# R package 'Pasha' (Preprocessing of Aligned Sequences from HTS Analyses)

Romain Fenouil; Lionel Spinelli

This vignette gives an overview of the 'typical' application of Pasha package. For computing time and disk space constraints, all the scripts are illustrated on a minimal reference dataset which is hand-made, hence totally artificial. All examples however are selected for their biological relevance and can be used on real datasets.

## 1 What is in the package ?

Pasha package stands for 'Preprocessing of Aligned Sequences from HTS Analyses'. It provides several functions dedicated for transforming read alignments in enrichment scores (piling) and more importantly, compiles all these tools in an automated pipeline. It has been designed to provide a versatile behaviour depending on users needs.

## 2 What is this document done for ?

This document is a quick overview of the pipeline and helps users to understand how to use the pipeline and some underlying functions of the package.

## 3 Pipeline description

The pipeline is described in the help page of 'processPipeline' and in the corresponding publication. Briefly, it aims at transforming files generated by Chromatin Sequencing experiments (ChIP-seq FAIRE-seq, MNAse-seq) into accurate enrichment scores (piled) that can be used for further analyses.
After sequencing, most experimental setups require to align the sequenced read to a reference genome using one of the available dedicated tools (ie. bowtie). For chromatin and epigenome studies, several techniques (chIP-seq, MNase-seq, FAIRE-seq...) allow to isolate and map reads that correspond to chromatin regions with specific characteristics or to specific factors bound to chromatin. Once these reads are correctly mapped to the reference genome, it gives an interesting overview of enriched chromosomic regions. However, several issues avoid a direct interpretation of these datasets.
First, the pre-sequencing procedure, which includes genetic material amplification can lead to over-amplification of some fragments. This concern which is known to generate a fragment over-representation in some regions (artificial reads piling) is often refered as 'PCR-artefacts'.
Second, if HTS technique allows the sequencing of (hundreds of) millions of DNA fragments, it can only provide sequence information for the few first basepairs of each read (25bp to few hundreds depending on the technology). Thus the aligned reads only represent a sub-fraction of the sequenced DNA fragments and an important part of the information is missing when computing the read coverage without an appropriate preprocessing of these reads.
Finally, for some analyses (average profiles, clustering...) and for browsing datasets, a piled (see several strategies proposed by the package, classic or midpoint) representation which gives direct scores for each genomic location is preferable for the ease of parsing and score computing.
Technical issues are also to be taken in account. Each mapping program has a different proprietary format for output files. Even if a standard has been defined, some specific formats keep on being used. The pipeline is able to read the standard format but also relies on previously developped packages (ShortRead) to handle various specific input formats (Solexa, Bowtie, SOAP etc...).

All these concerns and several other options are addressed by dedicated functions provided in this package. The pipeline is an 'all-included' tool that helps to automatize these processings for several and various datasets. Using the pipeline will also give precious piece of information on processed datasets by writing logs and figures summarizing the computation progress.

The pipeline also include some innovative options which give some added value as compared to concurrent tools. Namely :

- one can take in account reads that were mapped in several locations using one of the two dedicated algorithms
- seamless processing of paired-ends sequencing experiments
- it can generate midpoint pileups (crucial to represent nucleosome occupancy as opposed to nucleosome density observed with classic pileup strategy)
- it includes an option to generate subgroups of reads from an experiment based on the insert-size (in case of paired-ends experiments) or read size (for single-end experiment, useful for shortRNA processing after adapters trimming)
- it gives advanced reports on experiment characteristic and reads distribution
- it uses an original algorithm to estimate the fragments size in-silico (it favors estimation in enriched regions as opposed to cross-correlation)
- it is able to work on several cores at the same time (this option is memory consuming, use with care)

## 4 Starting the pipeline

For an automatized use of the pipeline, a specific function called 'processPipeline' is available to users. For basic options, few arguments relative to the experiment and the desired processings have to be specified. Once started, the pipeline will run and generate results and figures in specified folders and subfolders until all options/datasets are processed.

### 4.1 Arguments

processPipeline provides a large set of arguments divided in different classes/types. There are four different kind of arguments : 'I/O general arguments', 'complex arguments', 'single arguments', and 'list parameters'.

