FunChIP: A Functional Data Analysis approach to cluster ChIP-Seq peaks according to their shapes

Alice Parodi *alicecarla.parodi@polimi.it*

Laura M. Sangalli
Piercesare Secchi
Simone Vantini
Marco J. Morelli

May 2, 2019

Contents


```
> library(FunChIP)
```

1 Introduction

The *FunChIP* package provides a set of methods for the *GRanges* class of the package *GenomicRanges* to cluster ChIP-Seq peaks according to their shapes, starting from a `BAM` file containing the aligned reads and a *GRanges* object with the corresponding enriched regions.

2 Input and Preprocessing

ChIP-Seq enriched regions are provided by the user in a *GRanges* object `GR`. The user must provide the `BAM` file containing the reads aligned on the positive and negative strands of the DNA. From the `BAM` file we can compute, for each region of the *GRanges* (let N be the total number of regions), the base-level coverage separately for positive and negative reads. These two count vectors are used to estimate the distance d_{pn} between positive and negative reads and then the total length of the fragments of the ChIP-Seq experiment d. In particular, we assume that the positive and negative counts measure the same signal, shifted by d_{pn}, as they are computed from the two ends of the sequencing fragments. The global length of the fragment is the sum between the length of the reads of the `BAM` file, r[1], and the distance between the positive and negative coverage d_{pn}

$$d = d_{pn} + r.$$

[1] If in the `BAM` file multiple length are present, r is estimated as the average length.

The function `compute_fragments_length` computes, from the *GRanges* object and the `BAM` file, the estimated length of the fragments. Given a range for d_{pn}: $[d_{\min}; d_{\max}]$, the optimum distance d_{pn} is

$$d_{pn} = \text{argmin}_{\delta \in [d_{\min}:d_{\max}]} \sum_{n=1}^{N} D(f_{n+}, f^{\delta}_{n-}),$$

where f_{n+} is the positive coverage function of the n-th region, and f^{δ}_{n-} is the negative coverage of the n th region, shifted by δ. The distance D is the square of the L^2 distance between the coverages, normalized by the width of the region. The definition of the L^2 distance is detailed in Section 4.

```
> # load the GRanges object associated
> # to the ChIP-Seq experiment on the
> # transcription factor c-Myc in murine cells
>
> data(GR100)
> # name of the .bam file (the
> # .bam.bai index file must also be present)
>
> bamf <- system.file("extdata", "test.bam",
+                     package="FunChIP", mustWork=TRUE)
> # compute d
>
> d <- compute_fragments_length(GR, bamf, min.d = 0, max.d = 300)

[1] "estimated distance positive - negative read 148"
[1] "estimated read length 51"

> d

[1] 199

>
```

In Figure 1 the distance function is shown varying the parameter δ, and the minimum value d_{pn} is computed.

Once we have correctly identified the fragment length we can compute the final coverage function to obtain the shape of the peaks. The `pileup_peak` method for the *GRanges* class uses the `BAM` file to compute the base-level coverage on these regions, once the reads are

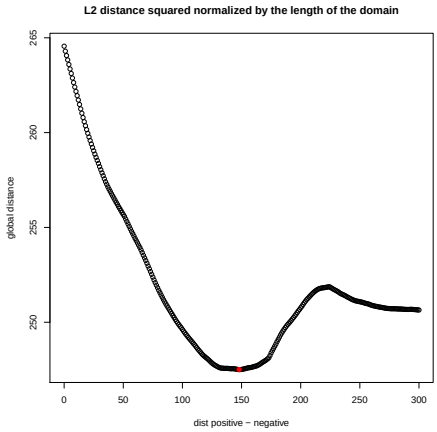

Figure 1: Identification of d
optimal value of d_{pn} is presented. It is the minimum of the global distance function.

extended up to their final length d. `pileup_peak` adds to the *GRanges* a `counts` metadata column, containing for each region a vector with length equal to the width of the region storing the coverage function.

```
> # associate to each peak
> # of the GRanges object the correspondent
> # coverage function
>
> peaks <- pileup_peak(GR, bamf, d = d)
> peaks

GRanges object with 100 ranges and 1 metadata column:
        seqnames            ranges strand |
           <Rle>         <IRanges>  <Rle> |
    [1]    chr18   3337524-3338025      * |
    [2]    chr18   4369126-4369352      * |
    [3]    chr18   4375448-4375883      * |
    [4]    chr18   4715744-4716162      * |
    [5]    chr18   4716374-4716597      * |
    ...      ...               ...    ... .
   [96]    chr18 35112325-35112593      * |
   [97]    chr18 35113538-35114826      * |
   [98]    chr18 35118063-35118304      * |
   [99]    chr18 35182164-35182425      * |
  [100]    chr18 35205390-35205649      * |


    [1]
    [2]
    [3]
    [4]
    [5]
    ...
   [96]
```

```
 [97] c(9, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 9, 8, 8, 8, 8, 9, 9, 9, 9, 9, 10, 10, 11, 11, 11, 12,
 [98]
 [99]
[100]
-------
seqinfo: 20 sequences from an unspecified genome; no seqlengths
>
```

Additional information can be found in the help page of the `pileup_peak` method.

3 Smoothing

The `counts` metadata is approximated by a combination of splines to guarantee the smoothness and regularity needed for further analysis, as described in the following Sections.
The preprocessing steps carried out in the `smooth_peak` method are the following:

- *Removal of the background and extension.* In ChIP-Seq experiments, peaks may have an additive noisy background, and the removal of this background is mandatory to compare different peaks. The background is estimated as a constant value "raising" the peak and equal to the minimum value the coverage assumes. Consequently, once the background has been removed, each peak has zero as minimum value, thus allowing the peak to be indefinitely extended with zeros, if necessary. In Section 4, how this choice affects the algorithm will be discussed.
- *Smoothing.* In order to be regular enough to computed derivatives, a peak has to be transformed in a suitable functional object, as described in Section 4. The smoothing of the count vector c is performed through the projection of c on a cubic B-spline basis $\Phi = \{\phi_1, \dots \phi_K\}$ with a penalization on the second derivative [1]. The result is a spline approximation of the data, which is continuous on the whole domain, together with its first order derivatives. Moreover, the penalization on the second derivative allows to control the global regularity of the function avoiding over-fitting and a consequent noisy spline definition. The spline approximation $s = \sum_{k=1}^{K} \theta_k \phi_k$ of the count vector $c = \{c_j\}$ is defined minimizing

$$S(\lambda) = \sum_{j=1}^{n} [c_j - s(x_j)]^2 + \lambda \int [s''(x)]^2 \, dx,$$

with x_j being the relative genomic coordinate the counts. The multiplying coefficient λ quantifies the penalization on the second derivative and is chosen through the Generalized Cross Validation criteria. For each peak i the GCV_i index is computed with a leave-one-out cross validation

$$GCV_i = \left(\frac{n}{n - df(\lambda)}\right)\left(\frac{SSE_i}{n - df(\lambda)}\right)$$

and then it is summed on the whole data set to obtain the global GCV. The number of degrees of freedom $df(\lambda)$ is automatically computed from the definition of the basis Φ.
The error SSE_i can be computed either on the data (SSE_i^0) or on the derivatives

(SSE_i^1), to control the regularity of the function or the regularity of the derivatives, respectively:

$$SSE_i^0 = \sqrt{\sum_{j=1}^{n} (c_j - s(x_j))^2} \text{ or } SSE_i^1 = \sqrt{\sum_{j=1}^{n-1} (\nabla c_j - s'(x_j))^2},$$

with ∇c_j being the finite-difference approximation of the derivative of the `counts` vector c for the data i: $c = c(i)$, while $s'(x_i)$ is the evaluation of the first derivative $s' = s'(i)$ on the genomic coordinates. For further details on the spline definition see the `spline` function of the *fda* package.

- *Scaling of the peaks.* This optional preprocessing step makes all the curves having the same width and area. In particular all the abscissa grid are scaled to become equal to the smallest grid throughout the data, while $y-$values are scaled to make areas of all the curves equal to 1.

