nucleo_miner Documentation

Release 2.3.1

Florent CHUFFART, Jean-Baptiste VEYRIERAS, Gael YVERT

January 14, 2014

CHAPTER

ONE

README / DOCUMENTATION FOR *NUCLEOMINER2*

NucleoMiner2 offers Python API and R package allowing to perform quantitative analysis of nucleosomal epigenome. It is especially well suited for scripting to extract natural Single-Nucleosome Epi- Polymorphisms (SNEP) from ChIP-Seq data.

1.1 License

1.2 Installation Instructions

1.2.1 Links

NucleoMiner2 home page and documentation: https://forge.cbp.ens-lyon.fr/redmine/projects/nucleominer

Gael Yvert lab page: http://www.ens-lyon.fr/LBMC/gisv/

1.2.2 Installation

- Download archive
- Compile bowtie2
- Compile samtools
- Compile bedtools
- Compile TemplateFilter

Required R pacakes:

- bot - fork
- rjson
- seqinr

```
cd src/r_packages/
      tar xfvz R-latest.tar.gz
      cd R-patched
      ./configure --with-x=no PDFLATEX="ls"
      make
cd ../../..
R_BIN=src/r_packages/R-patched/bin/R
      $R_BIN CMD INSTALL src/r_packages/rjson_0.2.12.tar.gz
      $R_BIN CMD INSTALL src/r_packages/seqinr_3.0-7.tar.gz
      $R_BIN CMD INSTALL src/r_packages/plotrix_3.4-5.tar.gz
      $R_BIN CMD INSTALL src/r_packages/nm_2.0.tar.gz
      $R_BIN CMD INSTALL src/r_packages/fork_1.2.4.tar.gz
      $R_BIN CMD INSTALL src/r_packages/bot_0.9.tar.gz
      $R_BIN CMD INSTALL src/r_packages/DESeq_1.14.0.tar.gz
```

...

1.3 usage

See html documentation for *NucleoMiner2*: http://www.ens-lyon.fr/LBMC/gisv/

CHAPTER

TWO

TUTORIAL

This tutorial describes steps allowing to perform quantitave analysis of nucleosomal epigenome. We assume that files are organised around a given hierarchie and that all command lines are launched from project's root.

This tutorial is divided into t=wo main parts. First one consists in the python script *wf.py* that aligns and convert Illumina reads. Second one is the R script *main.r* that extracts information (nucleosome position and indicators) from the dataset.

2.1 Dataset and Configuration File

We want to compare nucleosomes of 3 yeast strains:

- BY
- RM
- YJM

For each strain we perform Mnase-Seq and ChIP-Seq using the 5 following markers:

- H3K4me1
- H3K4me3
- H3K9ac
- H3K14ac
- H4K12ac

In order to simplify the design of exeriment, we considere Mnase as a marker. For each couple *(strain, marker)* we perform 3 replicates. So, theoritically we should have *3 * (1 + 5) * 3 = 54* samples. In practice we only obtain 2 replicates for *(YJM, H3K4me1)*. Each one of the 53 samples is indentify by a uniq identifier. The file *CSV_SAMPLE_FILE* sums up this information.

`configurator.`**`CSV_SAMPLE_FILE`** **= None**
Path to cvs file that contains sample information.

We use a convention to link sample and Illumina fastq outputs. Illumina output files of the sample *ID* will be stored in the directory *ILLUMINA_OUTPUTFILE_PREFIX + ID*. For example, sample 41 outputs will be stored in the directory *data/2012-09-05/FASTQ/Sample_Yvert_Bq41/*.

`configurator.`**`ILLUMINA_OUTPUTFILE_PREFIX`** **= None**
Prefix for Illumina fastq output files.

For BY (resp. RM and YJM) we use following reference genome *saccharomyces_cerevisiae_BY_S288c_chromosomes.fasta* (resp. *saccharomyces_cerevisiae_rm11-1a_1_supercontigs.fasta*

and *saccharomyces_cerevisiae_YJM_789_screencontig.fasta*). The index *FASTA_REFERENCE_GENOME_FILES* stores this information.

`configurator.`**`FASTA_REFERENCE_GENOME_FILES`** **= None**
Dictionary where each fasta reference genomes is indexed by reference strain that it corresponds.

Each chromosome/contig is identify in the fasta file by an obscure identifier. For example, BY chromosome I is identify by *gi|144228165|ref|NC_001133.7|* when TemplateFilter is waiting for an integer. So, we translate it. The index *FASTA_INDEXES* stores this translation.

`configurator.`**`FASTA_INDEXES`** **= None**
Dictionary of strain that indexes dictionaries where keys are chromosome reference from Fastq file and value are its correspondance for Templatefilter.

From a pragamatical point of view we discard some part of the genome (repeated sequence etc...). The list of the black listed area is explicitely detailled in *AREA_BLACK_LIST*.

`configurator.`**`AREA_BLACK_LIST`** **= None**
Dictionary where keys are strain and values are black listed of geneome region.

For BY-RM (resp. BY-YJM and RM-YJM) genome sequence alignment we use previously compute .c2c file *data/2012-03_primarydata/BY_RM_gxcomp.c2c* (resp. *BY_YJM_GComp_All.c2c* and *RM_YJM_gxcomp.c2c*). For more information about .c2c files, please read section 5 of the manual of *NucleoMiner*, the old version of *NucleoMiner2* (http://www.ens-lyon.fr/LBMC/gisv/NucleoMiner_Manual/manual.pdf).

`configurator.`**`C2C_FILES`** **= None**
Dictionary where each strain combination indexes genome aligment.

nucleominer uses specific directory to work in, these are described in *INDEX_DIR*, *ALIGN_DIR* and *LOG_DIR*.

Finally, *nucleominer* use external ressources, the path to these resspources are describe in *BOWTIE_BUILD_BIN*, *BOWTIE2_BIN*, *SAMTOOLS_BIN*, *BEDTOOLS_BIN* and *TF_BIN* and *TF_TEMPLATES_FILE*.

All paths, prefixes and indexes could be change in the *src/nucleo_miner/nucleo_miner_config.json* file.

