

# Package ‘metagene2’

March 20, 2025

**Version** 1.22.0

**Date** 2022-03-03

**Title** A package to produce metagene plots

**Description** This package produces metagene plots to compare coverages of sequencing experiments at selected groups of genomic regions. It can be used for such analyses as assessing the binding of DNA-interacting proteins at promoter regions or surveying antisense transcription over the length of a gene. The metagene2 package can manage all aspects of the analysis, from normalization of coverages to plot faceting according to experimental metadata. Bootstrapping analysis is used to provide confidence intervals of per-sample mean coverages.

**biocViews** ChIPSeq, Genetics, MultipleComparison, Coverage, Alignment, Sequencing

**License** Artistic-2.0

**LazyData** true

**BugReports** <https://github.com/ArnaudDroitLab/metagene2/issues>

**URL** <https://github.com/ArnaudDroitLab/metagene2>

**VignetteBuilder** knitr

**Depends** R (>= 4.0), R6 (>= 2.0), GenomicRanges, BiocParallel

**Imports** rtracklayer, tools, GenomicAlignments, GenomeInfoDb, IRanges, ggplot2, Rsamtools, purrr, data.table, methods, dplyr, magrittr, reshape2

**Suggests** BiocGenerics, RUnit, knitr, BiocStyle, rmarkdown

**RoxygenNote** 7.1.1

**git\_url** <https://git.bioconductor.org/packages/metagene2>

**git\_branch** RELEASE\_3\_20

**git\_last\_commit** c4df766

**git\_last\_commit\_date** 2024-10-29

**Repository** Bioconductor 3.20

**Date/Publication** 2025-03-20

**Author** Eric Fournier [cre, aut],  
Charles Joly Beuparlant [aut],  
Cedric Lippens [aut],  
Arnaud Droit [aut]

**Maintainer** Eric Fournier <ericfournier2@yahoo.ca>

## Contents

as_is_region_order . . . . .	2
Bam_Handler . . . . .	3
coverage_order . . . . .	6
get_demo_bam_files . . . . .	6
get_demo_design . . . . .	7
get_demo_metagene . . . . .	7
get_demo_regions . . . . .	8
get_demo_region_filenames . . . . .	8
get_demo_rna_bam_files . . . . .	9
get_demo_rna_regions . . . . .	9
metagene2 . . . . .	10
metagene2_heatmap . . . . .	19
plot_metagene . . . . .	20
<b>Index</b>	<b>22</b>

---

as_is_region_order	<i>Returns an "as-is" ordering of regions.</i>
--------------------	--

---

### Description

This function creates an ordering of regions to be used with the [metagene2\\_heatmap](#) function. The regions are not actually reordered, but returned as-is.

### Usage

```
as_is_region_order(metagene)
```

### Arguments

metagene      The metagene object whose grouped regions should be ordered.

### Value

A list, with as many elements as there are region groups in the metagene object. Each element of that list is an ordering of the regions of that group based on their original ordering in the metagene2 object.

### Examples

```
demo_metagene = get_demo_metagene()
as_is_region_order(demo_metagene)
```

Bam\_Handler

*A class to manage BAM files.***Description**

This class will allow to load, convert and normalize alignments and regions files/data.

**Format**

A BAM manager

**Value**

Bam\_Handler\$new returns a Bam\_Handler object which contains coverage related information for every BAM files.

**Constructor**

```
bh <- Bam_Handler$new(bam_files, cores = SerialParam())
```

**bam\_files** A vector of BAM filenames. The BAM files must be indexed. i.e.: if a file is named file.bam, there must be a file named file.bam.bai or file.bai in the same directory.

**cores** The number of cores available to parallelize the analysis. Either a positive integer or a BiocParallelParam. Default: SerialParam().

**paired\_end** If TRUE, metagene will deal with paired-end data. If FALSE, single-end data are expected

Bam\_Handler\$new returns a Bam\_Handler object that contains and manages BAM files. Coverage related information as alignment count can be obtain by using this object.