The `smooth_peak` method approximates the `counts` metadata by removing the background, computing the spline and potentially defining the scaled approximation. Focusing on the spline approximation, `smooth_peak` automatically chooses the optimal λ parameter according to the GCV criteria; the user can decide whether to consider the data or the derivatives to compute the SSE.

```
> # the method smooth_peak
> # removes the background and defines the spline
> # approximation from the previously computed peaks
> # with lambda estimated from the
> # GCV on derivatives. The method spans a non-uniform
> # grid for lambda from 10^-4 to 10^12.
> # ( the grid is uniform for log10(lambda) )
>
> peaks.smooth <- smooth_peak(peaks, lambda = 10^(-4:12),
+                             subsample.data = 50, GCV.derivatives = TRUE,
+                             plot.GCV = TRUE, rescale = FALSE )

[1] "iteration on lambda:  1"
[1] "iteration on lambda:  2"
[1] "iteration on lambda:  3"
[1] "iteration on lambda:  4"
[1] "iteration on lambda:  5"
[1] "iteration on lambda:  6"
[1] "iteration on lambda:  7"
[1] "iteration on lambda:  8"
[1] "iteration on lambda:  9"
[1] "iteration on lambda:  10"
[1] "iteration on lambda:  11"
[1] "iteration on lambda:  12"
[1] "iteration on lambda:  13"
[1] "iteration on lambda:  14"
[1] "iteration on lambda:  15"
[1] "iteration on lambda:  16"
[1] "iteration on lambda:  17"

>
```

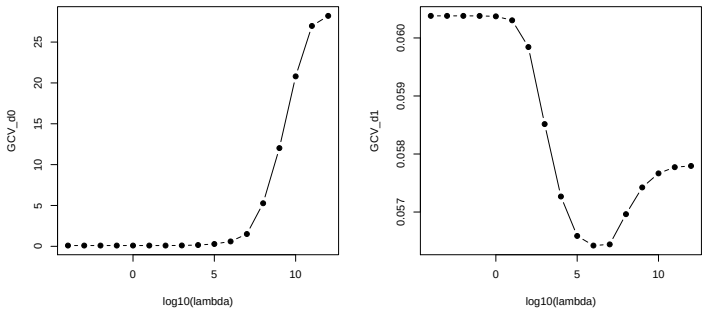

Figure 2: Generalized Cross Validation index
GCV computed on data (left), and on the derivatives (right), as a function of λ.

In Figure 2, the plot of the GCV for both data and derivatives is shown. From this Figure we see that the optimum value of λ, which minimizes the GCV for the derivatives, is also associated to a small value of the GCV for the data thus supporting the automatic choice.

```
> # the automatic choice is lambda = 10^6
>
> peaks.smooth <- smooth_peak(peaks, lambda = 10^6,
+                             plot.GCV = FALSE)
> head(peaks.smooth)

GRanges object with 6 ranges and 6 metadata columns:
      seqnames          ranges strand |
         <Rle>       <IRanges>  <Rle> |
  [1]    chr18 3337524-3338025      * |
  [2]    chr18 4369126-4369352      * |
  [3]    chr18 4375448-4375883      * |
  [4]    chr18 4715744-4716162      * |
  [5]    chr18 4716374-4716597      * |
  [6]    chr18 4921270-4921506      * |


  [1] c(7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8
  [2]
  [3]                       c(8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 11, 11, 11, 1
  [4]
  [5]
  [6]


  [1] c(0.107576984737605, 0.115943638884655, 0.124571296466484, 0.133463063982055, 0.142622047930333, 0.152
  [2]
  [3]
  [4]
  [5]
  [6]
```

```
  [1] c(0.00823718792931595, 0.00849663811461222, 0.00875919479887256, 0.00902485798209696, 0.00929362766428
  [2]
  [3]
  [4]
  [5]
  [6]
      width_spline start_spline end_spline
         <integer>    <numeric>  <numeric>
  [1]          593      3337483    3338075
  [2]          335      4369075    4369409
  [3]          508      4375417    4375924
  [4]          523      4715698    4716220
  [5]          315      4716303    4716617
  [6]          332      4921234    4921565
  -------
  seqinfo: 20 sequences from an unspecified genome; no seqlengths

> # mantaining this choice of lambda smooth_peak
> # can also define the scaled approximation
> # of the spline
>
> peaks.smooth.scaled <- smooth_peak(peaks, lambda = 10^6,
+                             plot.GCV = FALSE, rescale = TRUE)
> head(peaks.smooth.scaled)

GRanges object with 6 ranges and 8 metadata columns:
      seqnames          ranges strand |
         <Rle>       <IRanges>  <Rle> |
  [1]    chr18 3337524-3338025      * |
  [2]    chr18 4369126-4369352      * |
  [3]    chr18 4375448-4375883      * |
  [4]    chr18 4715744-4716162      * |
  [5]    chr18 4716374-4716597      * |
  [6]    chr18 4921270-4921506      * |


  [1] c(7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
  [2]
  [3]                       c(8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 11, 11, 11, 1
  [4]
  [5]
  [6]


  [1] c(0.107576984737605, 0.115943638884655, 0.124571296466484, 0.133463063982055, 0.142622047930333, 0.152
  [2]
  [3]
  [4]
  [5]
  [6]
```

```
  [1] c(0.00823718792931595, 0.00849663811461222, 0.00875919479887256, 0.00902485798209696, 0.00929362766428
  [2]
  [3]
  [4]
  [5]
  [6]
      width_spline start_spline end_spline
         <integer>    <numeric>  <numeric>
  [1]          593      3337483    3338075
  [2]          335      4369075    4369409
  [3]          508      4375417    4375924
  [4]          523      4715698    4716220
  [5]          315      4716303    4716617
  [6]          332      4921234    4921565


  [1]                                                                  c(0.00263609710663517, 0.0030440063908373
  [2]              c(0.00122834484766426, 0.00147221033932346, 0.00172653795382648, 0.00199155049789865, 0.00
  [3]            c(0.000864564256647586, 0.00120333309941394, 0.00156495680464632, 0.00195011992527804, 0.002
  [4]            c(0.000788310894250635, 0.000956445833827378, 0.00113504914292024, 0.0013243784475671, 0.001
  [5] c(0.00235269135248368, 0.00273549871311195, 0.00313329749625751, 0.00354638183650247, 0.00397503469483
  [6]         c(0.00139356234463668, 0.0016220858943517, 0.00186006713373021, 0.00210769852286914, 0.0023651


  [1]
  [2]                                  c(0.000238708662407768, 0.000249059436995862, 0.000259632908095442, 0
  [3]                              c(0.000327571761954349, 0.000350081098788865, 0.000373281486886386, 0.0003
  [4]                                                                  c(0.000162985979570835, 0.0001733265
  [5]       c(0.000375410625363164, 0.00039025358389014, 0.00040539347039775, 0.00042082190469042, 0.0004365
  [6] c(0.000223858745449888, 0.000233220374263462, 0.000242774124776864, 0.000252521012181378, 0.0002624620
  -------
  seqinfo: 20 sequences from an unspecified genome; no seqlengths

>
```

Now the *GRanges* object contains, besides `counts`, 5 new metadata columns with the spline approximation evaluated on the base-level grid, its derivatives, the width of the spline and the new starting and ending points (see Figure 10). For a more detailed description of the metadata columns, see the help page of the `smooth_peak` method.

With the introduction of the smoothing, counts at the edges of the peak are connected with regularity to 0, and therefore new values different from zeros may be introduced. In order to maintain regularity, the grid is extended up to the new boundaries.

Adding to `smooth_peak` the option `rescale = TRUE` the method, beside the 5 metadata columns previously introduced, returns 2 more metadata columns with the scaled approximation of the spline and its derivatives.