`wf.`**`json_conf_file`** **= 'src/nucleo_miner/nucleo_miner_config.json'**
Path to the json configuration file.

2.2 Preprocessing Illumina Fastq Reads for Each Sample

This preprocessing step consists in the 4 main steps embed in the *wf.py* and described bellow. As a preamble, this script computes *samples samples_mnase* and *strains* that will be used along the 4 steps.

`wf.`**`samples`** **= []**
List of samples where a sample is identify by an id (key: *id*) and a strain name (key *strain*).

`wf.`**`samples_mnase`** **= []**
List of Mnase samples.

`wf.`**`strains`** **= []**
List of reference strains.

2.2.1 Creating Bowtie Index from each Reference Genome

For each strain, we need to create bowtie index. Bowtie index of a strain is a tree view of the genemoe reference for this strain. It will be used by bowtie to align reads. This step is performed by the following part of the *wf.py* script:

```
for strain in strains:
  per_strain_stats[strain] = create_bowtie_index(strain,
    config["FASTA_REFERENCE_GENOME_FILES"][strain], config["INDEX_DIR"],
    config["BOWTIE_BUILD_BIN"])
```

The following table sum up involved file sizes and process durations concerning this step.

strain	fasta genome file size	bowtie index file size	process duration
BY	12 Mo	25 Mo	11 s.
RM	12 Mo	24 Mo	9 s.
YJM	12 Mo	25 Mo	11 s.

2.2.2 Aligning Reads to Reference Genome

Next, we launch bowtie to align reads to the reference genome. It produces a *.sam* file that we convert into a *.bed* file. Binaries for *bowtie*, *samtools* and *bedtools* are wrapped using python *subprocess* class. This step is performed by the followinw part of the *wf.py* script:

```
for sample in samples:
  per_sample_align_stats["sample_%s" % sample["id"]] = align_reads(sample,
    config["ALIGN_DIR"], config["LOG_DIR"], config["INDEX_DIR"],
    config["ILLUMINA_OUTPUTFILE_PREFIX"], config["BOWTIE2_BIN"],
    config["SAMTOOLS_BIN"], config["BEDTOOLS_BIN"])
```

2.2.3 Convert Aligned Reads for TemplateFilter

TemplateFilter use particular input format for reads, so we convert *.bed* file. TemplateFilter expect reads as following: *chr coord strand #read* where:

- chr is the number of the chromosome;
- coord is the coordinate of the reads;
- strand is *F* for forward and *R* for reverse;
- #reads the number of reads for this position.

Each entry is *tab*-separated.

WARNING for reverse strand bowtie returns the position of left first nucleotid when TemplateFilter is waiting for right one. So this step takes it into account and add lenght of reads (in our case 50) to reverse reads coordinate.

This step is performed by the followinw part of the *wf.py* script:

```
for sample in samples:
  per_sample_convert_stats["sample_%s" % sample["id"]] = split_fr_4_TF(sample,
    config["ALIGN_DIR"], config["FASTA_INDEXES"], config["AREA_BLACK_LIST"],
    config["READ_LENGTH"],config["MAPQ_THRES"])
```

The following table sum up number of reads, involved file sizes and process durations concerning the two last steps. In our case, aligment process have been multuthreaded over over 3 cores.

id	Illumina reads	aligned and filtred reads	ratio	*.bed* file size	TF input file size	process duration
1	16436138	10199695	62,06%	1064 Mo	60 Mo	383 s.
2	16911132	12512727	73,99%	1298 Mo	64 Mo	437 s.
3	15946902	12340426	77,38%	1280 Mo	65 Mo	423 s.

Continued on next page

Table 2.1 – continued from previous page

id	Illumina reads	aligned and filtred reads	ratio	*.bed* file size	TF input file size	process duration
4	13765584	10381903	75,42%	931 Mo	59 Mo	352 s.
5	15168268	11502855	75,83%	1031 Mo	64 Mo	386 s.
6	18850820	14024905	74,40%	1254 Mo	69 Mo	482 s.
7	15591124	12126623	77,78%	1163 Mo	72 Mo	405 s.
8	15659905	12475664	79,67%	1194 Mo	71 Mo	416 s.
9	14668641	10960565	74,72%	1052 Mo	70 Mo	375 s.
10	14339179	10454451	72,91%	1049 Mo	51 Mo	363 s.
11	18019895	13688774	75,96%	1378 Mo	59 Mo	474 s.
12	13746796	10810022	78,64%	1084 Mo	54 Mo	360 s.
13	15205065	11766016	77,38%	990 Mo	54 Mo	381 s.
14	17803097	13838883	77,73%	1154 Mo	60 Mo	452 s.
15	15434564	12307878	79,74%	1032 Mo	57 Mo	394 s.
16	16802587	12725665	75,74%	1221 Mo	48 Mo	438 s.
17	16058417	12513734	77,93%	1192 Mo	63 Mo	422 s.
18	16154482	13204331	81,74%	1277 Mo	52 Mo	430 s.
19	21013924	17102120	81,38%	1646 Mo	59 Mo	555 s.
20	17213114	14433357	83,85%	1389 Mo	53 Mo	459 s.
21	17360907	14733001	84,86%	1203 Mo	55 Mo	450 s.
22	18136816	15389581	84,85%	1257 Mo	53 Mo	469 s.
23	14763678	12173025	82,45%	1140 Mo	56 Mo	393 s.
24	15541709	12890345	82,94%	1057 Mo	48 Mo	398 s.
25	16433215	13094314	79,68%	1241 Mo	57 Mo	433 s.
26	17370850	14264136	82,12%	1347 Mo	51 Mo	466 s.
27	14613512	8654495	59,22%	887 Mo	56 Mo	339 s.
28	15248545	11367589	74,55%	1166 Mo	67 Mo	405 s.
29	14316809	10767926	75,21%	1103 Mo	63 Mo	379 s.
30	15178058	12265794	80,81%	1030 Mo	66 Mo	390 s.
31	14968579	11876186	79,34%	1009 Mo	63 Mo	387 s.
32	16912705	13550508	80,12%	1143 Mo	70 Mo	442 s.
33	16782154	12755111	76,00%	1227 Mo	65 Mo	438 s.
34	16741443	13168071	78,66%	1260 Mo	71 Mo	442 s.
35	13096171	10367041	79,16%	992 Mo	62 Mo	350 s.
36	17715118	14092985	79,55%	1404 Mo	68 Mo	483 s.
37	17288466	7402082	42,82%	741 Mo	48 Mo	339 s.
38	16116394	13178457	81,77%	1101 Mo	63 Mo	420 s.
39	14241106	10537228	73,99%	880 Mo	57 Mo	348 s.
40	13784738	10598464	76,89%	1005 Mo	64 Mo	358 s.
41	12438007	9620975	77,35%	911 Mo	60 Mo	326 s.
42	13853959	11031238	79,63%	1045 Mo	64 Mo	365 s.
43	12036162	6654780	55,29%	684 Mo	46 Mo	268 s.
44	13873129	10251074	73,89%	1048 Mo	61 Mo	365 s.
45	19817751	14904502	75,21%	1520 Mo	72 Mo	528 s.
46	13368959	10818619	80,92%	912 Mo	63 Mo	350 s.
47	7566467	6139001	81,13%	520 Mo	44 Mo	201 s.
48	32586928	21191363	65,03%	1816 Mo	82 Mo	766 s.
49	30733184	18791373	61,14%	1801 Mo	89 Mo	721 s.
50	41287616	30383875	73,59%	2911 Mo	112 Mo	1065 s.
51	40439965	31177914	77,10%	2981 Mo	117 Mo	1070 s.
53	40876476	33780065	82,64%	3316 Mo	103 Mo	1165 s.
55	52424414	47117107	89,88%	3811 Mo	119 Mo	1477 s.