**Methods**

```
bh$get_aligned_count(bam_file)
```

**bam\_file** The name of the BAM file.

```
bg$get_bam_name(bam_file)
```

**bam\_file** The name of the BAM file.

```
bh$get_rpm_coefficient(bam_file)
```

**bam\_file** The name of the BAM file.

```
bh$index_bam_files(bam_files)
```

**bam\_files** A vector of BAM filenames.

```
bh$get_bam_files()
```

```
bh$get_coverage(bam_file, regions) force_seqlevels = FALSE)
```

**bam\_file** The name of the BAM file.

**regions** A not empty GRanges object.

**force\_seqlevels** If TRUE, Remove regions that are not found in bam file header. Default: FALSE. TRUE and FALSE respectively correspond to pruning.mode = "coarse" and "error" in ?seqinfo.

```
bh$get_normalized_coverage(bam_file, regions) force_seqlevels = FALSE)
```

**bam\_file** The name of the BAM file.

**regions** A not empty GRanges object.

**force\_seqlevels** If TRUE, Remove regions that are not found in bam file header. Default: FALSE. TRUE and FALSE respectively correspond to pruning.mode = "coarse" and "error" in ?seqinfo.

**chip\_bam\_file** The path to the chip bam file.

**input\_bam\_file** The path to the input (control) bam file.

## Methods

### Public methods:

- [Bam\\_Handler\\$new\(\)](#)
- [Bam\\_Handler\\$get\\_bam\\_name\(\)](#)
- [Bam\\_Handler\\$get\\_aligned\\_count\(\)](#)
- [Bam\\_Handler\\$get\\_rpm\\_coefficient\(\)](#)
- [Bam\\_Handler\\$index\\_bam\\_files\(\)](#)
- [Bam\\_Handler\\$get\\_bam\\_files\(\)](#)
- [Bam\\_Handler\\$get\\_coverage\(\)](#)
- [Bam\\_Handler\\$get\\_normalized\\_coverage\(\)](#)
- [Bam\\_Handler\\$clone\(\)](#)

### Method new():

*Usage:*

```
Bam_Handler$new(
  bam_files,
  cores = SerialParam(),
  paired_end = FALSE,
  strand_specific = FALSE,
  paired_end_strand_mode = 2,
  extend_reads = 0,
  invert_strand = FALSE
)
```

### Method get\_bam\_name():

*Usage:*

```
Bam_Handler$get_bam_name(bam_file)
```

### Method get\_aligned\_count():

*Usage:*

```
Bam_Handler$get_aligned_count(bam_file)
```

### Method get\_rpm\_coefficient():

*Usage:*

```
Bam_Handler$get_rpm_coefficient(bam_file)
```

**Method** `index_bam_files()`:

*Usage:*

```
Bam_Handler$index_bam_files(bam_files)
```

**Method** `get_bam_files()`:

*Usage:*

```
Bam_Handler$get_bam_files()
```

**Method** `get_coverage()`:

*Usage:*

```
Bam_Handler$get_coverage(  
  bam_file,  
  regions,  
  force_seqlevels = FALSE,  
  simplify = TRUE  
)
```

**Method** `get_normalized_coverage()`:

*Usage:*

```
Bam_Handler$get_normalized_coverage(  
  bam_file,  
  regions,  
  force_seqlevels = FALSE,  
  simplify = TRUE  
)
```

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
Bam_Handler$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
bam_file <- get_demo_bam_files()[1]  
bh <- metagene2::Bam_Handler$new(bam_files = bam_file)  
bh$get_aligned_count(bam_file)
```

---

coverage_order	<i>Determines ordering of regions as a function of coverage.</i>
----------------	--

---

### Description

This function creates an ordering of regions within region groups based on ascending or descending mean coverage. This is used with the `metagene2_heatmap` function.

### Usage

```
coverage_order(metagene, design_groups = NULL, decreasing = TRUE)
```

### Arguments

metagene	The metagene object whose grouped regions should be ordered.
design_groups	A vector of design groups to be used for determining the ordering. If NULL, all design groups are used.
decreasing	If TRUE, regions are ordered from the highest mean coverage to the lowest mean coverage, and vice versa.

### Value

A list, with as many elements as there are region groups in the metagene object. Each element of that list is an ordering of the regions of that group based on their mean coverage.

### Examples