Once the spline approximation is defined, the summit of the smoothed peak (or even of the scaled peak), i.e. of its spline approximation, can be detected. The summit will be used to initialize the peak alignment procedure, described in Section 4, and it can either be a user-defined parameter, stored in a vector of the same length of the `GR`, or automatically computed

as the maximum height of the spline. The summit is stored in the new metadata column `summit_spline`. If the `rescale` option is set to `TRUE` the summit of the scaled approximation is also returned in the metadata colum `summit_spline_rescaled`.

```
> # peaks.summit identifies the maximum point
> # of the smoothed peaks
>
> peaks.summit <- summit_peak(peaks.smooth)
> head(peaks.summit)

GRanges object with 6 ranges and 7 metadata columns:
      seqnames          ranges strand |
         <Rle>       <IRanges>  <Rle> |
  [1]    chr18 3337524-3338025      * |
  [2]    chr18 4369126-4369352      * |
  [3]    chr18 4375448-4375883      * |
  [4]    chr18 4715744-4716162      * |
  [5]    chr18 4716374-4716597      * |
  [6]    chr18 4921270-4921506      * |


  [1] c(7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
  [2]
  [3]                       c(8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 11, 11, 11, 1
  [4]
  [5]
  [6]


  [1] c(0.107576984737605, 0.115943638884655, 0.124571296466484, 0.133463063982055, 0.142622047930333, 0.152
  [2]
  [3]
  [4]
  [5]
  [6]


  [1] c(0.00823718792931595, 0.00849663811461222, 0.00875919479887256, 0.00902485798209696, 0.009293627664285
  [2]
  [3]
  [4]
  [5]
  [6]
      width_spline start_spline end_spline summit_spline
         <integer>    <numeric>  <numeric>     <integer>
  [1]          593      3337483    3338075           444
  [2]          335      4369075    4369409           186
  [3]          508      4375417    4375924           174
  [4]          523      4715698    4716220           310
  [5]          315      4716303    4716617           121
  [6]          332      4921234    4921565           169
  -------
```

```
  seqinfo: 20 sequences from an unspecified genome; no seqlengths

> # peaks.summit can identify also the maximum
> # point of the scaled approximation
>
> peaks.summit.scaled <- summit_peak(peaks.smooth.scaled,
+                            rescale = TRUE)
> head(peaks.summit.scaled)

GRanges object with 6 ranges and 10 metadata columns:
      seqnames          ranges strand |
         <Rle>       <IRanges>  <Rle> |
  [1]    chr18 3337524-3338025      * |
  [2]    chr18 4369126-4369352      * |
  [3]    chr18 4375448-4375883      * |
  [4]    chr18 4715744-4716162      * |
  [5]    chr18 4716374-4716597      * |
  [6]    chr18 4921270-4921506      * |


  [1] c(7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
  [2]
  [3]                       c(8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 11, 11, 11, 1
  [4]
  [5]
  [6]


  [1] c(0.107576984737605, 0.115943638884655, 0.124571296466484, 0.133463063982055, 0.142622047930333, 0.152
  [2]
  [3]
  [4]
  [5]
  [6]


  [1] c(0.00823718792931595, 0.00849663811461222, 0.00875919479887256, 0.00902485798209696, 0.009293627664285
  [2]
  [3]
  [4]
  [5]
  [6]
      width_spline start_spline end_spline
         <integer>    <numeric>  <numeric>
  [1]          593      3337483    3338075
  [2]          335      4369075    4369409
  [3]          508      4375417    4375924
  [4]          523      4715698    4716220
  [5]          315      4716303    4716617
  [6]          332      4921234    4921565
```

```
[1]                                                                  c(0.00263609710663517, 0.003044006390837
[2]              c(0.00122834484766426, 0.00147221033932346, 0.00172653795382648, 0.00199155049789865, 0.00
[3]            c(0.000864564256647586, 0.00120333309941394, 0.00156495680464632, 0.00195011992527804, 0.002
[4]            c(0.000788310894250635, 0.000956445833827378, 0.00113504914292024, 0.0013243784475671, 0.001
[5] c(0.00235269135248368, 0.00273549871311195, 0.00313329749625751, 0.00354638183650247, 0.00397503469483
[6]         c(0.00139356234463668, 0.0016220858943517, 0.00186006713373021, 0.00210769852286914, 0.0023651


[1]
[2]                                  c(0.000238708662407768, 0.000249059436995862, 0.000259632908095442, 0
[3]                              c(0.000327571761954349, 0.000350081098788865, 0.000373281486886386, 0.0003
[4]                                                                  c(0.000162985979570835, 0.0001733265
[5]       c(0.000375410625363164, 0.00039025358389014, 0.00040539347039775, 0.00042082190469042, 0.0004365
[6] c(0.000223858745449888, 0.000233220374263462, 0.000242774124776864, 0.000252521012181378, 0.0002624620
    summit_spline_rescaled summit_spline
                 <integer>     <integer>
[1]                    227           444
[2]                    168           186
[3]                    104           174
[4]                    180           310
[5]                    117           121
[6]                    155           169
-------
seqinfo: 20 sequences from an unspecified genome; no seqlengths
```

4 The k-mean alignment algorithm and the `cluster_peak` method

The k-mean alignment algorithm is an efficient method to classify functional data allowing for general transformation of abscissae [2]; this general method is implemented in the package *fdakma* and various applications to real dataset are introduced in [3], [4], [5].

In particular, given

- a set of curves $s_1, \ldots, s_n$,
- the number of clusters K,
- a distance function $d(s_i, s_j)$ between two curves s_i and s_j, as for example the integral of the difference $s_i - s_j$,
- a family of warping functions $\mathcal{W}$ to transform the abscissae of the curves and therefore align the peaks. Generally, $\mathcal{W}$ is the set of shifts or dilations or affine transformations (shift + dilation),

the algorithm, presented in Algorithm 1, is an iterative procedure to split the curves into K clusters. The introduction of the warping function $h \in \mathcal{W}$ allows each curve to be shifted, dilated, or both, to define the minimum distance between curves. The new curve $s \circ h$ has the same values of s, but its abscissa grid is modified. For example, in Figure 3 two peaks are presented: in the left panel, they are not aligned, while the right panel shows the effects of alignment; the transformation of the abscissae (shift transformation) makes the two peaks

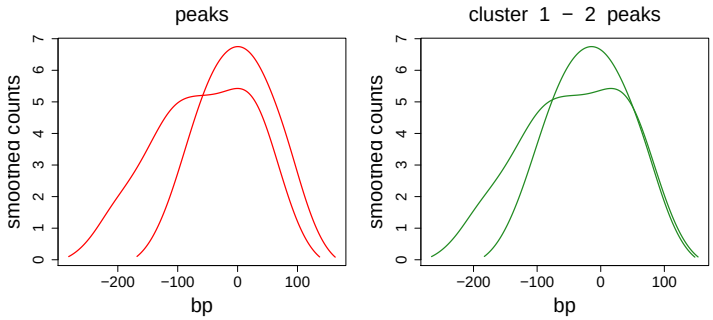

Figure 3: Alignment procedure
Representation of two smoothed peaks. In the left panel they are not aligned, while in the right panel they are aligned with an integer shift.

more similar, and the distance d is not anymore affected by artificial phase distance. The code generating Figure 3 calls `cluster_peak` and `plot_peak`, which are described in Section 4.2 and Section 5.

```
> # representation of two peaks
>
> par (mfrow = c(1,2))
> plot_peak(peaks.summit, index = c(6,7), col=c('red',2), cex.main = 2, cex.lab = 2, cex.axis = 1.5, lwd = 2,
> aligned.peaks <- cluster_peak(peaks.summit[c(6,7)], parallel = FALSE ,
+                                    n.clust = 1, seeds = 1, shift.peak = TRUE,
+                                    weight = 1, alpha = 1, p = 2, t.max = 2,
+                                    plot.graph.k = FALSE, verbose = FALSE)
> aligned.peaks

GRanges object with 2 ranges and 10 metadata columns:
      seqnames          ranges strand |
         <Rle>       <IRanges>  <Rle> |
  [1]    chr18 4921270-4921506      * |
  [2]    chr18 5078473-5078803      * |


  [1]
  [2] c(11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 1


  [1]
  [2] c(0.100641001405493, 0.110866046987915, 0.121441682487124, 0.132373529688273, 0.143667210376516, 0.155


  [1]
  [2] c(0.0100516245524145, 0.0103994035766233, 0.0107528043859863, 0.0111118269805035, 0.011476471360175, 0
      width_spline start_spline end_spline summit_spline cluster_shift coef_shift
         <integer>    <numeric>  <numeric>     <integer>       <list>     <list>
  [1]          332      4921234    4921565           169            1        -15
  [2]          420      5078418    5078837           283            1         16
            dist_shift
                <list>
```

```
  [1] 0.447149205202752
  [2]                 0
  -------
  seqinfo: 20 sequences from an unspecified genome; no seqlengths

> # shift coefficients
> aligned.peaks$coef_shift

[[1]]
[1] -15

[[2]]
[1] 16

> plot_peak(aligned.peaks, col = 'forestgreen',
+           shift = TRUE, k = 1, cluster.peak = TRUE,
+           line.plot = 'spline',
+           cex.main = 2,  cex.lab = 2, cex.axis = 1.5, lwd = 2)
```

For the specific case of ChIP-Seq data, the admitted warping functions for the k-mean alignment algorithm (in the `cluster_peak` method), are integer shifts:

$$\mathcal{W} = \{h : h(t) = t + q \text{ with } q \in \mathbb{Z}\}. \quad (1)$$

In other words, with this choice, peaks can be shifted by integer values in the *alignment* procedure of the algorithm.