### 4.2 inputFileList

inputFileList is the first and most important argument. It describe the experiments/samples and let the user specify the location of aligned files and some other parameters. This argument must be a named list, the pipeline will use each elements name to identify experiment and name output files. Each element is a nested list which describe the experiment. This list must provide these informations (information available from 'processPipeline' help page) :

**folderName** is the full path to the folder containing the file describing aligned reads.

**fileName** is the filename containing the aligned data, typically the output from a program of read alignment.

**fileType** defines the format in which the data is stored in the file. The pipeline handle preferentially standard BAM files ('BAM' value), but can also read several proprietary format by using the ShortRead library (see readAligned function from ShortRead library for a complete list of supported formats).

**chrPrefix** is a regular expression string defining the prefix concatenated to each chromosome name. Typically this value is 'chr' but can be different depending on how the reference genome for alignment has been created and named.

**chrSuffix** see 'chrPrefix' argument. Typically, this is defined as an empty string ''.

**pairedEnds** a logical value specifying is the experiment should be considered as single or paired -end. In the latter case, only BAM files are supported.

**midPoint** a logical value. If TRUE a specific piling method will be applied to the reads and the output files will have '_midpoint' suffix.

A simple example of list construction could be :

```
# Define an experiment description list with a classic epigenetic mark
# and one MNase experiment that will be seen as nucleosome 'density' (typical
# piling) or nucleosome 'positionning' (midpoint piling)

myExperimentsList=list();

myExperimentsList[["mES_H3K4me3"]]=list('folderName'="/home/exp",
        'fileName'="SRR432543.BAM",
        'fileType'="BAM",
        'chrPrefix'="chr",
        'chrSuffix'="",
        'pairedEnds'=FALSE,
        'midPoint'=FALSE);

myExperimentsList[["mES_MNase"]]=list('folderName'="/home/exp",
        'fileName'="SPT543426.BAM",
        'fileType'="BAM",
        'chrPrefix'="chr",
        'chrSuffix'="",
        'pairedEnds'=TRUE,
        'midPoint'=FALSE);

myExperimentsList[["mES_MNase_MIDPOINT"]]=list('folderName'="/home/exp",
        'fileName'="SPT543426.BAM",
        'fileType'="BAM",
        'chrPrefix'="chr",
        'chrSuffix'="",
        'pairedEnds'=TRUE,
        'midPoint'=TRUE);
```

## 4.3 'I/O general arguments'

### 4.3.1 Type

These arguments expect single values relative to the Input or Outputs that will be read or generated.

### 4.3.2 Description

The first two ones are 'resultSubFolder' and 'reportFilesSubFolder' which ask the user where to write results and reports respectively. 'resultSubFolder' is relative to the experiment aligned file folder. Thus, the user must not give an absolute path, typically a simple name like 'result' is expected. 'reportFilesSubFolder' is relative to 'resultSubFolder'. As above, typically a simple name like 'reportFiles' is expected.

The six other parameters are single logical values that define the type of output format for results. The pipeline is able to generate WIG variable step, WIG fixed step with binning of values, Bedgraph, BIGWIG and GFF compatible format with binning of values. Typical users will need WIGvs (variable steps) and WIGfs (WIG fixed steps) for browsing purpose and bioinformatic analyses. WIGvs has the main advantage of being more compact for files with a lot of equal consecutive values (0's in most cases) and allows to give full resolution (1bp) result files in a reasonable amount of disk space. Depsite the waste of disk space, WIGfs format is very convenient for simple parsing and is often coupled to binning (resolution reduction, ie. binsize >1) in order to reduce files size. The package also provides individual functions for reading and writing from/to WIGfs files (see readWIG, writeWIG). BIGWIG format is converted internally from WIGvs. Be aware that 'compatibilityOutputWIG' generates non-standard file format (see documentation).