For some reasons (manipulation efficency, e.g. PCR...), we remove samples 33, 45, 48 and 55.

2.2.4 Run TemplateFilter on Mnase Samples

Finally, for each sample we perfome TemplateFilter analysis.

WARNING TemplateFilter returns a list of nucleosomes. Each nucleosome is define by its center and its width. An odd width leads us to considere non interger lower and upper bound.

WARNING TemplateFilter is not design to deal with replicate. So we choose to keep a maximum of nucleosome and filter in a second time using the benefit of replicate. To do that we set a low correlation threshold parameter (*0.5*) and a particularly high value of overlaping (*300%*).

This step is performed by the followinw part of the *wf.py* script:

```
for sample in samples_mnase:
  per_mnase_sample_stats["sample_%s" % sample["id"]] = template_filter(sample,
    config["ALIGN_DIR"], config["LOG_DIR"], config["TF_BIN"],
    config["TF_TEMPLATES_FILE"], config["TF_CORR"], config["TF_MINW"],
    config["TF_MAXW"], config["TF_OL"])
```

id	strain	found nucs	nuc file size	process duration
1	BY	96214	68 Mo	1022 s.
2	BY	91694	65 Mo	1038 s.
3	BY	91205	65 Mo	1036 s.
4	RM	88076	62 Mo	984 s.
5	RM	90141	64 Mo	967 s.
6	RM	87517	62 Mo	980 s.
7	YJM	88945	64 Mo	566 s.
8	YJM	88689	64 Mo	570 s.
9	YJM	88128	63 Mo	565 s.

2.3 Inferring Nucleosome Position and Extracting Read Counts

This preprocessing step consists in the 4 main steps embed in the *wf.py* and described bellow. As a preamble, this script computes *samples samples_mnase* and *strains* that will be used along the 4 steps.

The second part of the tutoriel use *R* (http://http://www.r-project.org). It consists in the 3 main steps corresponding to 4 R scripts:

- compute_rois.R
- extract_maps.R
- count_reads.R
- get_size_factors
- launch_deseq.R

2.3.1 Computing Common Genome Region Between Strains

```
R CMD BATCH src/nucleo_miner/compute_rois.R
```

2.3.2 Extracting Maps for Well Positionned and Fuzzy Nucleosomes

```
R CMD BATCH src/nucleo_miner/extract_maps.R
```

2.3.3 Count Reads for Each Nucleosome

```
R CMD BATCH src/nucleo_miner/count_reads.R
```

2.3.4 Get Size Factors Using DESeq

```
R CMD BATCH src/nucleo_miner/get_size_factors.R
```

2.3.5 Performing DESeq Analysis

```
R CMD BATCH src/nucleo_miner/launch_deseq.R
```

2.4 Results

2.4.1 Output Files Organisation

Previous steps produce following 45 files. Each filename is under the form

```
results/nucleo_miner/[combi]_[marker]_[form]_snep.tab
```

Where combi is in {BY_RM, BY_YJM, RM_YJM} for each strain combination, marker is in {H3K4me1, H3K4me3, H3K9ac, H3K14ac, H4K12ac} for each post translational histone modification and form is in {wp, fuzzy, wpfuzzy} considering well positionned nucleosomes, fuzzy nucleosomes or both for SNEP computation.

chr_BY lower_bound_BY upper_bound_BY index_nuc_BY chr_RM lower_bound_RM upper_bound_RM index_nuc_RM roi_index form BY_Mnase_Seq_1 BY_Mnase_Seq_2 BY_Mnase_Seq_3 RM_Mnase_Seq_4 RM_Mnase_Seq_5 RM_Mnase_Seq_6 BY_H3K14ac_36 BY_H3K14ac_37 BY_H3K14ac_53 RM_H3K14ac_38 RM_H3K14ac_39 pvalsGLM

For each file, there is 1 line per nucleosome and each line is composed of many columns divided into 3 main topics:

- nuc information
- number opf reads for each sample
- DESeq analysis results.

For exemple for the file *BY_RM_H3K14ac_wp_snep.tab* informations are:

- chr_BY, the BY chr involved
- lower_bound_BY, the lower bound of the BY nuc
- upper_bound_BY, the upper_bound of the BY nuc
- index_nuc_BY, the index of the nuc in the entire list of BY nucs
- **chr_RM, lower_bound_RM, upper_bound_RM, index_nuc_RM** are the same information for the RM strain

- roi_index, the index of the region of interrest involved.