Algorithm 1: k-mean alignment algorithm

Given a set of functions $s_1, \ldots, s_n$ and a number K of clusters
Template: random choice (if not provided) of the initial centers of the clusters $c_1 \ldots, c_k$
while *decrease of the distance higher than a fixed threshold* **do**
  foreach $i \in 1 : n$ **do**
    Alignment: s_i is aligned to each template c_k: the optimal warping function $h^\star_{i,k}$ in $\mathcal{W}$ is detected

$$h^\star_{i,k} = \text{argmin}_{h \in \mathcal{W}}\ d(c_k, x_i \circ h)$$

    with the corresponding distance $d^\star_{i,k} = \min_{h \in \mathcal{W}} d(c_k, x_i \circ h)$
    Assignment: s_i is assigned to the best cluster

$$k^\star_i = \text{argmin}_{k \in 1:K}\ d^\star_{i,k}$$

  end
  foreach $k \in 1 : K$ **do**
    Template: identification of the new template of the cluster c_k
    Normalization: the average warping function of the curves belonging to k is set to be the identity transformation

$$h(s) = s$$

  end
end

In the `cluster_peak` method the distance between two curves s_1 and s_2 is defined as

$$\begin{aligned} d(s_1, s_2) &= (1-\alpha)\, d_0(s_1, s_2) + \alpha\, w\, d_1(s_1, s_2) = \\ &= (1-\alpha)\, \|s_1^e - s_2^e\|_p + \alpha\, w\, \|(s_1^e)' - (s_2^e)'\|_p, \end{aligned} \quad (2)$$

where

- $\|f\|_p$ is the p norm of f. In particular, for $p = 0$, $\|\cdot\|_p$ is the L^∞ norm

$$\|f\|_0 = \|f\|_{L^\infty} = \max_{x \in U} |f(x)|,$$

  with U being the domain of f.
  For $p = 1$, $\|\cdot\|_p$ is the L^1 norm

$$\|f\|_1 = \|f\|_{L^1} = \int_U |f(x)| dx.$$

  And for $p = 2$, $\|\cdot\|_p$ is the L^2 norm

$$\|f\|_2 = \|f\|_{L^2} = \int_U (f(x))^2 \, dx.$$

- s_1^e and s_2^e are the functions s_1 and s_2 extended with zeros where not defined, after their backgrounds have been removed (see Section 2). The distance function is computed on the union of the domains of s_1 and s_2 (U); s_1 and s_2 need to be extended to cover the whole U.
- $\alpha \in [0, 1]$ is a coefficient tuning the contributions of the norm of the data and the norm of the derivatives. If $\alpha = 0$, the distance is computed on the data, while if $\alpha = 1$ it is based on the derivatives. Intermediate values balance these two contributions: increasing the relevance given to the derivatives emphasizes the shapes of the peaks, while data are more related to the height.
- w is a weight coefficient, essential to make the norm of the data and of the derivatives comparable. It can be user defined or computed inside the `cluster_peak` method. A suggestion for computing the weight w is given in Section 4.1.

4.1 Definition of weight in the distance function

If not provided, the method `cluster_peak` defines w as

$$w = \text{median}\left(\frac{d_0(s_i, s_j)}{d_1(s_i, s_j)}\right)$$

where $d_0(i, j) = \|s_i^e - s_j^e\|_p$ and $d_1(i, j) = \|(s_1^e)' - (s_2^e)'\|_p$. These matrices can be automatically computed with the `distance_peak` function.

```
> # compute the weight from the first 10 peaks
>
> dist_matrix <- distance_peak(peaks.summit)
> # dist matrix contains the two matrices d_0(i,j)
> # and d_1(i,j), used to compute w
> names(dist_matrix)

[1] "dist_matrix_d0" "dist_matrix_d1"

> ratio_norm <- dist_matrix$dist_matrix_d0 / dist_matrix$dist_matrix_d1
> ratio_norm_upper_tri <- ratio_norm[upper.tri(ratio_norm)]
> summary(ratio_norm_upper_tri)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  28.21   91.36  117.79  124.07  148.17  330.95

> # suggestion: use the median as weight
> w <- median(ratio_norm_upper_tri)
> w

[1] 117.7907

>
```

4.2 The `cluster_peak` method

The two main characteristics of the k-mean alignment algorithm used in *FunChIP* are the distance function d (defined in Equation (2)), used to compute the distance between curves, and the set of warping functions $\mathcal{W}$ (defined in Equation (1)) considered for the alignment. The `cluster_peak` method applies the k-mean alignment algorithm with these specifications to the set of peaks stored in the *GRanges* object. In particular, the parameters `weight`[2], `alpha` and `p` define the distance used in the algorithm, while `t.max` sets the maximum shift of each peak in each iteration (in this particular case, q of Equation (1) does not vary in the whole $\mathbb{Z}$ but $q \in \{-$`t.max`$\cdot|U|, \ldots, +$`t.max`$\cdot|U|\}$, with $|U|$ being the maximum width of the spline approximation of the peaks.

[2] `weight` can be also set to `NULL` and it will be automatically computed as specified in Section 4.1. To save computational time, it is generally computed on a random sub-sample of data, whose size is set by the `subsample.weight` parameter.

Given a *GRanges* `GR` containing the metadata columns computed from the `smooth_peak` method, `cluster_peak` applies the k-mean alignment algorithm for all the values of k between 1 and `n.clust` (parameter of the function).

The algorithm can be run in parallel, setting to `TRUE` the `parallel` argument of the method and providing the number of cores `num.cores`. With these settings, the different applications of the algorithm, corresponding to different numbers of clusters, are executed in parallel.

As detailed in the help, the `cluster_peak` method has 2 outputs:

- The *GRanges* object, updated with new metadata columns associated to the classification. In particular, in the general case of classification with and without alignment, columns with information on the clustering of the peaks (`cluster_shift` and `cluster_NOshift`), the corresponding shifts (`coef_shift`) and the distances from the template of the clusters (`dist_shift` and `dist_NOshift`) are added.
- The graph of the global distance within clusters[3] as a function of the number of clusters (if `plot.graph.k = TRUE`). This plot can be used to identify the optimal number of clusters of the partition of the data set and the effect of the alignment procedure. In particular, if `shift = NULL`, the algorithm is run both with and without alignment and two trend lines are plotted: the black line corresponds to the global distance without the shift, and the red line corresponds to the distance obtained with alignment. If `shift` is set to `TRUE` or `FALSE`, just one type of algorithm is run and the correspondent curve is plotted. For each trend line, this graph allows the identification of the optimal value of the number of clusters: for this value, the distance significantly decreases with respect to the lower values of k, and negligibly increases with respect to higher values of k (elbow in the line). The gap between the red and the black line, instead, shows the decrease of the distance when the shift is introduced.

[3] sum over all the peaks of the distance of each peak from the corresponding template.