```
demo_metagene = get_demo_metagene()
coverage_order(demo_metagene)
```

---

get_demo_bam_files	<i>Get BAM filenames for demo</i>
--------------------	-----------------------------------

---

### Description

Get BAM filenames for demo

### Usage

```
get_demo_bam_files()
```

### Value

A vector of BAM filenames

### Examples

```
bam_files <- get_demo_bam_files()
```

---

`get_demo_design`      *Get a demo design object*

---

**Description**

Get a demo design object

**Usage**

```
get_demo_design()
```

**Value**

A `data.frame` corresponding to a valid design.

**Examples**

```
mg <- get_demo_design()
```

---

`get_demo_metagene`      *Get a demo metagene object*

---

**Description**

Get a demo metagene object

**Usage**

```
get_demo_metagene()
```

**Value**

A metagene object

**Examples**

```
mg <- get_demo_metagene()
```

---

<code>get_demo_regions</code>	<i>Get demo regions</i>
-------------------------------	-------------------------

---

**Description**

Get demo regions

**Usage**

```
get_demo_regions()
```

**Value**

A vector of regions filenames

**Examples**

```
regions <- get_demo_regions()
```

---

<code>get_demo_region_filenames</code>	<i>Get regions filenames for demo</i>
--	---------------------------------------

---

**Description**

Get regions filenames for demo

**Usage**

```
get_demo_region_filenames()
```

**Value**

A vector of regions filenames

**Examples**

```
regions <- get_demo_regions()
```



---

`get_demo_rna_bam_files`

*Get BAM filenames for demo*

---

**Description**

Get BAM filenames for demo

**Usage**

```
get_demo_rna_bam_files()
```

**Value**

A vector of BAM filenames

**Examples**

```
bam_files <- get_demo_rna_bam_files()
```

---

`get_demo_rna_regions` *Get demo regions*

---

**Description**

Get demo regions

**Usage**

```
get_demo_rna_regions()
```

**Value**

A GRangesList with two genes

**Examples**

```
regions <- get_demo_rna_regions()
```

---

metagene2

*A class to manage metagene analysis.*

---

## Description

This metagene2 class encapsulates all of the steps necessary to perform metagene analyses, which are aggregations of coverages over multiple regions (genes) to reveal patterns that might not be apparent from looking at individual regions. It will allow to load, convert and normalize bam alignments and regions files/data. Once the data is ready, the user can then choose to produce metagene plots on the data or some subset of it.

## Format

A metagene experiment manager

## Details

Most metagene analyses are a two-step affair:

1. Initialize the object using `mg = metagene2$new()`, specifying which regions and bam files should be used for the analysis.
2. Generate a metagene plot using `mg$produce_metagene`, specifying any additional parameter (Number of bins, faceting variables, etc).

The metagene2 object will then internally chain all 6 required processing steps, updating its internal caches along the way:

1. Coverages are inferred from bam files (`metagene2$new`).
2. Coverages from multiple bam files are grouped and normalized (`mg$group_coverages`).
3. Coverages are binned together (`mg$bin_coverages`).
4. Binned coverages are split according to the type of region they belong to (`mg$split_coverages_by_regions`).
5. Coverage means and confidence intervals are calculated for each region \* group combination (`mg$calculate_ci`).
6. Metadata is added to the calculated coverages (`mg$add_metadata`).
7. The metagene is plotted (`mg$plot`).

Each of these steps has an associated function, which takes as input certain parameters of the metagene analysis and returns an intermediary structure of interest (coverages, binned coverages, long-form data frame of confidence intervals, etc). Those are described below, in the "Processing methods" section.

All processing methods automatically call previous processing steps if those have not already been run. For example, there is no need to call `mg$group_coverages()` before calling `mg$bin_coverages()`: the metagene2 object will automatically detect that certain prerequisite steps have not yet been performed, and run them.

Additionally, when calling `produce_metagene` a second time to change certain analysis parameters after generating an initial metagene plot, only the required caches are reset: all non-impacted aspects of the analysis are left untouched, decreasing processing time.

For further examples, see the metagene2 vignette.

**Value**

metagene2\$new returns a metagene2 object which contains the normalized coverage values for every regions in all specified BAM files.

**Constructor****Usage:**