In summary, these parameters can be defined as following :

```
# name of the folder that will contain results
resultSubFolder = "Results"
# name of the folder that wil contain logs
reportFilesSubFolder = "ReportFiles"
# generate results as WIG fixed steps
WIGfs = FALSE
# generate results as WIG variable steps
WIGvs = TRUE
# generate results as GFF files
GFF = FALSE
# generate results as Bedgraph files
BED = FALSE
# generate results as BIGWIG files
BIGWIG = FALSE
# compatibility mode for WIG files : keep to FALSE ! (see documentation)
compatibilityOutputWIG = FALSE
```

## 4.4 'complex arguments'

### 4.4.1 Type

These arguments provide flexibility in order to specify values for all samples or individual ones. For each sample, user can use a single value or a vector of values. Each value will be used sequentially by the pipeline (internal loops) to generate results identified with different filenames. Alternatively the user can attribute arguments manually (single or several values) to each experiment individually using a named list. When several values are provided for a sample, the internal looping mechanism takes advantage of previously computed results. This sequential automatization is therefore more efficient than starting the pipeline several times with different parameters.

For instance a user can define following arguments as followsing (see '?processPipeline' for more examples, specially elongationSize and rangeSelection):

```
# a single value
rangeSelection = IRanges(0,-1)

# a vector of values, each value will be used sequentially for all experiments
incrArtefactThrEvery = c(10000000,NA)
elongationSize = c(0,NA)

# a list specifying arguments for individual experiments
binSize = list("mES_H3K4me3"=200, "mES_MNase"=50, "mES_MNase_MIDPOINT"=c(0,50))
```

In this specific case (see more exhaustive examples in '?processPipeline'), the script will generate 4 different results for experiment "mES_H3K4me3", 4 results for experiment "mES_MNase" and 8 result files for "mES_MNase_MIDPOINT". Resulting files will be named and stored according to the specified parameters.

### 4.4.2 Description (as in'?processPipeline')

**incrArtefactThrEvery** A numeric value or NA. A positive numeric value used to define a threshold for considering piles of reads as 'artefacts' when higher than 'number of reads in the experiment/value'. A negative value allows to directly set the threshold manually, independently of the number of reads. A NA will ignore the eventual artefactual piles. Note that for multiread experiments, this step has to be done on multireads externally (multiread_RemoveArtifact function) before starting the pipeline (see multiread chapter). See '?getArtefactsIndexes'.

**binSize** A stricly positive value (>0) defining the size of the steps in the resulting WIG file. If set as 1 there is one score per genomic coordinate (no binning).

**elongationSize** A numeric value that tells how much each read must be extended to fit the actual insert size. If NA, the elongation estimation module will be used to automatically determine the overrepresented insert size in the experiment. If 0, the reads will not be extended. For paired-end experiments, this value will override the real insert size, except if NA value is used (recommended in most cases). See '?estimateElongationSize'.

**rangeSelection** An 'IRanges' object that defines the different groups of reads to be made. In case of single-end experiments, these ranges are applied to reads size whereas in case of paired-ends experiments, the groups are made based on the insert size. An empty range (as defined for default value IRanges(0,-1)) will not perform any selection, using all the reads independtly of their size. NOTE : if one or several subrange(s) are specified, results including all reads (no range selection) will also be computed and reported (in a distinct result subfolder) for comparison purpose.

**annotationFilesGFF** A named vector of paths to GFF files (NA to ignore). If this argument and annotationGenomeFiles are provided, the pipeline will generate (for each rangeSelection and total) a pdf file summarizing reads occupancy among annotations in gff files.

**annotationGenomeFiles** Needed for plotting reads statistics on annotations (see argument 'annotationFilesGFF'). A single path to a genome reference file or the ID of a genome for which a reference is bundled in the package (to see bundled files use the command : 'dir(system.file("resources", package="Pasha"))'. IMPORTANT : in order to use a bundled reference file, one must NOT specify the file extension (examples : hg18, hg19, mm9...)

## 4.5 'single arguments'

### 4.5.1 Type

These arguments are common to all experiments that will be treated in the run and define general parameters

### 4.5.2 Description (as in '?processPipeline')

**elongationEstimationRange** A numeric vector with 3 named elements : 'mini', 'maxi', 'by'. These values define the range and the granularity that will be used for elongation estimation (See 'estimateElongationSize').