It is relevant to point out that the algorithm can be run both on the original data and on the scaled peaks, depending on the focus of the analysis. The logic paramter `rescale` allows the user to choose.

```
> # classification of the smooth peaks in different
> # numbers of clusters, from 1 ( no distinction, only shift )
> # to 4.
>
> # here the analysis is run on the spline approximation
> # without scaling
> peaks.cluster <- cluster_peak(peaks.summit, parallel = FALSE ,  seeds=1:4,
+                                  n.clust = 1:4, shift = NULL,
+                                  weight = 1, alpha = 1, p = 2, t.max = 2,
+                                  plot.graph.k = TRUE, verbose = FALSE)
> head(peaks.cluster)

GRanges object with 6 ranges and 12 metadata columns:
      seqnames          ranges strand |
         <Rle>       <IRanges>  <Rle> |
  [1]    chr18 3337524-3338025      * |
  [2]    chr18 4369126-4369352      * |
  [3]    chr18 4375448-4375883      * |
  [4]    chr18 4715744-4716162      * |
  [5]    chr18 4716374-4716597      * |
  [6]    chr18 4921270-4921506      * |


  [1] c(7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8
  [2]
  [3]                      c(8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 11, 11, 11, 1
  [4]
  [5]
  [6]


  [1] c(0.107576984737605, 0.115943638884655, 0.124571296466484, 0.133463063982055, 0.142622047930333, 0.152
  [2]
  [3]
  [4]
  [5]
  [6]


  [1] c(0.00823718792931595, 0.00849663811461222, 0.00875919479887256, 0.00902485798209696, 0.009293627664285
  [2]
  [3]
  [4]
  [5]
  [6]
      width_spline start_spline end_spline summit_spline cluster_NOshift
         <integer>    <numeric>  <numeric>     <integer>          <list>
  [1]          593      3337483    3338075           444   c(1, 1, 1, 1)
```

```
  [2]          335      4369075    4369409           186   c(1, 2, 2, 2)
  [3]          508      4375417    4375924           174   c(1, 2, 2, 3)
  [4]          523      4715698    4716220           310   c(1, 2, 2, 3)
  [5]          315      4716303    4716617           121   c(1, 1, 1, 1)
  [6]          332      4921234    4921565           169   c(1, 1, 2, 2)
                                                                  dist_NOshift
                                                                        <list>
  [1] c(0.624908234181997, 0.225283720971178, 0.225283720971178, 0.225283720971178)
  [2] c(0.436709932429605, 0.517291820291409, 0.432369848037446, 0.283883228538212)
  [3] c(0.378181153745047, 0.485135818958827, 0.382850137813199, 0.485135818958827)
  [4]     c(0.539505144001724, 0.5547517314999, 0.578858851697885, 0.5547517314999)
  [5] c(0.534987295842725, 0.217738151115933, 0.217738151115933, 0.217738151115933)
  [6]                 c(0.293729665431876, 0.414829041487123, 0.292960162760959, 0)
      cluster_shift          coef_shift
             <list>              <list>
  [1] c(1, 1, 1, 1)    c(6, 18, 23, 28)
  [2] c(1, 2, 2, 2)   c(16, 21, 22, 14)
  [3] c(1, 2, 2, 4)  c(-11, -3, -5, -3)
  [4] c(1, 2, 2, 4) c(-13, -6, -10, -6)
  [5] c(1, 1, 1, 1)   c(-14, 9, 14, 19)
  [6] c(1, 1, 2, 2)      c(1, 14, 6, 0)
                                                                     dist_shift
                                                                         <list>
  [1]  c(0.611798948745457, 0.217935827035599, 0.217935827035599, 0.217935827035599)
  [2]  c(0.296361824245803, 0.492957766323058, 0.252045266845235, 0.185082312495507)
  [3]  c(0.371425430479149, 0.382138694017752, 0.372516258232315, 0.387954685125311)
  [4]   c(0.531584502689481, 0.556766222517974, 0.55605805078994, 0.440775393310266)
  [5]  c(0.528633634301522, 0.217563128269193, 0.217563128269193, 0.217563128269193)
  [6] c(0.273367776201618, 0.413050475331536, 0.273737759495666, 0.0893377359638826)
  -------
  seqinfo: 20 sequences from an unspecified genome; no seqlengths

>
```

```
> # here the analysis is run on the spline approximation
> # with scaling
> peaks.cluster.scaled <- cluster_peak(peaks.summit.scaled, parallel = FALSE ,  seeds=1:4,
+                                      n.clust = 1:4, shift = NULL,
+                                      weight = 1, alpha = 1, p = 2, t.max = 2,
+                                      plot.graph.k = TRUE, verbose = FALSE,
+                                      rescale = TRUE)
> head(peaks.cluster.scaled)

GRanges object with 6 ranges and 15 metadata columns:
      seqnames          ranges strand |
         <Rle>       <IRanges>  <Rle> |
  [1]    chr18 3337524-3338025      * |
  [2]    chr18 4369126-4369352      * |
  [3]    chr18 4375448-4375883      * |
  [4]    chr18 4715744-4716162      * |
  [5]    chr18 4716374-4716597      * |
  [6]    chr18 4921270-4921506      * |
```

```
[1] c(7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
[2]
[3]                       c(8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 11, 11, 11, 1
[4]
[5]
[6]


[1] c(0.107576984737605, 0.115943638884655, 0.124571296466484, 0.133463063982055, 0.142622047930333, 0.152
[2]
[3]
[4]
[5]
[6]


[1] c(0.00823718792931595, 0.00849663811461222, 0.00875919479887256, 0.00902485798209696, 0.009293627664285
[2]
[3]
[4]
[5]
[6]
    width_spline start_spline end_spline
       <integer>    <numeric>  <numeric>
[1]          593      3337483    3338075
[2]          335      4369075    4369409
[3]          508      4375417    4375924
[4]          523      4715698    4716220
[5]          315      4716303    4716617
[6]          332      4921234    4921565


[1]                                                                  c(0.00263609710663517, 0.0030440063908373
[2]              c(0.00122834484766426, 0.00147221033932346, 0.00172653795382648, 0.00199155049789865, 0.00
[3]            c(0.000864564256647586, 0.00120333309941394, 0.00156495680464632, 0.00195011992527804, 0.002
[4]            c(0.000788310894250635, 0.000956445833827378, 0.00113504914292024, 0.0013243784475671, 0.001
[5] c(0.00235269135248368, 0.00273549871311195, 0.00313329749625751, 0.00354638183650247, 0.00397503469483
[6]         c(0.00139356234463668, 0.0016220858943517, 0.00186006713373021, 0.00210769852286914, 0.0023651


[1]
[2]                                  c(0.000238708662407768, 0.000249059436995862, 0.000259632908095442, 0
[3]                              c(0.000327571761954349, 0.000350081098788865, 0.000373281486886386, 0.0003
[4]                                                                  c(0.000162985979570835, 0.0001733265
[5]       c(0.000375410625363164, 0.00039025358389014, 0.00040539347039775, 0.00042082190469042, 0.0004365
[6] c(0.000223858745449888, 0.000233220374263462, 0.000242774124776864, 0.000252521012181378, 0.0002624620
    summit_spline_rescaled summit_spline cluster_NOshift
                 <integer>     <integer>          <list>
[1]                    227           444   c(1, 1, 1, 1)
```

```
[2]                   168           186   c(1, 2, 2, 2)
[3]                   104           174   c(1, 2, 3, 4)
[4]                   180           310   c(1, 2, 2, 4)
[5]                   117           121   c(1, 2, 3, 3)
[6]                   155           169   c(1, 2, 2, 2)
                                                                                dist_NOshift
                                                                                     <list>
[1]  c(0.0135465516288349, 0.00634872462194527, 0.00634872462194527, 0.00726552411035082)
[2]       c(0.00314304602114034, 0.00377608653776684, 0.00260029589647, 0.00260029589647)
[3] c(0.00668285220715305, 0.00627910073393782, 0.00460131880636674, 0.00446383231788615)
[4] c(0.00461630755109992, 0.00470065453190587, 0.00449050254427536, 0.00236167494392528)
[5] c(0.00446076173955213, 0.00382455339523424, 0.00269385225273107, 0.00266812526234695)
[6] c(0.00130263542562362, 0.00127949199373814, 0.00132011451049066, 0.00132011451049066)
    cluster_shift          coef_shift
           <list>              <list>
[1] c(1, 1, 1, 1)    c(1, 36, 41, 41)
[2] c(1, 2, 2, 2)     c(7, 8, 10, 10)
[3] c(1, 2, 3, 3) c(-17, -13, -4, -4)
[4] c(1, 2, 2, 4)   c(-6, -2, -1, -2)
[5] c(1, 2, 3, 3)   c(-13, -11, 2, 2)
[6] c(1, 2, 2, 2)   c(-3, -3, -2, -2)
                                                                              dist_shift
                                                                                  <list>
[1]  c(0.0136019700774668, 0.00529868829574966, 0.00529868829574966, 0.00529868829574966)
[2] c(0.00288555205001011, 0.00288555205001011, 0.00196125806891698, 0.00196125806891698)
[3]                  c(0.00555633810032155, 0.00561814086680531, 0.00411761765869642, 0)
[4] c(0.00469497672879888, 0.00474230622233802, 0.00488326800693634, 0.00234934010901782)
[5]    c(0.00330416023152333, 0.00331770403459043, 0.0057934253968632, 0.002767058747939)
[6]                                  c(0.00121592857872764, 0.00123025283644485, 0, 0)
-------
seqinfo: 20 sequences from an unspecified genome; no seqlengths

>
```