```
mg <- metagene2$new(regions, bam_files, padding_size = 0, cores = SerialParam(), verbose = FALSE, force_seqlevels = FALSE, paired_end = FALSE, assay = 'chipseq', strand_specific=FALSE, paired_end_strand_mode=2, region_mode="auto", region_metadata=NULL, extend_reads=0, invert_strand=FALSE, ...)
```

**Description:**

This method returns a new metagene2 object. Upon initialization, a metagene2 object calculates coverages over all given regions in the provided bam files. Any and all parameter associated with any of the processing steps can be initialized upon object construction. All analysis parameters that are not explicitly specified in the constructor call are initialized to sensible defaults.

**Parameters:**

**regions** A description of all regions over which metagenes will be calculated.

When region\_mode is "separate", those should be provided using a GRanges object representing all individual, contiguous regions to be examined.

When region\_mode is "stitch", those should be provided using a GRangesList object where each individual GRanges element represents a set of regions to be stitched together.

As a convenience, in "separate" mode, metagene2 will convert any passed in GRangesList into an unlisted GRanges with an additional region\_name metadata column containing the name of the GRangesList element it was extracted from.

Also as a convenience, regions can also be a character vector of filenames, which are then imported into a GRangesList. Supported file formats are BED, narrowPeak, broadPeak, gff and gtf.

**bam\_files** A vector of BAM filenames. The BAM files must be indexed. i.e.: if a file is named file.bam, there must be a file named file.bam.bai or file.bai in the same directory. If bam\_files is a named vector, then the provided names can be used downstream to refer to those bam files. If no names are provided, metagene2 will try to infer appropriate ones.

**assay** 'chipseq', 'rnaseq' or NULL. If non-NULL, metagene will set other parameters, such as region\_mode and strand\_specific, to logical values for the given assay. Default: 'chipseq'

**region\_mode** Set the way the regions parameter is interpreted. Can be 'separate', 'stitch' or 'auto'. In separate mode, regions is expected to be a GRanges defining individual, contiguous regions. In 'stitch' mode, regions is expected to be a GRangesList where each GRanges element represents a set of regions to be stitched together and treated as a single logical region. If 'auto' then a logical value is inferred from the assay parameter. Default: 'auto'

**region\_metadata** A data-frame of metadata to be associated with the elements of regions. It must contain has many rows as there are elements in regions. If region\_metadata is NULL but regions has an mcols element, then it is used.

**padding\_size** The provided regions will be extended on each side by the value of this parameter. The padding\_size must be a non-negative integer. Default = 0.

**cores** The number of cores available to parallelize the analysis. Either a positive integer or a BiocParallelParam. Default: SerialParam().

**verbose** Print progression of the analysis. A logical constant. Default: FALSE.

**force\_seqlevels** If TRUE, remove regions that are not found in bam file header. Default: FALSE. TRUE and FALSE respectively correspond to `pruning.mode = "coarse"` and `"error"` in `?seqinfo`.

**paired\_end** Set this to TRUE if the provided bam files describe paired-end reads. If FALSE, single-ended data are expected. Default: FALSE

**strand\_specific** If TRUE, only reads which align to the same strand as those specified in regions will count toward coverage for that region. Useful for RNA-seq profiles generated from strand-specific libraries, such as Illumina TruSeq. Default: 'FALSE'

**paired\_end\_strand\_mode** '1' or '2'. In paired-end mode, indicates which read in a pair sets the pair's strand. If 1, this is the first read (This should be used with directional protocols such as Directional Illumina (Ligation) or Standard SOLiD). If 2, this is the second read (This should be used with directional protocols such as dUTP, NSR, NNSR, or Illumina stranded TruSeq PE). Ignored if either `paired_end` or `strand_specific` is FALSE. Default: '2'

**extend\_reads** Extend individual reads to have a minimum length equal to this parameter. When set to 0, no read extension occurs. This is useful for single-end chip-seq experiments, where the length of the captured fragment is usually longer than the sequenced read.

**invert\_strand** If TRUE, coverages for the given regions will be inferred from the coverage on the strand opposite theirs. Useful for single-end stranded experiments which use cDNA. This parameter is ignored if `strand-specific` is FALSE.

... Additional parameters for the metagene analysis. See `produce_metagene` for a list of possible parameters.

`metagene2$new` returns a `metagene2` object that contains the coverages for every BAM files in the `regions` from the `regions` parameter.

## **produce\_metagene()**

### **Usage:**