**rehabilitationStep** A character vector. Can contain 'orphans', 'orphansFromArtefacts'. See 'Rehabilitation steps' in 'details' section.

**removeChrNamesContaining** A single regular expression. Defines a pattern that will match chromosome names to be removed from the experiment. All reads aligned to the concerned chromosomes will be ignored.

**ignoreInsertsOver** A single strictly positive integer, or NA. In case of paired-ends experiments, one might want to ignore inserts above a certain size (which are probably the result of misalignment).

**nbCPUs** A single strictly positive integer.If several cores are available, the program can work on several chromosomes in parallel. This decreases the time needed for processing but uses more memory.

**keepTemp** A single logical. If TRUE, after the merge, the intermediary files will not be erased. It concerns the wig files of subgroups such as 'orphans', 'orphansFromArtefacts', 'multireads'. If FALSE only the merged result will be kept.

**logTofile** A single string, or NULL. If the string defines a valid filename, the log messages for all experiments will be written in it. Note that there is a local copy of the log for each experiment in the result folder.

**eraseLog** A single logical. If FALSE, the pipeline stops if a log file with the same name already exists. This is a security to avoid deleting previously computed results. One can disable this security by putting this parameter as TRUE.

## 4.6 'list parameters'

### 4.6.1 Type

These arguments define (or not) a value specific for an exepriment (a dataset). They are named lists of elements, each element name corresponding to an element name in 'inputFileList'. If one element name is not found in 'inputFileList', an error is raised. If one element of 'inputFileList' has no corresponding element in one of these parameters, the module that handle this parameter is ignored.

### 4.6.2 Description (as in'?processPipeline')

**multiLocFilesList** A named (or empty) list. Names of elements must match the experiments names defined in 'INPUTFilesList'. Each element must be the full path to a tex file resulting from multiread processing (see multiread specific commands in the package).

## 4.7 Example

As a summary the following code will start the pipeline using as input the test file bundled in the package. These common parameters (most of them are default values) are well suited for most chIP-seq and Mnase-seq experiments (midpoint or not).

---

```
library(Pasha);

# Get the location of the test file bundled in the package
testFile <- system.file("extdata", "embedDataTest.bam",package="Pasha")

# Create a temporary directory where the original file will be copied and the results generated
temporaryDirectory=tempdir();
file.copy(testFile, temporaryDirectory);

# Define the input file structure and fill it
myExperiments <- list();

myExperiments[["testFile_example"]] <- list(
    'folderName'=temporaryDirectory,
    'fileName'=basename(testFile),
    'fileType'="BAM",
    'chrPrefix'="chr",
    'chrSuffix'="",
    'pairedEnds'=FALSE,
    'midPoint'=FALSE);

# Start the pipeline with basic parameters
processPipeline(
       # I/O GENERAL PARAMETERS
       INPUTFilesList,
       resultSubFolder           = "Results_Pasha",
       reportFilesSubFolder      = "ReportFiles",
       WIGfs                     = TRUE,
       WIGvs                     = FALSE,
       GFF                       = FALSE,
       BED                       = FALSE,
       BIGWIG                    = FALSE,
       compatibilityOutputWIG    = FALSE,
       # COMPLEX PARAMETERS (SINGLE OR VECTORS OR LIST OF IT)
       incrArtefactThrEvery      = 7000000,
       binSize                   = 50,
       elongationSize            = NA,
       rangeSelection            = IRanges(0,-1),
       annotationFilesGFF        = NA, # GFF files
       annotationGenomeFiles     = NA, # path to file or "mm9", "hg19"...
       # SINGLE PARAMETERS
```

```
        elongationEstimationRange = c(mini=150, maxi=400, by=10),
        rehabilitationStep        = c("orphans","orphansFromArtefacts"),
        removeChrNamesContaining  = "random|hap",
        ignoreInsertsOver         = 500,
        nbCPUs                    = 1,
        keepTemp                  = TRUE,
        logTofile                 = "./log.txt",
        eraseLog                  = FALSE,
        # LIST PARAMETERS (one element per expName)
        multiLocFilesList         = list());
```

This code should generate a result folder containing WIG files and reports are stored in a subfolder. The latter include graphics generated by different modules (elongation estimation, artefacts removing, fragment size grouping...), and a log file giving all details on running process. If the user specify several different values for 'binsize', 'incrArtefactThrEvery' and 'elongationSize', several subfolder and other files will be generated.