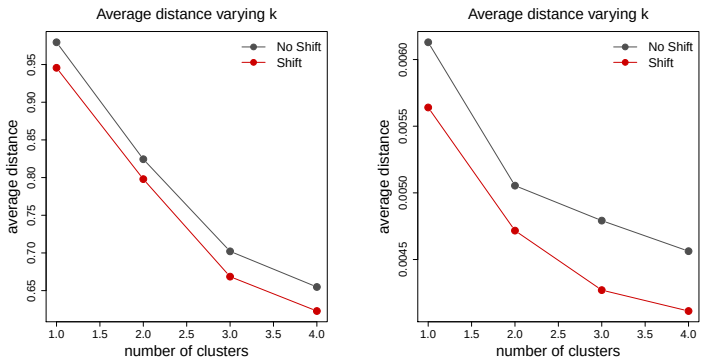

Figure 4: Global distance within clusters. Global distance of the peaks form the corresponding template, as a function of the number of clusters k
In the left panel the graph for the original spline approximation, while in the right panel restuls are relative to the scaled approximation.

The particular case of k-mean alignment with $k = 1$ clusters can be used to highlight the effects of the alignment of the peaks: no grouping is performed, just the shifts are computed. Therefore, the decrease of the global distance is solely due to a change of the abscissae of the functions, as Figure 3 shows. Moreover, focusing for exemple on the first panel of Figure ??, we can deduce that, for this case

- the alignment can effectively decrease the distance, for exemple for $k = 6$, the gap between red and black line is significant;
- the alignment may change the optimal k: looking at the black line, one would have chosen $k = 4$, while the red line suggests $k = 3$ is the best choice. With the introduction of the shifts, data which are originally different becomes more similar and therefore one less cluster is needed; it has to be noted that the distance obtained with $k = 3$ and alignment is very similar to the one obtained with $k = 4$ and no alignment.

Therefore, for this case, one possible classification is the one associated to $k = 3$ with shift. On the contray for the scaled peaks the value of k we can identify as crucial is $k = 2$ and shift is relevant since it reduces a lot the global distance. In Section 4.3 we present two quantitative methods to isolate the most suitable value of k: the first quantifies the elbow rule we have presented here and the second that computes the Silhouette index to consider the global structure of the dataset. The results for this specific number of clusters can then be selected with the `choose_k` method:

```
> # select the results for k = 3 with alignment
> peaks.classified.short <- choose_k(peaks.cluster, k = 3,
+                                    shift = TRUE, cleaning = TRUE)
> head(peaks.classified.short)

GRanges object with 6 ranges and 1 metadata column:
      seqnames          ranges strand |   cluster
         <Rle>       <IRanges>  <Rle> | <numeric>
  [1]    chr18 3337524-3338025      * |         1
  [2]    chr18 4369126-4369352      * |         2
  [3]    chr18 4375448-4375883      * |         2
  [4]    chr18 4715744-4716162      * |         2
  [5]    chr18 4716374-4716597      * |         1
  [6]    chr18 4921270-4921506      * |         2
  -------
  seqinfo: 20 sequences from an unspecified genome; no seqlengths

> peaks.classified.extended <- choose_k(peaks.cluster, k = 3,
+                                    shift = TRUE, cleaning = FALSE)
> # and for the scaled version for k =2 and alignment
>
> peaks.classified.scaled.short <- choose_k(peaks.cluster.scaled, k = 2,
+                                    shift = TRUE, cleaning = TRUE)
> head(peaks.classified.scaled.short)

GRanges object with 6 ranges and 4 metadata columns:
      seqnames          ranges strand |
         <Rle>       <IRanges>  <Rle> |
  [1]    chr18 3337524-3338025      * |
  [2]    chr18 4369126-4369352      * |
  [3]    chr18 4375448-4375883      * |
  [4]    chr18 4715744-4716162      * |
```

```
  [5]    chr18 4716374-4716597      * |
  [6]    chr18 4921270-4921506      * |


  [1]                                                                  c(0.00263609710663517, 0.003044006390837
  [2]              c(0.00122834484766426, 0.00147221033932346, 0.00172653795382648, 0.00199155049789865, 0.00
  [3]            c(0.000864564256647586, 0.00120333309941394, 0.00156495680464632, 0.00195011992527804, 0.002
  [4]            c(0.000788310894250635, 0.000956445833827378, 0.00113504914292024, 0.0013243784475671, 0.001
  [5] c(0.00235269135248368, 0.00273549871311195, 0.00313329749625751, 0.00354638183650247, 0.00397503469483
  [6]         c(0.00139356234463668, 0.0016220858943517, 0.00186006713373021, 0.00210769852286914, 0.0023651


  [1]
  [2]                                  c(0.000238708662407768, 0.000249059436995862, 0.000259632908095442, 0
  [3]                              c(0.000327571761954349, 0.000350081098788865, 0.000373281486886386, 0.0003
  [4]                                                                  c(0.000162985979570835, 0.0001733265
  [5]       c(0.000375410625363164, 0.00039025358389014, 0.00040539347039775, 0.00042082190469042, 0.0004365
  [6] c(0.000223858745449888, 0.000233220374263462, 0.000242774124776864, 0.000252521012181378, 0.0002624620
      summit_spline_rescaled   cluster
                   <integer> <numeric>
  [1]                    227         1
  [2]                    168         2
  [3]                    104         2
  [4]                    180         2
  [5]                    117         2
  [6]                    155         2
  -------
  seqinfo: 20 sequences from an unspecified genome; no seqlengths
> peaks.classified.scaled.extended <- choose_k(peaks.cluster.scaled, k = 2,
+                                  shift = TRUE, cleaning = FALSE)
```

The `choose_k` method allows, respectively, to remove all the metadata columns computed by *FunChIP* and obtain a *GRanges* equivalent to the initial one, with an extra the metadata column `cluster` containing the classification labels (`cleaning = TRUE`), or a *GRanges* retaining all the details of the prepossessing and clustering (all the previously described metadata columns), with the extra column `cluster` (`cleaning = FALSE`).

4.3 The choice of the final classification with the bending index and the Silhouette plot

In this section we present two analyses which could be useful to identify the most suitable value of the number of clusters.

The first analysis quantifies the elbow rule we proposed in the previous section. The function `bending_index` computes for each value of k in the range $[2 : K-1]$, with K the maximum number of clusters allowed in `cluster_peak`, an index that quantifies the bending of the curve. The higher this index, the more appropriate is the correspondent value of k. The index is computed as the distance of the point P in k of the global distance function, normalized to its maximum value, ($P = (k, \tilde{D}(k))$) from the line r passing by the point in $k-1$ and in

$k+1$ ($r = r((k-1, \tilde{D}(k-1)), (k+1, \tilde{D}(k+1)))$). The normalized distance function is

$$\tilde{D}(k) = \frac{D(k)}{\max_{k=1:K} D(k)}$$

with $D(k)$ being the global distance of the set of classified peaks, when divided into k clusters:

$$D(k) = \sum_{i=1}^{n} d^{\star}_{i,k}.$$

In particular, $d^{\star}_{i,k}$ is the distance of each peak i from the center of the correspondent cluster $c_{k^{\star}_i}$. Then, the index can be written as:

$$r(k) = \text{dist}((k, \tilde{D}(k)), r((k-1, \tilde{D}(k-1)), (k+1, \tilde{D}(k+1))))$$

```
> # here we compute the bending index for the classification
> # provided for the non scaled dataset
>
> index <- bending_index(peaks.cluster, plot.graph.k = FALSE)
> index

$index_shift
          2           3
0.009578264 0.044113662

$index_NOshift
          2           3
0.01664078 0.03811411

> # and then for the scaled dataset
> index_scaled <- bending_index(peaks.cluster.scaled, plot.graph.k = FALSE)
> index_scaled

$index_shift
          2           3
0.04202466 0.02561751

$index_NOshift
          2           3
0.065931403 0.002706642

>
```

The function returns the value of the bending index varying k for both the classification with `index_shift` and without `index_NOshift` alignment, if both classifications are stored in the `object`. It can also show the plot of the global distance curves of Figure ??, if `plot.graph.k = TRUE`.