Next cols concern indicators for each sample. They are labeled [strain]_[marker]_[sample_id] and each value represents the number of reads for the current nuc for the sample *sample_id*.

The 5 final columns concern DESeq analysis:

- manip[a_manip] strain[a_strain] manip[a_strain]:strain[a_strain], the manip (marker) effect, the strain effect and the snep effect.
- pvalsGLM, the pvalue resulting of the comparison of the GLM model considering or the interaction term *marker:strain*
- snep_index, a boolean set to TRUE if the *pvalueGLM* value is under the threshold computed with FDR function with a rate set to 0.01%.

It also produces the file that explicts size factor for each involved sample in differents strain combination and nucleosomal region type:

TODO: include this file... /home/filleton/analyses/snepcatalog/data/2013-10-09/nucleo_miner/README.txt

`results/nucleo_miner/size_factors.tab`

2.4.2 Number of SNEPs

Here are the number of computed for each forms.

```
[1] "wp"
       #nucs H3K4me1 H3K4me3 H3K9ac H3K14ac H4K12ac
BY-RM  30234     520     798     83    3566      26
BY-YJM 31298     303     619    102     103     128
RM-YJM 29863     129     340     46    3177      18
[1] "fuzzy"
       #nucs H3K4me1 H3K4me3 H3K9ac H3K14ac H4K12ac
BY-RM  10748     294     308    101    1681      42
BY-YJM 10669     122     176    124      93      87
RM-YJM 11478      54     112     41    1389      20
[1] "wpfuzzy"
       #nucs H3K4me1 H3K4me3 H3K9ac H3K14ac H4K12ac
BY-RM  40982     770    1136    183    5404      73
BY-YJM 41967     439     804    214     198     199
RM-YJM 41341     184     468     87    4687      37
```

TODO:

- Print/study intra/inter strain LODs.
- Check the normality of sample using Shapiro–Wilk (Hypothesis for computing LODs)

CHAPTER

THREE

REFERENCES

3.1 Python Reference

configurator.**CSV_SAMPLE_FILE = None**
Path to cvs file that contains sample information.

configurator.**BOWTIE_BUILD_BIN = None**
Path for bowtie2 build bin.

configurator.**BOWTIE2_BIN = None**
Path for bowtie2 bin.

configurator.**SAMTOOLS_BIN = None**
Path for samtools bin.

configurator.**BEDTOOLS_BIN = None**
Path for bedtools bin.

configurator.**TF_BIN = None**
Path for TemplateFilter bin.

configurator.**TF_TEMPLATES_FILE = None**
Path for TemplateFilter templates file.

configurator.**ILLUMINA_OUTPUTFILE_PREFIX = None**
Prefix for Illumina fastq output files.

configurator.**INDEX_DIR = None**
Path for index dir.

configurator.**ALIGN_DIR = None**
Path for align dir.

configurator.**LOG_DIR = None**
Path for log dir

configurator.**CACHE_DIR = None**
Path for cache dir.

configurator.**RESULTS_DIR = None**
Path for results dir

configurator.**FASTA_REFERENCE_GENOME_FILES = None**
Dictionary where each fasta reference genomes is indexed by reference strain that it corresponds.

configurator.**AREA_BLACK_LIST = None**
Dictionary where keys are strain and values are black listed of geneome region.

configurator.**FASTA_INDEXES = None**
Dictionary of strain that indexes dictionaries where keys are chromosome reference from Fastq file and value are its correspondance for Templatefilter.

configurator.**C2C_FILES = None**
Dictionary where each strain combination indexes genome aligment.

configurator.**READ_LENGTH = None**
Length of Illumina reads.

configurator.**MAPQ_THRES = None**
Aligment quality thresold.

configurator.**TF_CORR = None**
TemplateFilter Template correlation threshold.

configurator.**TF_MINW = None**
TemplateFilter minimum width of a nucleosome.

configurator.**TF_MAXW = None**
TemplateFilter maximum width of a nucleosome.

configurator.**TF_OL = None**
TemplateFilter maximum allowed overlap for two nucleosomes.

wf.**json_conf_file = 'src/nucleo_miner/nucleo_miner_config.json'**
Path to the json configuration file.

wf.**samples = []**
List of samples where a sample is identify by an id (key: *id*) and a strain name (key *strain*).

wf.**samples_mnase = []**
List of Mnase samples.

wf.**strains = []**
List of reference strains.

libcoverage.**create_bowtie_index**(*strain*, *strain_fasta_ref*, *index_dir*, *bowtie_build_bin*)
Creates bowtie index for a strain *strain*.

Parameters

- **strain** – the strain reference.
- **strain_fasta_ref** – fasta reference genome.
- **index_dir** – directories where to put bowtie index.
- **bowtie_build_bin** – bowtie2 build binary.

libcoverage.**align_reads**(*sample*, *align_dir*, *log_dir*, *index_dir*, *illumina_outputfile_prefix*, *bowtie2_bin*, *samtools_bin*, *bedtools_bin*)
Aligns reads to reference genomes. It produces .sam files, that are converted to .bam, that are converted to .bed.

Parameters

- **sample** – a dict that describe a sample.
- **align_dir** – directory where aligned reads will be stored.
- **log_dir** – directory where logs will be stored.
- **illumina_outputfile_prefix** – prefix of Illumina sequencer fastq.gz output files.
- **bowtie2_bin** – bowtie2 binary.
- **samtools_bin** – samtools binary.

- **bedtools_bin** – bedtools binary.
- **index_dir** – bowtie index directory.

libcoverage.**split_fr_4_TF**(*sample*, *align_dir*, *fasta_indexes*, *area_black_list*, *read_length*, *mapq_thres*)

Create TempleFilter input files form bed files. This function appends in two times. First, it collects reads from bed files and feeds a datastructure

Parameters

- **sample** – a dict that describe a sample.
- **align_dir** – directory where aligned reads will be stored.
- **fasta_index** – the chr reference from the illumina output file.
- **area_black_list** – the description of genome that will be omit.
- **read_length** – Length of Illumina reads.
- **mapq_thres** – mapping quality criterion threshold, see MAPQ in BED/BAM file format.

libcoverage.**template_filter**(*sample*, *align_dir*, *log_dir*, *tf_bin*, *tf_templates_file*, *corr*, *minw*, *maxw*, *ol*)

Run TemplateFilter on a specifi sample. It produces .tab file.