```
mg$produce_metagene(...)
```

### **Description:**

`produce_metagene` is the workhorse method of the `metagene2` object. This method performs all of the necessary analysis steps for the production of the metagene plot, and returns that plot. Any and all parameters of the metagene analysis, as documented in the individual processing steps, can be passed to `produce_metagene`. The `metagene2` object will then determines which intermediate caches would be affected by changes to those parameters, invalidate them, and rerun all steps up to the plotting. This makes `produce_metagene` ideal for fast, iterative takes on the data.

Below we present those parameters and a brief description of their usage. Please refer to the affected processing step for a more in-depth explanation of each parameter.

### **Parameters:**

**design** A `data.frame` that describes the grouping of the bam files into design groups. By default, each bam file is its own design group. See `group_coverages`.

**normalization** The algorithm to use to normalize coverages, NULL (no normalization) or "RPM". By default, no normalization occurs. See `group_coverages`.

**design\_filter** Indices indicating which subset of design groups should be included in the analysis. By default, all design groups/bam files are included. See `group_coverages`.

**bin\_count** The number of bins regions should be split into. Defaults to 100. See `bin_coverages`.

**region\_filter** The subset of regions to be kept for the analysis. By default, all regions are kept. See `bin_coverages`

- split\_by** Which metadata columns should we use to split the set of regions into subset of interests? Defaults to "region\_name", an automatically added column. See `split_coverages_by_regions`.
- alpha** The alpha level of the confidence interval estimates. Defaults to 0.05. See `calculate_ci`.
- sample\_count** The number of draws to perform in the bootstrap calculations used to calculate the confidence interval. Defaults to 1000. See `calculate_ci`.
- resampling\_strategy** The resampling strategy to be used when performing the bootstrap analysis, which can be either 'profile' or 'bin'. Defaults to 'bin'. See `calculate_ci`.
- design\_metadata** A data-frame containing metadata for the design groups. By default, no metadata is associated. See `add_metadata`.
- title** A title to add to the graph. See `plot`.
- x\_label** X-axis label for the metagene plot. See `plot`.
- facet\_by** A formula to be used for faceting the metagene plot. By default, no faceting occurs. See `plot`.
- group\_by** The metadata column used to build the color scale. By default, the combination of design and region name is used. See `plot`.

## Processing methods

Each of the following methods perform one step of metagene processing. Most do not need to be called explicitly. Instead, you can simply call `produce_metagene`. However, you can use them to access intermediary results: grouped coverages, binned coverages, split coverages, and long-form data-frame of coverages with confidence intervals.

### group\_coverages

#### Usage:

```
mg$group_coverages(design=NA, normalization=NA, design_filter=NA, simplify=FALSE)
```

#### Description:

This method normalizes genome-wide coverages, then groups them according to the specified design groups. It returns a list of possible read orientations (+, -, \*), each element of which is either NULL (depending on the value of the `strand_specific` parameter) or a list of possible design groups. In turn, the lists of design groups contain lists of R1e objects representing coverage over a specific chromosome or sequence.

#### Parameters:

- design** A data.frame that describes the grouping of the bam files into design groups. The first column of the design should contain the names of bam\_files passed on initialization. Each subsequent columns represents a design group, that is to say a combination of bam files whose coverages should be grouped together into a logical unit. These columns should contain integer values indicating whether the bam files on that row should be excluded (0), included as an 'input' (1) or included as a "control" (2) within the specified design group. Control samples are used for "log2\_ratio" normalization, but are ignored for no or "RPM" normalization. NA can be used keep previous design value. Default: NA.
- normalization** The algorithm to use to normalize coverages. Possible values are NULL (no normalization), "RPM" and "log2\_ratio". "RPM" transforms raw counts into Reads-Per-Million. "log2\_ratio" uses the formula  $\log_2((\text{input RPM} + 1) / (\text{control RPM} + 1))$  to calculate a log-ratio between input and control. NA can be used keep the previous value. Default: NA
- design\_filter** A logical vector specifying which of the design groups specified within the design parameter should be included in the metagene. Useful for quickly reprocessing a subset of samples. NA can be used keep previous design value. Default: NA

**simplify** In single strand mode, set `simplify` to `TRUE` to return only the '\*' coverage and omit the empty '+' and '-' components. Default: `FALSE`

### **bin\_coverages**

**Usage:**