Moreover, we present another method to evaluate the choice the proper number of clusters, which takes into account the global set of data. This method is based on the defintion of the Silhouette index of [6] and, for the peak i, the index is computed as

$$s(i) = \frac{a(i) - b(i)}{\max(a(i), b(i))},$$

with $a(i)$ being the average dissimilarity of peak i with all other data within the same cluster and $b(i)$ the lowest average dissimilarity of i to any other cluster, of which i is not a member. To compute the $b(i)$ value it is necessary to compare peaks belonging to different clusters. In case of classification with alignment, to define the distance of peaks of different clusters it is necessary first to align the two clusters. To perform this global alignment in an efficient way, we align the centers of the clusters and then use the estimated shift coefficient to align all the peaks of the two clusters.

```
> sil <- silhouette_plot(peaks.cluster, p=2, weight = 1, alpha = 1,
+                        rescale = FALSE, t.max = 2)

[1] "Silhouette for the non aligned peaks"
[1] "Silhouette for the aligned peaks"
```

```
> sil <- silhouette_plot(peaks.cluster.scaled, p=2, weight = 1, alpha = 1,
+                        rescale = FALSE, t.max = 2)

[1] "Silhouette for the non aligned peaks"
[1] "Silhouette for the aligned peaks"
```

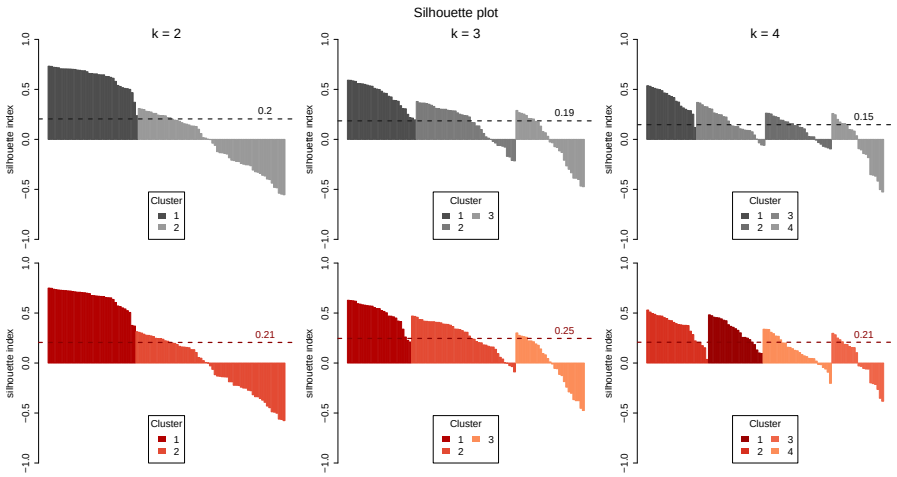

Figure 5: **Silhouette graph for the classification of unscaled peaks. In the top panel, Silhouette plots for the classification without alignment, as function of the number of clusters; in the bottom panel, Silhouette plots for the classification with alignment**

Th plots shown in Figure ?? represent the Silhouette index for all the peaks divided in the clusters, together with the global average Silhouette index. The average Silhouette index is maximum when the clustering is separating the data in the most suited manner and then this index has to be maximized to choose the best classification. In the case here presented here, the best classification according to this criteria is the classification with alignment and $k = 2$ or $k = 3$. Considering both Silhouette and the bending criteria, in the next section we present the results of the classification with alignement and $k = 3$ clusters.

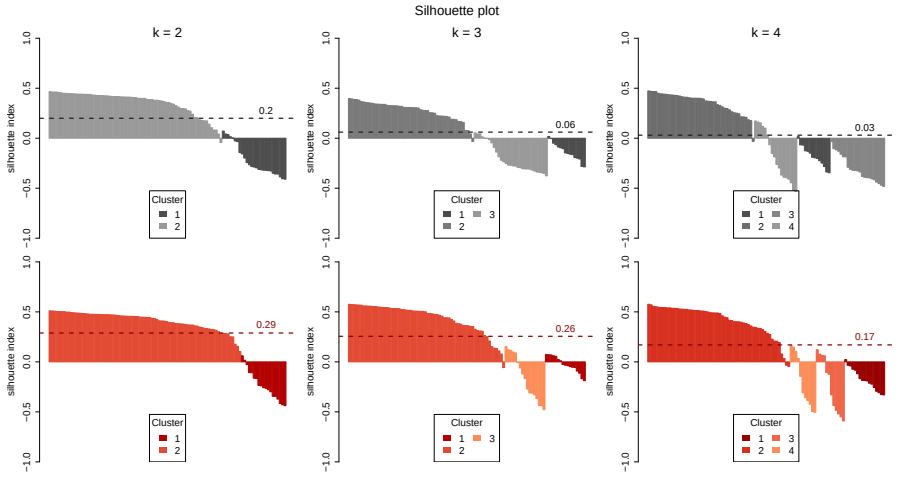

Figure 6: **Silhouette graph for the classification of scaled peaks. In the top panel, Silhouette plots for the classification without alignment, as a function of the number of clusters; in the bottom panel, Silhouette plots for the classification with alignment**

5 Visualization of the peaks

The `plot_peak` method is a very flexible function for displaying ChIP-Seq peaks. In particular, it allows to plot the raw counts obtained by the `pileup_peak` method, as in Figure 7. It can also plot smoothed peaks, possibly centered around the summit, as in Figure **??**, or scaled as in Figure **??** and centerd.

```
> # plot of the first 10 peaks (raw data)
> par(mar=c(4.5,5,4,4))
> plot_peak(peaks, index = 1:10, line.plot = 'count',
+           cex.main = 2, cex.lab = 2, cex.axis =2)
```

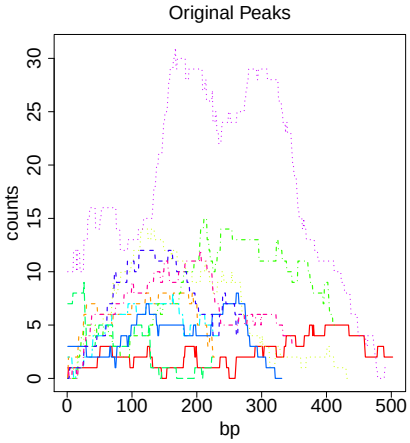

Figure 7: **10 peaks: counts**
Representation of the original peaks as raw counts (no smoothing).

```
> # plot of the smoothed approximation of the first 10 peaks
> par(mar=c(4.5,5,4,4))
> plot_peak(peaks.smooth, index = 1:10, line.plot = 'spline',  cex.main = 2,cex.lab = 2, cex.axis =2)
>
```

```
> # plot of the smoothed approximation of the first 10 peaks,
> # centering peaks around their summits
> par(mar=c(4.5,5,4,4))
> plot_peak(peaks.summit, index = 1:10, line.plot = 'spline', cex.main = 2,cex.lab = 2, cex.axis =2)
```

```
> # plot of the smoothed approximation of the first 10 peaks;
> # the scaled functions are plotted.
> #
> par(mar=c(4.5,5,4,4))
> plot_peak(peaks.smooth.scaled, index = 1:10,
+           line.plot = 'spline', rescale = TRUE,
+            cex.main = 2,cex.lab = 2, cex.axis =2)
>
```

```
> # plot of the scaled approximation of the first 10 peaks,
> # centering peaks around their summits
> par(mar=c(4.5,5,4,4))
> plot_peak(peaks.summit.scaled, index = 1:10,
+           line.plot = 'spline', rescale = TRUE,
+            cex.main = 2,cex.lab = 2, lwd = 2,cex.axis =2)
```

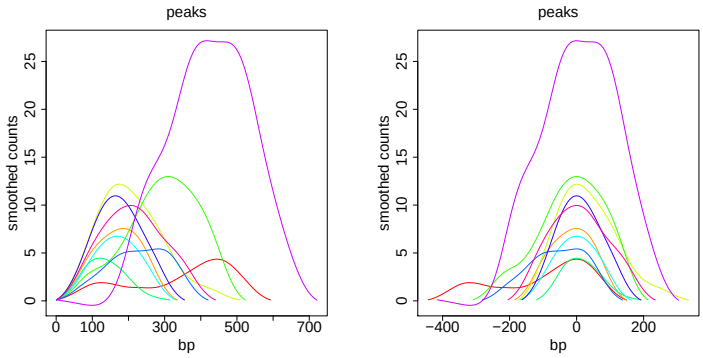

Figure 8: 10 spline-smoothed peaks. In the left panel, smoothed peaks are shown, while in the right panel the same peaks are centered around their summits

From the comparison of Figure ?? and Figure ?? it is clear how the scaling affects the shape of splines. Now peaks are no more related to the magnitude, but just to their shapes.

