob, prob.S, prob.L, "3 sequences" = scoreSequences2probabilityVector(MySeqsScore))
#>          prob     prob.S     prob.L 3 sequences
#> -5 0.06392694 0.03225806 0.07410796  0.07148176
#> -4 0.25570776 0.06451613 0.20311070  0.20848846
#> -3 0.02283105 0.00000000 0.02012809  0.02010424
#> -2 0.03652968 0.00000000 0.07502287  0.06701415
#> -1 0.15068493 0.22580645 0.21225984  0.20253165
#> 0  0.06849315 0.06451613 0.07776761  0.07594937
#> 1  0.00000000 0.00000000 0.00000000  0.00000000
#> 2  0.08675799 0.09677419 0.07136322  0.07446016
#> 3  0.07762557 0.09677419 0.09423605  0.09158600
```

```
#> 4  0.17808219 0.25806452 0.14364135  0.15189873
#> 5  0.05936073 0.16129032 0.02836231  0.03648548
```

Using the probability vector of the three sequences to compute the p-value of the local score of the short sequence with the function `daudin()`, we recover an identical p-value than we have obtained with the `automatic analysis()`.

```
daudin.bis <- daudin(local_score = LS.S, sequence_length = n.short, score_probabilities = scoreSequences
daudin.bis
#> [1] 0.0005164345
AA$P49755.fasta$`p-value`
#> [1] 0.08626993

# automatic_analysis(sequences=list('MySeq.Short'=MySeq.Short), model='iid', distribution=proba.S)
```

A larger example with a SCOP data base

```
library(localScore)
data(HydroScore)
data(SeqListSCOPe)
MySeqScoreList <- lapply(SeqListSCOPe, FUN = CharSequence2ScoreSequence, HydroScore)
head(MySeqScoreList)
#> $P50456
#>   [1]  2 -5 -4  4  5 -4  4  4  5 -4 -3 -4  4  0  2  0  4  5 -1 -4  0 -3  4  4 -3
#>  [26]  2  0 -1 -1 -1  4  4 -4  5  0 -3 -1 -4  4 -4 -2 -1  0 -4 -5  3 -1  3  0 -4
#>  [51] -3  0  3  4 -4 -1  5  2 -1  4 -4 -1  5  4 -4  4  2 -4  4 -5  4 -4 -4 -1  2
#>  [76] -1 -1  2  4 -3  0 -4 -2  4 -1  4 -4 -1  4  3 -4  2  2  4 -5  0 -4  4  4  2
#> [101] -4 -4  5  5 -1  0  4  0  2 -3  4  0 -5  5  4  2  5  2  4 -4  4  3 -4 -2 -4
#> [126] -4  5  4  5  0 -1 -2  4 -1 -4  2  2 -4  5  4  3 -2  4  5 -1 -4 -1  5 -5 -4
#> [151] -4  2  4 -2  2 -1 -1 -4 -3  5 -1  4 -4 -1 -1
#>
#> $P14859
#>  [1] -4 -4 -2 -1 -4  4 -4 -4  4 -4 -4  3  2 -4 -1  3 -4 -4 -5 -5  5 -4  4  0  3
#> [26] -1 -4  0 -4  4  0  4  2  2  0 -4  4 -1  0 -4 -4  3 -1 -4 -1 -1  5 -1 -5  3
#> [51] -4  2  4 -4  4 -1  3 -4 -4  2  3 -4  4 -4 -2  4  4 -4 -4 -1  4 -4 -4  2 -4
#>
#> $P14859
#>  [1]  5 -4 -1 -4  5 -5  4  2  4 -4 -4 -1  3  4 -4 -4 -4 -4 -2 -1 -1 -4 -4  5 -1
#> [26]  2  5  2 -4 -4  4 -4  2 -4 -4 -4  4  5 -5  4 -1  3  3 -4 -5 -5 -4 -4 -4 -4
#> [51] -5  5
#>
#> $P10037
#>  [1]  5 -1  5  2  2 -4 -4  2  4 -4 -5 -3  3  0 -4 -3 -1 -4 -2 -1 -1 -4 -4  5  2
#> [26] -5  2  2 -4 -4  4 -4  4 -4 -4 -4  4  4 -5  4 -1  3  3 -4 -5 -5 -4 -5 -4 -4
#> [51] -5  4
#>
#> $Q13619
#>  [1] -4 -1  4 -4 -4 -4  4 -1 -1 -1 -4 -5  4  3 -4 -4 -5 -4 -1 -4  5 -4  2  2  5
#> [26]  4 -5  5  2 -4  2 -5 -4 -1  4  0 -3 -4  4  4  4 -1 -4  4 -1 -4 -4  4 -4  3
#> [51] -2  4 -4 -2  0 -4  4 -4 -4 -5  5 -4 -1  4  5 -4 -5 -4 -1  2 -4 -5 -4 -4 -4
#> [76] -4 -2 -4 -4 -1 -3 -1  4  2
```