Parameters

- **sample** – a dict that describe a sample.
- **align_dir** – directory where aligned reads will be stored.
- **log_dir** – directory where logs will be stored.
- **tf_bin** – path to the TemplateFilter binary.
- **tf_templates_file** – path to the TemplateFilter templates file.
- **corr** – correlation threshold transmits to TemplateFilter.
- **minw** – minimum width of a nuc, transmits to TemplateFilter.
- **maxw** – maximum width of a nuc, transmits to TemplateFilter.
- **ol** – maximum overlaps for 2 nuc, transmits to TemplateFilter.

3.2 R Reference

3.2.1 Arabic to Roman pair list.

Description

Util to convert Arabicto Roman

Usage

```
ARAB2ROM()
```

Author(s)

Florent Chuffart

R: False Discovery Rate

3.2.2 False Discovery Rate

Description

From a vector x of independent p-values, extract the cutoff corresponding to the specified FDR. See Benjamini & Hochberg 1995 paper

Usage

```
FDR(x, FDR)
```

Arguments

x

A vector x of independent p-values.

FDR

The specified FDR.

Value

Return the the corresponding cutoff.

Author(s)

Gael Yvert, Florent Chuffart

Examples

```
print("example")
```

R: Roman to Arabic pair list.

3.2.3 Roman to Arabic pair list.

Description

Util to convert Roman to Arabic

Usage

```
ROM2ARAB()
```

Author(s)

Florent Chuffart

R: Aggregate replicated sample's nucleosomes.

3.2.4 Aggregate replicated sample's nucleosomes.

Description

This function aggregates nucleosome for replicated samples. It uses TemplateFilter ouput of each sample as replicate. Each sample owns a set of nucleosomes computed using TemplateFilter and ordered by the position of their center. Adajacent nucleosomes are compared two by two. Comparison is based on a log likelihood ratio score. The issue of comparison is adjacents nucleosomes merge or separation. Finally the function returns a list of clusters and all computed *lod_scores*. Each cluster ows an attribute *wp* for "well positionned". This attribute is set as *TRUE* if the cluster is composed of exactly one nucleosomes of each sample.

Usage

```
aggregate_intra_strain_nucs(samples, lod_thres = -20, coord_max = 2e+07)
```

Arguments

`samples`

A list of samples. Each sample is a list like *sample = list(id=..., marker=..., strain=..., roi=..., inputs=..., outputs=...)* with *roi = list(name=..., begin=..., end=..., chr=..., genome=...)*.

`lod_thres`

Log likelihood ration threshold.

`coord_max`

A too big value to be a coord for a nucleosome lower bound.

Value

Returns a list of clusterized nucleosomes, and all computed lod scores.

Author(s)

Florent Chuffart

Examples

```
# Dealing with a region of interest
roi =list(name="example", begin=1000,  end=1300, chr="1", genome=rep("A",301))
samples = list()
for (i in 1:3) {
    # Create TF output
    tf_nuc = list("chr"=paste("chr", roi$chr, sep=""), "center"=(roi$end + roi$begin)/2, "width"= 150
    outputs = dfadd(NULL,tf_nuc)
    outputs = filter_tf_outputs(outputs, roi$chr, roi$begin, roi$end)
    # Generate corresponding reads
    nb_reads = round(runif(1,170,230))
    reads = round(rnorm(nb_reads, tf_nuc$center,20))
    u_reads = sort(unique(reads))
    strands = sample(c(rep("R",ceiling(length(u_reads)/2)),rep("F",floor(length(u_reads)/2))))
    counts = apply(t(u_reads), 2, function(r) { sum(reads == r)})
    shifts = apply(t(strands), 2, function(s) { if (s == "F") return(-tf_nuc$width/2) else return(tf_
    u_reads = u_reads + shifts
    inputs = data.frame(list("V1" = rep(roi$chr, length(u_reads)),
                             "V2" = u_reads,
                                                    "V3" = strands,
                                                    "V4" = counts), stringsAsFactors=FALSE)
    samples[[length(samples) + 1]] = list(id=1, marker="Mnase_Seq", strain="strain_ex", total_reads =
}
print(aggregate_intra_strain_nucs(samples))
```

R: Aligns nucleosomes between 2 strains.

3.2.5 Aligns nucleosomes between 2 strains.

Description

This function aligns nucs between two strains for a given genome region.

Usage

```
align_inter_strain_nucs(replicates, wp_nucs_strain_ref1 = NULL,
    wp_nucs_strain_ref2 = NULL, corr_thres = 0.5, lod_thres = -100,
    config = NULL, ...)
```

Arguments

`replicates`

Set of replicates, ideally 3 per strain.

`wp_nucs_strain_ref1`

List of aggregates nucleosome for strain 1. If it's null this list will be computed.

`wp_nucs_strain_ref2`

List of aggregates nucleosome for strain 2. If it's null this list will be computed.

`corr_thres`

Correlation threshold.

`lod_thres`

LOD cut off.

`config`

GLOBAL config variable

`...`

A list of parameters that will be passed to *aggregate_intra_strain_nucs* if needed.

Value

Returns a list of clusterized nucleosomes, and all computed lod scores.