Some more complex examples are shown in '?processPipeline' documentation or in the help of individual modules functions.

## 5 Using basic functions outside of the pipeline

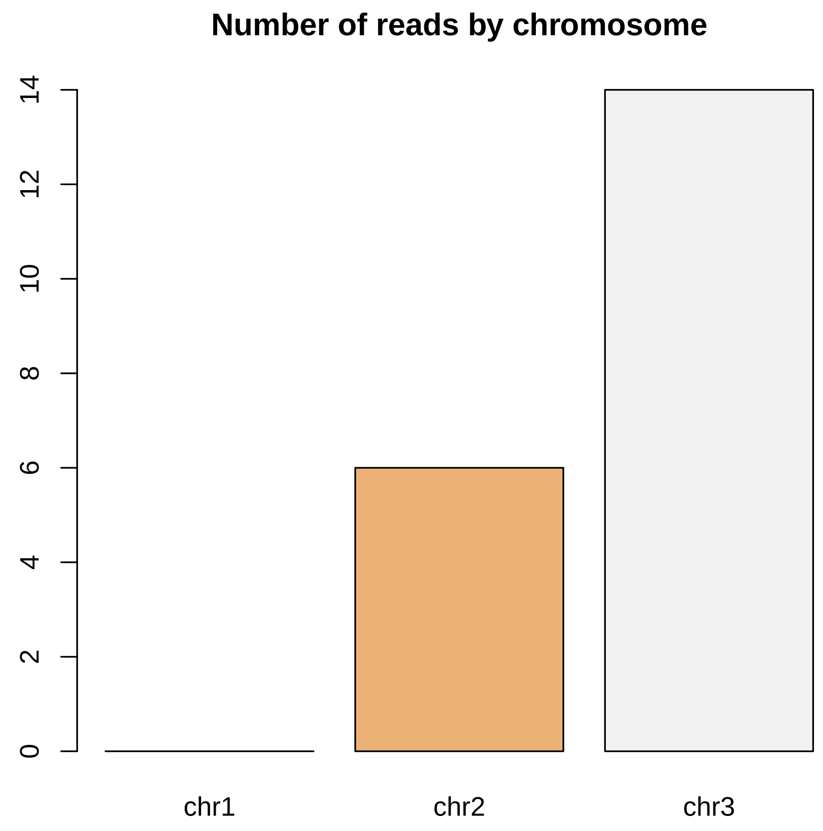
### 5.1 AlignedData-class

The package is built around a class that defines the memory structure of the aligned reads. This class is designed to store information read from BAM or other aligned file and to give methods in order to use this information. The help page '?AlignedData' and '?readAlignedData' give an extensive description of this structure and its main constructor function.

The following code could be used to read data from a BAM file and to extract some useful information for further processings.

```
> library(Pasha);
> # Get the location of the test file bundled in the package
> testFile <- system.file("extdata", "embedDataTest.bam",package="Pasha");
> # Load the file in an object
> myData <- readAlignedData(
+ folderName <- dirname(testFile),
+ fileName <- basename(testFile),
+ fileType <- "BAM",
+ pairedEnds <- TRUE);
> cat("\nNumber of reads in the loaded file :", length(myData), "\n");

Number of reads in the loaded file : 22

> # Remove reads that don't have a mate (orphans)
> myData <- myData[!getOrphansIndexes(myData, quiet=FALSE)];

 Searching for orphans reads, (incomplete pairs)

> # Checking that paired-end representation is compatible with internal
> # class representation
> if(!checkPairsOK(myData))
+ {
+     myData <- sortByPairs(myData, quiet=FALSE);
+ }

 Checking pairs consistency
     OK
 Check that sorting seems ok...
     OK
```

```
> # Split the dataset in a list by chromosomes
> myData_ChrList <- split(myData, seqnames(myData));
> # Plot the number of reads per chromosome in the dataset
> barplot(sapply(myData_ChrList, length),
+         main="Number of reads by chromosome",
+         col=terrain.colors(length(myData_ChrList)));
```



## 5.2 Estimate fragment size, Artefacts piles and Piling scores

These modules which are key components of the pipeline are also available as separate functions. See help pages for 'estimateElongationSize', 'generatePiled' and 'getArtefactsIndexes' for more details.