Moreover, plotting both raw counts and spline is also possible: Figure 10 shows a single peak in its raw and smoothed version. This representation is useful to check the accuracy of the smoothing and, if needed, manually set the λ parameter of the spline approximation.

```
> # plot of a peak comparing its raw structure and
> # its spline-smoothed version.
> par(mar=c(4.5,5,4,4))
> plot_peak(peaks.summit, index = 3, lwd =2, line.plot = 'both', col = 'darkblue', cex.main = 2,cex.lab = 2,
```

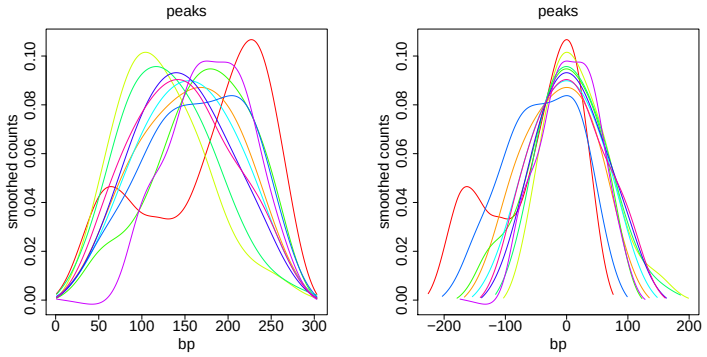

Figure 9: 10 spline-smoothed and scaled peaks. In the left panel, scaled peaks are shown, while in the right panel the same peaks are centered around their summits

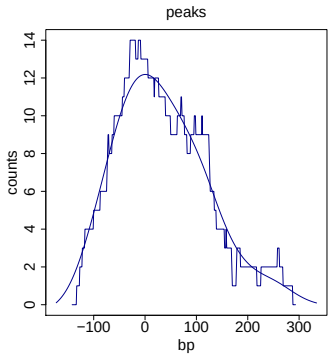

Figure 10: Read coverage and spline approximation
Plot of the original read coverage of a peak and its smoothing (spline approximation), centered around the summit.

```
> # plot of the results of the kmean alignment.
> # Peaks are plotted in three different panels
> # according to the clustering results.
>
> plot_peak(peaks.cluster, index = 1:100, line.plot = 'spline',
+           shift = TRUE, k = 3, cluster.peak = TRUE,
+           col = topo.colors(3),  cex.main = 2,cex.lab = 2, cex.axis =2)
```

```
> # plot of the results of the kmean alignment.
> # Scaled peaks are plotted in three different panels
> # according to the clustering results.
>
> plot_peak(peaks.cluster.scaled, index = 1:100, line.plot = 'spline',
+           shift = TRUE, k = 2, cluster.peak = TRUE, rescale = TRUE,
+           col = heat.colors(2), cex.main = 2,cex.lab = 2, cex.axis =2)
>
```

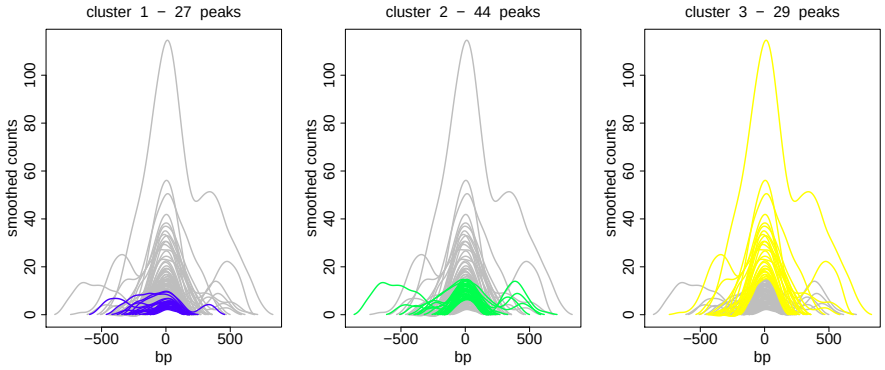

Figure 11: **Peaks divided in the three clusters The same spline-smoothed peaks are plotted in grey, and for each panel the peaks in the corresponding cluster are colored to show their different shapes**
Peaks are aligned with the shift coefficients obtained by the k-mean alignment algorithm.

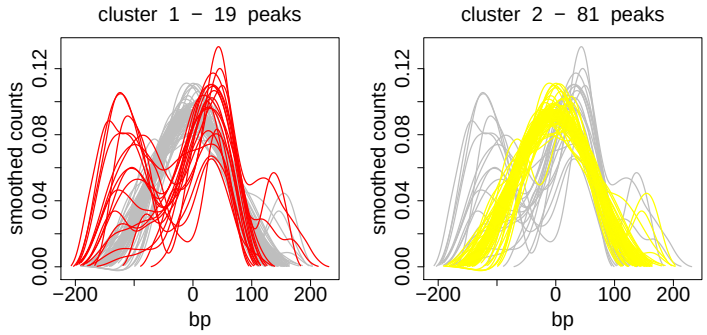

Figure 12: **Scaled peaks divided in the two clusters The same spline-smoothed scaled peaks are plotted in grey, and for each panel the peaks in the corresponding cluster are colored to show their different shapes**
Peaks are aligned with the shift coefficients obtained by the k-mean alignment algorithm.

Finally, the `plot_peak` method allows to plot the results of the clustering via the k-mean alignment. In Figure 11 and Figure 12, smoothed and scaled peaks are divided into the three clusters and plotted with the optimal shift obtained with the alignment.

References

[1] J.O. Ramsay and B.W. Silverman. *Functional Data Analysis.* Springer, New York, NY, 2005.

[2] Laura M. Sangalli, Piercesare Secchi, Simone Vantini, and Valeria Vitelli. K-mean alignment for curve clustering. *Computational Statistics & Data Analysis*, 54(5):1219 – 1233, 2010. URL: http://www.sciencedirect.com/science/article/pii/S0167947309004605, doi:http://dx.doi.org/10.1016/j.csda.2009.12.008.

[3] Laura M. Sangalli, Piercesare Secchi, and Simone Vantini. Analysis of aneurisk65 data: k-mean alignment. *Electron. J. Statist.*, 8(2):1891–1904, 2014. URL: http://dx.doi.org/10.1214/14-EJS938A, doi:10.1214/14-EJS938A.

[4] Mara Bernardi, Laura M. Sangalli, Piercesare Secchi, and Simone Vantini. Analysis of juggling data: An application of k-mean alignment. *Electron. J. Statist.*, 8(2):1817–1824, 2014. URL: http://dx.doi.org/10.1214/14-EJS937A, doi:10.1214/14-EJS937A.

[5] Mirco Patriarca, Laura M. Sangalli, Piercesare Secchi, and Simone Vantini. Analysis of spike train data: An application of k-mean alignment. *Electron. J. Statist.*, 8(2):1769–1775, 2014. URL: http://dx.doi.org/10.1214/14-EJS865A, doi:10.1214/14-EJS865A.

[6] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65, 1987. URL: http://www.sciencedirect.com/science/article/pii/0377042787901257, doi:http://dx.doi.org/10.1016/0377-0427(87)90125-7.