```
mg$bin_coverages(bin_count=NA, region_filter=NA)
```

**Description:**

This method summarizes the coverage over regions of interests into a specified number of bins. For each design group, it produces a matrix of binned coverages where each row represents a region, and each column represents a bin. Those are returned in a named list where each element contains the resulting matrix for a specific design group.

**Parameters:**

**bin\_count** The number of bins regions should be split into. The specified `bin_count` must always be equal or higher than the minimum size of the specified regions. `NA` can be used to keep the previous value. Default: `NA`.

**region\_filter** This parameter defines the subset of regions within the `regions` parameter passed on initialization on which the metagene should be generated. `region_filter` can be (1) a quosure, to be evaluated in the context of the `region_metadata` data-frame, (2) a character vector containing the names of the regions to be used or (3) a logical or numeric vector to be used for subsetting. `NA` can be used to keep the previous value. Default: `NA`

### **split\_coverages\_by\_regions**

**Usage:**

```
mg$split_coverages_by_regions(split_by=NA)
```

**Description:**

This methods splits the matrices generated by `mg$bin_coverages` into groups of regions where the values of the metadata columns specified by `split_by` are homogeneous. It returns a list where each element represents a design group: each of those element is in turn a list representing groups of regions for which all metadata values specified by "split\_by" are equal. The leaf elements of this list hierarchy are coverage matrices where each row represents a region, and each column represents a bin.

**Parameters:**

**split\_by** A vector of column names from the `region_metadata` parameter, as specified on metagene initialization. The selected columns must allow conversion into a factor. By default, this is set to `region_name`, a metadata column which is automatically generated by metagene. `NA` can be used to keep the previous value. Default: `NA`

### **calculate\_ci**

**Usage:**

```
mg$calculate_ci(alpha = NA, sample_count = NA, resampling_strategy=NA)
```

**Description:**

This method calculates coverage means and confidence intervals for all `design_group * region * bin` combination. These are returned as a long-form data-frame.

**Parameters:**

**alpha** The alpha level of the confidence interval estimate. NA can be used to keep the previous value. Default: NA

**sample\_count** The number of draws to perform in the bootstrap calculations used to calculate the confidence interval. NA can be used to keep the previous value. Default: NA

**resampling\_strategy** The resampling strategy to be used when performing the bootstrap analysis, which can be either 'profile' or 'bin'. In 'profile' mode, whole profiles across all bins are resampled. In 'bin' mode, each bin is resampled individually and independently from all others. NA can be used to keep the previous value. Default: NA

### add\_metadata

#### Usage:

```
mg$add_metadata(design_metadata=NA)
```

#### Description:

This method adds design group and region metadata to the data-frame produced by `mg$calculate_ci` for easier plotting.

#### Parameters:

**design\_metadata** A data-frame containing metadata for the design groups. It must contain as many rows as there are design groups, and must contain at least one column named 'design' which is used to match the rows to design groups.

### plot

#### Usage:

```
mg$plot(region_names = NULL, design_names = NULL, title = NA, x_label = NA, facet_by=NA, group_by=NA)
```

#### Description:

This method produces a ggplot object giving a graphical representation of the metagene analysis.

#### Parameters:

**region\_names** The names of the regions to be plotted. If NULL, all the regions are plotted. Default: NULL.

**design\_names** The names of the design groups to be plotted. If NULL, all the design groups are plotted. Default: NULL.

**title** A title to add to the graph. NA can be used to keep the previous value. Default: NA

**x\_label** X-axis label for the metagene plot. NA can be used to keep the previous value. Default: NA.

**facet\_by** A formula to be used for faceting the metagene plot. This formula can include any design metadata, or region\_metadata NA can be used to keep the previous value. Default: NA.

**group\_by** A string representing a single column from design\_metadata or region\_metadata which will be used to group observations together into lines and which will be used to generate the color scale. NA can be used to keep the previous value. Default: NA.

### Getter methods

The following methods return various informations about the metagene object.

#### **mg\$get\_params()**

Returns a list of all parameters used to perform this metagene analysis.

**mg\$get\_design()**

Returns the design used to perform this metagene analysis.

**mg\$get\_regions()**

Returns the regions used for this metagene analysis.

**mg\$get\_data\_frame(region\_names = NULL, design\_names = NULL)**

Returns full data-frame of results.

**region\_names** The names of the regions to extract. If NULL, all the regions are returned. Default: NULL.

**design\_names** The names of the design groups to extract. If NULL, design groups are returned. Default: NULL.

**mg\$get\_plot()**

Returns the ggplot object generated by the metagene2\$plot function.

**mg\$get\_raw\_coverages()**

Returns raw coverages over the regions specified on initialization.

**mg\$get\_normalized\_coverages()**

Returns normalized coverages over the regions specified on initialization.

**Methods****Public methods:**

- metagene2\$new()
- metagene2\$get\_bam\_count()
- metagene2\$get\_params()
- metagene2\$get\_design()
- metagene2\$get\_design\_group\_names()
- metagene2\$get\_regions()
- metagene2\$get\_regions\_metadata()
- metagene2\$get\_split\_regions()
- metagene2\$get\_data\_frame()
- metagene2\$get\_plot()
- metagene2\$get\_raw\_coverages()
- metagene2\$get\_normalized\_coverages()
- metagene2\$set\_cores()
- metagene2\$group\_coverages()
- metagene2\$bin\_coverages()
- metagene2\$split\_coverages\_by\_regions()
- metagene2\$calculate\_ci()
- metagene2\$add\_metadata()
- metagene2\$plot()
- metagene2\$produce\_metagene()
- metagene2\$plot\_single\_region()



- `metagene2$replace_region_metadata()`
- `metagene2$clone()`

**Method new():**

*Usage:*

```
metagene2$new(  
  regions,  
  bam_files,  
  padding_size = 0,  
  cores = SerialParam(),  
  verbose = FALSE,  
  force_seqlevels = FALSE,  
  paired_end = FALSE,  
  assay = "chipseq",  
  strand_specific = FALSE,  
  paired_end_strand_mode = 2,  
  region_mode = "auto",  
  region_metadata = NULL,  
  extend_reads = 0,  
  invert_strand = FALSE,  
  ...  
)
```

**Method get\_bam\_count():**

*Usage:*

```
metagene2$get_bam_count(filename)
```

**Method get\_params():**

*Usage:*

```
metagene2$get_params()
```

**Method get\_design():**

*Usage:*

```
metagene2$get_design()
```

**Method get\_design\_group\_names():**

*Usage:*

```
metagene2$get_design_group_names()
```

**Method get\_regions():**

*Usage:*

```
metagene2$get_regions()
```

**Method get\_regions\_metadata():**

*Usage:*

```
metagene2$get_regions_metadata()
```

**Method get\_split\_regions():**

*Usage:*

```
metagene2$get_split_regions()
```

**Method** get\_data\_frame():*Usage:*`metagene2$get_data_frame(region_names = NULL, design_names = NULL)`**Method** get\_plot():*Usage:*`metagene2$get_plot()`**Method** get\_raw\_coverages():*Usage:*`metagene2$get_raw_coverages()`**Method** get\_normalized\_coverages():*Usage:*`metagene2$get_normalized_coverages()`**Method** set\_cores():*Usage:*`metagene2$set_cores(cores)`**Method** group\_coverages():*Usage:*