As 'pipeline module' functions, they expect as main argument an object of the class 'AlignedData' and few other parameters. If they are originately designed to work on a signle chromosome (the user must sort the reads prior to function call), 'estimateElongationSize' and 'generatePiled' are able to detect objects containing multiple chromosomes, and in this case they process sequentially all chromosomes and return a list of results, each element corresponding to a chromosome.

# 6 Using multireads option

The functions dedicated to multiread produce output that can then be injected in the main pipeline. This information will be integrated in the signal to produce the final results (merged WIG/GFF files) taking in account the multireads.

In order to get the file containing the multiread information, several steps must be done:

1. Align the reads with Bowtie with the right parameters to get the multiread you desire
2. (Optional) Prepare a reference file containing the information on the used genome

3. Remove the artifact (i.e. the reads that abnormaly accumulate at the exactly same genomic position)
4. Dispatch the signal along the various multiread position. Two methods are proposed: uniform or according to the CSEM algorithm (see Chung et al. ”Discovering Transcription Factor Binding Sites of Genomes with Multi-Read Analysis of ChIP-seq Data” (2011) PLoS Computational Biology)
5. Launch the Main Pipeline injecting the multiread signal
6. (Optional) Analyze the distribution of the multiread information along the repeated regions class and family.

## 6.1 Align the reads

To align the reads, you will run Bowtie command with the options requested to get read aligned in several location reported. In order to get an output file not too big (since multiread alignement can bring very large files), the ”–concise” option of Bowtie must be used. Other options are standard.

For instance, if you want to keep the read aligning on the mm9 genomes 100 times or less, authorizing 2 mismatches and run on 8 processors

```
    bowtie -q -v 2 -a -m 100 -p 8 --concise mm9 input_file.fastq > output_file.bow
```

## 6.2 Prepare the genome reference file

The scripts used in the next steps requires a reference file containing information on the chosen genome. This reference file is a simple text file with ”.ref” extension that must contain:

- At the first line, the number of chromosoms in the considered genome
- At the second line, the list of chromosoms names (separated by spaces)
- At the third line, the list of chormosoms sizes (separated by spaces)

You can use the ’bowtie-inspect’ command on the reference genome index files to get the information requested to build this reference file.

## 6.3 Remove the artifact piles

Accumulation of reads at the exact same position on the genome may be considered as consequences of artifacts encoutered during the experiment and may be removed. Apply the following R command to remove artifact from alignment.

```
multiread_RemoveArtifact(alignedFile, referenceFile, incrArtefactThrEvery, verbosity)
```

where:

- **alignedFile** is the .bow file outputed by bowtie in step 1
- **referenceFile** is the .ref file created in step 2
- **incrArtefactThrEvery** is the ratio used to detect artifact (see details above)
- **verbosity** is the verbosity level (0 = no message, 1 = trace level)

This command will output a file named with the alignedFile name suffixed by ”_NoA.bow”

## 6.4 Dispatch signal along multiread positions

You have now to decide how each multiread read score will be dispatched along its mutiple positions. Two options are offered here:

**Uniform method** With this method, the signal is equally dispatched along the read position i.e. for instance, if a read has 10 aligned positions, each position will receive a score of 1/10. To apply this method, execute the R command:

```
multiread_UniformDispatch(alignedFile, outputFolder, referenceFile)
```

where :

- **alignedFile** is the .bow file outputed by bowtie in step 3 (or in step 1 if you decided not to remove artifacts)
- **outputFolder** is the folder where you want to see the output file fall in.
- **referenceFile** is the .ref file created in step 2

This command will output a file named with the alignedFile name suffixed by "_uniformDispatch.txt".