Author(s)

Florent Chuffart

Examples

```
    # Define new translate_roi function...
    translate_roi = function(roi, strain2, big_roi=NULL, config=NULL) {
      return(roi)
    }
    # Binding it by uncomment follwing lines.
    unlockBinding("translate_roi", as.environment("package:nucleominer"))
    unlockBinding("translate_roi", getNamespace("nucleominer"))
    assign("translate_roi", translate_roi, "package:nucleominer")
    assign("translate_roi", translate_roi, getNamespace("nucleominer"))
    lockBinding("translate_roi", getNamespace("nucleominer"))
    lockBinding("translate_roi", as.environment("package:nucleominer"))

# Dealing with a region of interest
roi =list(name="example", begin=1000,  end=1300, chr="1", genome=rep("A",301), strain_ref1 = "STRAINE
roi2 = translate_roi(roi, roi$strain_ref1)
replicates = list()
for (j in 1:2) {
    samples = list()
    for (i in 1:3) {
        # Create TF output
        tf_nuc = list("chr"=paste("chr", roi$chr, sep=""), "center"=(roi$end + roi$begin)/2, "width"=
        outputs = dfadd(NULL,tf_nuc)
        outputs = filter_tf_outputs(outputs, roi$chr, roi$begin, roi$end)
        # Generate corresponding reads
        nb_reads = round(runif(1,170,230))
        reads = round(rnorm(nb_reads, tf_nuc$center,20))
        u_reads = sort(unique(reads))
        strands = sample(c(rep("R",ceiling(length(u_reads)/2)),rep("F",floor(length(u_reads)/2))))
        counts = apply(t(u_reads), 2, function(r) { sum(reads == r)})
        shifts = apply(t(strands), 2, function(s) { if (s == "F") return(-tf_nuc$width/2) else return
        u_reads = u_reads + shifts
        inputs = data.frame(list("V1" = rep(roi$chr, length(u_reads)),
                                 "V2" = u_reads,
                                                        "V3" = strands,
                                                        "V4" = counts), stringsAsFactors=FALSE)
```

```
        samples[[length(samples) + 1]] = list(id=1, marker="Mnase_Seq", strain=paste("strain_ex",j,se
    }
    replicates[[length(replicates) + 1]] = samples
}
print(align_inter_strain_nucs(replicates))
```

R: Launch deseq methods.

3.2.6 Launch deseq methods.

Description

This function is based on deseq example. It mormalizes data, fit data to GLM model with and without interaction term and compare the two l;=models.

Usage

```
analyse_design(snep_design, reads)
```

Arguments

`snep_design`

The design to considere.

`reads`

The data to considere.

Author(s)

Florent Chuffart

R: Compute Common Uninterrupted Regions (CUR)

3.2.7 Compute Common Uninterrupted Regions (CUR)

Description

CURs are regions that can be aligned between the genomes

Usage

```
compute_inter_all_strain_curs(diff_allowed = 10, min_cur_width = 200,
    config = NULL, plot = FALSE)
```

Arguments

`diff_allowed`

the maximum indel width allowe din a CUR

`min_cur_width`

The minimum width of a CUR

`config`

GLOBAL config variable

`plot`

Plot CURs or not

Author(s)

Florent Chuffart

R: Crop bound of regions according to region of interest bound

3.2.8 Crop bound of regions according to region of interest bound

Description

The fucntion is no more necessary since we remove "big_roi" bug in translate_roi function.

Usage

```
crop_fuzzy(tmp_fuzzy_nucs, roi, strain, config = NULL)
```

Arguments

`tmp_fuzzy_nucs`

the regiuons to be croped.

`roi`

The region of interest.

`strain`

The strain to consider.

`config`

GLOBAL config variable

Author(s)

Florent Chuffart

R: Adding list to a dataframe.

3.2.9 Adding list to a dataframe.

Description

Add a list *l* to a dataframe *df*. Create it if *df* is *NULL*. Return the dataframe *df*.

Usage

```
dfadd(df, l)
```

Arguments

`df`

A dataframe

`l`

A list

Value

Return the dataframe *df*.

Author(s)

Florent Chuffart

Examples

```
## Here dataframe is NULL
print(df)
df = NULL

# Initialize df
df = dfadd(df, list(key1 = "value1", key2 = "value2"))
print(df)

# Adding elements to df
df = dfadd(df, list(key1 = "value1'", key2 = "value2'"))
print(df)
```

R: Extract wp nucs from nuc map.

3.2.10 Extract wp nucs from nuc map.

Description

Function based on common wp nuc index and roi_index.

Usage

```
extract_wp(strain_maps, roi_index, strain, tmp_common_nucs)
```

Arguments

`strain_maps`

Nuc maps.

`roi_index`

The region of interest index.

`strain`

The strain to consider.

`tmp_common_nucs`

the list of wp nucs.

Author(s)

Florent Chuffart

R: Prefetch data

3.2.11 Prefetch data

Description

Fetch and filter inputs and outpouts per region of interest. Organize it per replicates.

Usage

```
fetch_mnase_replicates(strain, roi, all_samples, config = NULL,
    only_fetch = FALSE, get_genome = FALSE, get_ouputs = TRUE)
```

Arguments

`strain`

The strain we want mnase replicatesList of replicates. Each replicates is a vector of sample ids.

`roi`

Region of interest.

`all_samples`

Global list of samples.

`config`

GLOBAL config variable

`only_fetch`

If TRUE, only fetch and not filtering. It is used tio load sample files into memory before forking.

`get_genome`

If TRUE, load corresponding genome sequence.

`get_ouputs`

If TRUE, get also ouput corresponding TF output files.

Author(s)

Florent Chuffart

R: Filter TemplateFilter inputs

3.2.12 Filter TemplateFilter inputs

Description

This function filters TemplateFilter inputs according genome area observed properties. It takes into account reads that are at the frontier of this area and the strand of these reads.

Usage

```
filter_tf_inputs(inputs, chr, x_min, x_max, nuc_width = 160,
    only_f = FALSE, only_r = FALSE)
```

Arguments

`inputs`

TF inputs to be filtered.

`chr`

Chromosome observed, here chr is an integer.

`x_min`

Coordinate of the first bp observed.

`x_max`

Coordinate of the last bp observed.

`nuc_width`

Nucleosome width.

`only_f`

Filter only F reads.

`only_r`

Filter only R reads.

Value

Returns filtred inputs.

Author(s)

Florent Chuffart

R: Filter TemplateFilter outputs

3.2.13 Filter TemplateFilter outputs

Description

This function filters TemplateFilter outputs according, not only genome area observerved properties, but also correlation and overlap threshold.

Usage

```
filter_tf_outputs(tf_outputs, chr, x_min, x_max, nuc_width = 160,
    ol_bp = 59, corr_thres = 0.5)
```

Arguments

`tf_outputs`

TemplateFilter outputs.