```
metagene2$group_coverages(  
  design = NA,  
  normalization = NA,  
  design_filter = NA,  
  simplify = TRUE  
)
```

**Method** bin\_coverages():*Usage:*`metagene2$bin_coverages(bin_count = NA, region_filter = NA)`**Method** split\_coverages\_by\_regions():*Usage:*`metagene2$split_coverages_by_regions(split_by = NA)`**Method** calculate\_ci():*Usage:*`metagene2$calculate_ci(alpha = NA, sample_count = NA, resampling_strategy = NA)`**Method** add\_metadata():*Usage:*`metagene2$add_metadata(design_metadata = NA)`**Method** plot():*Usage:*

```
metagene2$plot(  
  region_names = NULL,  
  design_names = NULL,  
  title = NA,  
  x_label = NA,  
  facet_by = NA,  
  group_by = NA  
)
```

**Method** produce\_metagene():

*Usage:*

```
metagene2$produce_metagene(...)
```

**Method** plot\_single\_region():

*Usage:*

```
metagene2$plot_single_region(  
  region,  
  facet_by = NA,  
  group_by = NA,  
  no_binning = FALSE  
)
```

**Method** replace\_region\_metadata():

*Usage:*

```
metagene2$replace_region_metadata(region_metadata)
```

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
metagene2$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
mg <- metagene2$new(regions = get_demo_regions(), bam_files = get_demo_bam_files())  
## Not run:  
mg$plot()  
  
## End(Not run)
```

---

metagene2\_heatmap

*Plots a heatmap of coverages from a metagene2 object.*

---

## Description

This function creates an ordering of regions within region groups based on ascending or descending mean coverage. This is used with the [metagene2\\_heatmap](#) function.

**Usage**

```
metagene2_heatmap(
  metagene,
  region_order = as_is_region_order(metagene),
  scale_trans = "identity"
)
```

**Arguments**

metagene	The metagene object to be plotted as a heatmap.
region_order	A named list with as many elements as there are region groups, with each element containing an ordering for the regions within that group. The <a href="#">as_is_region_order</a> and <a href="#">coverage_order</a> functions can be used to generate a valid ordering. By default, <a href="#">as_is_region_order</a> is used.
scale_trans	A character string giving the transformation that should be applied to the coverage values. Common values are "identity" and "log1p". See the <a href="#">ggplot2</a> documentation for <a href="#">scale_continuous</a> for more details.

**Value**

A ggplot object containing a heatmap representation of the metagene2 object.

**Examples**

```
demo_metagene = get_demo_metagene()
metagene2_heatmap(demo_metagene)
```

---

plot_metagene	<i>Produce a metagene plot</i>
---------------	--------------------------------

---

**Description**

Produce a metagene plot

**Usage**

```
plot_metagene(df, facet_by = NULL, group_by = NULL)
```

**Arguments**

df	a <code>data.frame</code> obtained with the <code>get_data_frame</code> function. Must have the following columns: "region", "design", "bin", "value", "qinf" and "qsup".
facet_by	A formula to be used for facetting the metagene plot. This formula can include any design metadata, or <code>region_metadata</code> NA can be used to keep the previous value. Default: NA.
group_by	A string representing a single column from <code>design_metadata</code> or <code>region_metadata</code> which will be used to group observations together into lines and which will be used to generate the color scale. NA can be used to keep the previous value. Default: NA.

**Value**

A 'ggplot' object.

**Examples**

```
mg <- get_demo_metagene()
df <- mg$add_metadata()
p <- metagene2::plot_metagene(df)
```

# Index

`as_is_region_order`, [2](#), [20](#)

`Bam_Handler`, [3](#)

`coverage_order`, [6](#), [20](#)

`get_demo_bam_files`, [6](#)

`get_demo_design`, [7](#)

`get_demo_metagene`, [7](#)

`get_demo_region_filenames`, [8](#)

`get_demo_regions`, [8](#)

`get_demo_rna_bam_files`, [9](#)

`get_demo_rna_regions`, [9](#)

`metagene2`, [10](#)

`metagene2_heatmap`, [2](#), [6](#), [19](#), [19](#)

`plot_metagene`, [20](#)