```
#>
#> $Q13619
#>  [1]  4 -1  4  3 -4 -1  4  4  4  4  2  3 -4 -4  0 -4  0  3 -1  3 -4 -4  5 -4  2
#> [26]  2 -1  0  5 -4 -4 -1 -4  4 -5 -5 -1  4 -4 -1  4  2  3  0 -4  2 -5  4  4  5
#> [51] -4 -1 -2 -4  0 -4 -4  4 -4 -4  0 -4 -4  3  5  3 -4
AA <- automatic_analysis(sequences = MySeqScoreList, model = "iid")
AA[[1]]
#> $`p-value`
#> [1] 0.06389172
#>
#> $`method applied`
#> [1] "Exact Method Daudin et al"
#>
#> $localScore
#> $localScore$localScore
#> value begin   end
#>    67     4   144
#>
#> $localScore$suboptimalSegmentScores
#>   value begin end
#> 1     2     1   1
#> 2    67     4 144
#>
#> $localScore$RecordTime
#> [1] 0 2 3
# the p-value of the first 10 sequences
sapply(AA, function(x) {
  x$`p-value`
})[1:10]
#>     P50456     P14859     P10037     Q13619     P22262     P20823     P07014
#> 0.06389172 0.97260747 0.87470986 0.89625648 0.45058860 0.96651306 0.74891930
#>     Q9X399     Q0SB06     Q9I641
#> 0.68145292 0.99374006 0.51203351
# the 20th smallest p-values
sort(sapply(AA, function(x) {
  x$`p-value`
}))[1:20]
#>       Q5SMG8       P0A334       Q2W6R1       O27564       P12282       P50456
#> 9.485100e-07 3.442818e-04 4.406208e-04 4.548065e-04 6.167591e-02 6.389172e-02
#>       Q58194       Q8AA93       P05523       P28793       P55038       Q9WZ12
#> 7.290625e-02 9.038738e-02 9.064666e-02 9.414140e-02 9.498902e-02 1.017792e-01
#>       Q9KQJ1       Q5ZSV0       P0A544       P77072       P0A9G8       P08709
#> 1.304836e-01 1.341404e-01 1.356180e-01 1.410952e-01 1.481435e-01 1.490303e-01
#>       P00390       Q13564
#> 1.509551e-01 1.574591e-01
which(sapply(AA, function(x) {
  x$`p-value`
}) < 0.05)
#> Q2W6R1 O27564 P0A334 Q5SMG8
#>     14     90    150    192
table(sapply(AA, function(x) {
  x$`method`
}))
```

```
#>
#> Exact Method Daudin et al
#>                        206
# The maximum sequence length equals 404 so it here normal that the exact method is used for all the 60
scoreSequences2probabilityVector(MySeqScoreList)
#>         -5         -4         -3         -2         -1          0          1
#> 0.05537938 0.26383764 0.02153482 0.04105166 0.14754382 0.07195572 0.00000000
#>          2          3          4          5
#> 0.10487777 0.05241006 0.17481550 0.06659363
```

File Formats

This package allows input in file form. For the package to work, please respect the following conventions.

Sequence Files

The package accepts files in FASTA format: Every sequence is preceeded by a title (marked by a ">") and a line break. One sequence takes one line, followed by a line break and a line only containing a tab.

```
>HUMAN_NM_018998_2
TGAGTAGGGCTGGCAGAGCTGGGGCCTCATGGCTGTGTAGTAGCAGGCCCCCGCCCCGCGACCTGGCCAGGCGATCACTACAGCCGCCCCTGCCGAACAG

>Mouse_NM_013908_3
CCCCATGAGGACCCAGAACCCTCAATGGAGAAGAGTCAGGATTTGCTGTGCTGCCAGAGTGAACTGGCCTGGTAATTACCCTGCAGCCTTTCTGGAACAG

>HUMAN_NM_018998_3
GTGAGCACGGGCGGCGGGTTGACCCTGCCCCCGCCCCACGCCGACAGCCTGTCCAGCCCCGGCCTCCCCACAG

>Mouse_NM_013908_4
GTAAGTGTGGGCATTGGGTTGGGCTACCTGTCCCATTGTGCCCTGCCAGCAGTCTGCCCAGCTGTGGCCTTCCCCCCAG

>HUMAN_NM_018998_5
GGTGCTCACAGCCCCAGAGACACCACTGAGGTAGGAAGCTGCCCTGGAGTGATGTCCTTGGGGCATTGGACAGGGACCCTCACCGTAGCCCTCCCTGCAG
```

Score Files

A score file is a csv file that contains a header line and each line contains a letter and its score. Optionally one can also provide a probability for each score. Example:

```
Letters,Scores,Probabilities
L,-2,0.04
M,-1,0.04
N,0,0.04
```

Transition Matrix Files

A csv file only containing the values of the matrix. Example:

```
0.2,0.3,0.5
0.3,0.4,0.3
0.2,0.4,0.4
```