**CSEM method** With this method, the score on read position is allocated according to an algorithm developped by Chung et al. (see "Discovering Transcription Factor Binding Sites of Genomes with Multi-Read Analysis of ChIP-seq Data" (2011) PLoS Computational Biology). To apply this method, execute the R command:

```
multiread_CSEMDispatch( alignedFile, outputFolder, referenceFile, windowSize, iterationNumber)
```

where :

- **alignedFile** is the file outputed by bowtie in step 3 (or in step 1 if you decided not to remove artifacts) outputFolder : is the folder where you want to see the output file fall in.
- **referenceFile** is the .ref file created in step 2
- **windowSize** is the size of the window used by the algorithm (see algorithm details). Default value is 101.
- **iterationNumber** is the number of iteration executed by the algorithm. Default value is 200.

This command will output a file named with the alignedFile name suffixed by "_csemDispatch.txt".

## 6.5 Launch Pasha for multiread

One you have obtained a file containing the information of the dispatched scores over the read positions, you can use it in Pasha. See details above on 'multiLocFilesList' parameter.

## 6.6 Analyze repeat distribution

Once Pasha has been launched, you have obtained WIG files. Running the following command will permit you to obtain information and statistics about the dispertion of your signal on annotations identified as repeated regions in the genome.

Use the following command to launch the script:

```
WigRepeatAnalyzer(filename, inputFolder, outputFolder, repeatMaskerFilePath, isRegex)
```

where:

- **filename** is the file name of the wig file (fixed step WIG).
- **inputFolder** is the path to the wig file.
- **outputFolder** is the path to the folder where analysis results must be stored.
- **repeatMaskerFilePath** is the path to the file containing the repeat annotations (Repeat Masker file).

- **isRegex** If TRUE, the filename parameter is interpreted as a regular expression (LC_SYNTAX) and the script will search for a unique file corresponding to the provided regular expression. If no or several file are found, the scripts ends with error.

The script compute the coverage of each repeat class and family (i.e. the percentage of positions falling into each annotations) and the weight of each class and family (i.e. the percentage of score falling into each annotations). All results are provided as barplots figures and text files.

## 7 Additional functions

The package also provide some functions that can be helpful for dealing with WIG fixed step format (see specific help page of each function for more details).

'ReadWIG' and 'WriteWIG' are two functions that help to read from and write to files values in 'WIG fixed steps' format. They assume a representation as a list of numeric vectors, each element name being a chromosome name and each numeric value corresponding to a 'step' (or genomic coordinate).

NormAndSubtract helps to normalize the average level of enrichment in your wig files and allow to subtract scores of wig files (input for instance).

MergeWigs computes the average score of several wig files and generates a merged result file containing these merged scores.

## 8 Package self-test

The Pasha package comes with 'functionnal test' functions ('Pasha:::.testFunctional()' and 'Pasha:::.testFunctionalMultirea
The goal of these functions is to detect some behaviourial discrepancies from the package if running on different platforms/configurations. The package includes an archive containing results expected from the pipeline when started with several different parameters. The functionnal test functions are in charge of extracting these 'expected results' in a temporary folder, start the pipeline and features with the same parameters on the host and compare all resutling files with the reference.

If these tests can't guarantee that algorithms are perfect, they can however guarantee that the user obtains the same results as the developper for a set of extended parameters. This aspect can help for further development/debugging of the package if needed.

### 8.1 Main pipeline functionnal test

This test is systematically done on package checking but can be started manually at any time by calling the function '.testFunctional'. The arguments of this function allow to ask for a 'regular' check or for a 'complete' check. This latter will generate much more results and compare them to reference, trying to maximize the number of tested modules.

Most of these tests files were compared and validated with the pdf attached to the package ("Package-Installation-Folder/extData/resultTests.pdf").

### 8.2 Multiread workflow functionnal test

This test is also systematically done on package checking and can be started manually at any time by calling the function 'Pasha_testMultireadFunctional'.

This test compare the result of multiread read analysis that must be execute when users wants Pasha to take into account the multiread part of an alignment (see complete pipeline description for details).

see '?Pasha:::.testFunctional' for more detailed informations.