`chr`

Chromosome observed, here chr is an integer.

`x_min`

Coordinate of the first bp observed.

`x_max`

Coordinate of the last bp observed.

`nuc_width`

Nucleosome width.

`ol_bp`

Overlap Threshold.

`corr_thres`

Correlation threshold.

Value

Returns filtered TemplateFilter Outputs

Author(s)

Florent Chuffart

R: flat reads

3.2.14 flat reads

Description

Extract reads coordinates from TempleteFilter input sequence

Usage

```
flat_reads(reads, nuc_width)
```

Arguments

`reads`

TemplateFilter input reads

`nuc_width`

Width used to shift F and R reads.

Value

Returns a list of F reads, R reads and joint/shifted F and R reads.

Author(s)

Florent Chuffart

R: Retrieve Reads

3.2.15 Retrieve Reads

Description

Retrieve reads for a given marker, combi, form.

Usage

```
get_all_reads(marker, combi, form = "wp")
```

Arguments

`marker`

The marker to considere.

`combi`

The starin combination to considere.

`form`

The nuc form to considere.

Author(s)

Florent Chuffart

R: get comp strand

3.2.16 get comp strand

Description

Compute the complementatry strand.

Usage

```
get_comp_strand(strand)
```

Arguments

`strand`

The original strand.

Value

Returns the complementatry strand.

Author(s)

Florent Chuffart

R: Build the design for deseq

3.2.17 Build the design for deseq

Description

This function build the design according sample properties.

Usage

```
get_design(marker, combi, all_samples)
```

Arguments

`marker`

The marker to considere.

`combi`

The starin combination to considere.

`all_samples`

Global list of samples.

Author(s)

Florent Chuffart

R: Compute the fuzzy nucs.

3.2.18 Compute the fuzzy nucs.

Description

This function aggregate non common wp nucs for each strain and substract common wp nucs. It does not take care about the size of the resulting fuzzy regions. It will be take into account in the count read part og the pipeline.

Usage

```
get_fuzzy(combi, roi, roi_index, strain_maps, common_nuc_results,
    config = NULL)
```

Arguments

`combi`

The strain combination to consider.

`roi`

The region of interest.

`roi_index`

The region of interest index.

`strain_maps`

Nuc maps.

`common_nuc_results`

Common wp nuc maps

`config`

GLOBAL config variable

Author(s)

Florent Chuffart

R: Compute the list of SNEPs for a given set of marker, strain...

3.2.19 Compute the list of SNEPs for a given set of marker, strain combination and nuc form.

Description

This function uses

Usage

```
get_sneps(marker, combi, form, all_samples)
```

Arguments

`marker`

The marker involved.

`combi`

The strain combination involved.

`form`

the nuc form involved.

`all_samples`

Global list of samples.

Author(s)

Florent Chuffart

Examples

```
marker = "H3K4me1"
combi = c("BY", "YJM")
form = "wpfuzzy" # "wp" | "fuzzy" | "wpfuzzy"
# foo = get_sneps(marker, combi, form)
# foo = get_sneps("H4K12ac", c("BY", "RM"), "wp")
```

R: Likelihood ratio

3.2.20 Likelihood ratio

Description

Compute the likelihood log of two set of value from two models Vs. a unique model.

Usage

```
lod_score_vecs(x, y)
```

Arguments

x

First vector.

y

Second vector.

Value

Returns the likelihood ratio.

Author(s)

Florent Chuffart

Examples

```
# LOD score for 2 set of values
mean1=5; sd1=2; card2 = 250
mean2=6; sd2=3; card1 = 200
x1 = rnorm(card1, mean1, sd1)
x2 = rnorm(card2, mean2, sd2)
min = floor(min(c(x1,x2)))
max = ceiling(max(c(x1,x2)))
hist(c(x1,x2), xlim=c(min, max), breaks=min:max)
lines(min:max,dnorm(min:max,mean1,sd1)*card1,col=2)
lines(min:max,dnorm(min:max,mean2,sd2)*card2,col=3)
lines(min:max,dnorm(min:max,mean(c(x1,x2)),sd(c(x1,x2)))*card2,col=4)
lod_score_vecs(x1,x2)
```

R: nm

3.2.21 nm

Description

It provides a set of useful functions allowing to perform quantitative analysis of nucleosomal epigenome.

Details

Package:	nucleominer
Maintainer:	Florent Chuffart <florent.chuffart@ens-lyon.fr>
Author:	Florent Chuffart
Version:	2.3.1
License:	CeCILL
Title:	nm
Depends:	seqinr, plotrix, DESeq, cachecache

Author(s)

Florent Chuffart

R: Performaing ANOVAs

3.2.22 Performaing ANOVAs

Description

Counts reads and Performs ANOVAS for each common nucleosomes involved.

Usage

```
perform_anovas(replicates, aligned_inter_strain_nucs, inputs_name = "Mnase_Seq",
    plot_anova_boxes = FALSE)
```

Arguments

`replicates`

Set of replicates, each replicate is a list of samples (ideally 3). Each sample is a list like *sample = list(id=..., marker=..., strain=..., roi=..., inputs=..., outputs=...)* with *roi = list(name=..., begin=..., end=..., chr=..., genome=...)*. In the *perform_anovas* contexte, we need 4 replicates (4 * (3 samples)): 2 strains * (1 marker + 1 input (Mnase_Seq)).

`aligned_inter_strain_nucs`

List of common nucleosomes.

`inputs_name`

Name of the input.

`plot_anova_boxes`

Plot (or not) boxplot for each nuc.

Value

Returns ANOVA results and comunted reads.

Author(s)

Florent Chuffart

R: Plot the distribution of reads.

3.2.23 Plot the distribution of reads.

Description

This fuxntion use the deseq nomalization feature to compare qualitatively the distribution.

Usage

```
plot_dist_samples(strain, marker, res, all_samples, NEWPLOT = TRUE)
```

Arguments

`strain`

The strain to considere.

`marker`

The marker to considere.

`res`

Data

`all_samples`

Global list of samples.

`NEWPLOT`

If FALSE the curve will be add to the current plot.

Author(s)

Florent Chuffart

R: Remove wp nucs from common nucs list.

3.2.24 Remove wp nucs from common nucs list.

Description

It is based on common wp nucs index on nucs and region.

Usage

```
remove_aligned_wp(strain_maps, roi_index, tmp_common_nucs, strain)
```

Arguments

`strain_maps`

Nuc maps.

`roi_index`

The region of interest index.

`tmp_common_nucs`

the list of wp nucs.

`strain`

The strain to consider.

Author(s)

Florent Chuffart

R: sign from strand

3.2.25 sign from strand

Description

Get the sign of strand

Usage

```
sign_from_strand(strands)
```

Arguments

`strands`	

Value

If strand in forward then returns 1 else returns -1

Author(s)

Florent Chuffart

R: Substract to a list of regions an other list of regions that...

3.2.26 Substract to a list of regions an other list of regions that intersect it.

Description

This fucntion embed a recursive part. It occurs when a substracted region split an original region on two.

Usage

```
substract_region(region1, region2)
```

Arguments

`region1`

Original regions.

`region2`

Regions to substract.

Author(s)

Florent Chuffart

R: Switch a pairlist

3.2.27 Switch a pairlist

Description

Take a pairlist key:value and return the switched pairlist value:key.

Usage

```
switch_pairlist(l)
```

Arguments

`l`

The pairlist to switch.

Value

The switched pairlist.

Author(s)

Florent Chuffart

Examples

```
l = list(key1 = "value1", key2 = "value2")
print(switch_pairlist(l))
```

R: Translate a list of regions from a strain ref to another.

3.2.28 Translate a list of regions from a strain ref to another.

Description

This function is an eloborated call to translate_roi.

Usage

```
translate_regions(regions, combi, roi_index, config = NULL, roi)
```

Arguments

`regions`

Regions to be translated.

`combi`

Combination of strains.

`roi_index`

The region of interest index.

`config`

GLOBAL config variable

`roi`

The region of interest.

Author(s)

Florent Chuffart

R: Translate coords of a genome region.

3.2.29 Translate coords of a genome region.

Description

This function is used in the examples, usualy you have to define your own translation function and overwrite this one using *unlockBinding* features. Please, refer to the example.

Usage

```
translate_roi(roi, strain2, config = NULL, big_roi = NULL)
```

Arguments

`roi`

Original genome region of interest.

`strain2`

The strain in wich you want the genome region of interest.

`config`

GLOBAL config variable

`big_roi`

A largest region than roi use to filter c2c if it is needed.

Author(s)

Florent Chuffart

Examples

```
# Define new translate_roi function...
translate_roi = function(roi, strain2, config) {
    strain1 = roi$strain_ref
    if (strain1 == strain2) {
        return(roi)
    } else {
      stop("Here is my new translate_roi function...")
    }
}
# Binding it by uncomment follwing lines.
# unlockBinding("translate_roi", as.environment("package:nm"))
# unlockBinding("translate_roi", getNamespace("nm"))
# assign("translate_roi", translate_roi, "package:nm")
# assign("translate_roi", translate_roi, getNamespace("nm"))
# lockBinding("translate_roi", getNamespace("nm"))
# lockBinding("translate_roi", as.environment("package:nm"))
```

R: Aggregate regions that intersect themnselves.

3.2.30 Aggregate regions that intersect themnselves.

Description

This function is based on sort of lower bounds to detect regions that intersect. We compare lower bound and upper bound of the porevious item. This function embed a while loop and break break regions list become stable.

Usage

```
union_regions(regions)
```

Arguments

`regions`

The Regions to be aggregated

Author(s)

Florent Chuffart

R: Watching analysis of samples

3.2.31 Watching analysis of samples

Description

This function allows to view analysis for a particuler region of the genome.

Usage

```
watch_samples(replicates, read_length, plot_ref_genome = TRUE,
    plot_arrow_raw_reads = TRUE, plot_arrow_nuc_reads = TRUE,
    plot_squared_reads = TRUE, plot_coverage = FALSE, plot_gaussian_reads = TRUE,
    plot_gaussian_unified_reads = TRUE, plot_ellipse_nucs = TRUE,
    plot_wp_nucs = TRUE, plot_wp_nuc_model = TRUE, plot_common_nucs = TRUE,
    plot_anovas = FALSE, plot_anova_boxes = FALSE, plot_wp_nucs_4_nonmnase = FALSE,
    aggregated_intra_strain_nucs = NULL, aligned_inter_strain_nucs = NULL,
    height = 10, config = NULL)
```

Arguments

`replicates`

replicates under the form...

`read_length`

length of the reads

`plot_ref_genome`

Plot (or not) reference genome.

`plot_arrow_raw_reads`

Plot (or not) arrows for raw reads.

`plot_arrow_nuc_reads`

Plot (or not) arrows for reads aasiocied to a nucleosome.

`plot_squared_reads`

Plot (or not) reads in the square fashion.

`plot_coverage`

Plot (or not) reads in the covergae fashion. fashion.

`plot_gaussian_reads`

Plot (or not) gaussian model of a F anf R reads.

`plot_gaussian_unified_reads`

Plot (or not) gaussian model of a nuc.

`plot_ellipse_nucs`

Plot (or not) ellipse for a nuc.

`plot_wp_nucs`

Plot (or not) cluster of nucs

`plot_wp_nuc_model`

Plot (or not) gaussian model for a cluster of nucs

`plot_common_nucs`

Plot (or not) aligned reads.

`plot_anovas`

Plot (or not) scatter for each nuc.

`plot_anova_boxes`

Plot (or not) boxplot for each nuc.

`plot_wp_nucs_4_nonmnase`

Plot (or not) clusters for non inputs samples.

`aggregated_intra_strain_nucs`

list of aggregated intra strain nucs. If NULL, it will be computed.

`aligned_inter_strain_nucs`

list of aligned inter strain nucs. If NULL, it will be computed.

`height`

Number of reads in per million read for each sample, graphical parametre for the y axis.

`config`

GLOBAL config variable

Author(s)

Florent Chuffart

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *search*