pyicoteo Documentation

Release 2.0.3

Juan González-Vallinas

May 02, 2013

CONTENTS

Pyicoteo* is a suite of tools for the analysis of high-throughput sequencing data. It works with genomic coordinates, it was mainly developed using Solexa/Illumina mapped reads, but it it's core is platform-agnostic. There are currently 6 different tools (5 command-line based, one configuration file based) and a python library for scripting:

```
* Pronounced as in Spanish  "picoteo"_ /pkt/:
  (n) Appetizer-type foods that accompany drinks before or instead of a meal.
```

If you have any problems or suggestions please join the Pyicoteo Google Group and ask!

CHAPTER

ONE

GETTING STARTED

1.1 Download & Install

Download Pyicoteo Latest version from our repository.

You can also download older versions (up to 1.1b) from our Sourceforge repository.

The command line tools can be used directly without installation. However, installation is recommended, and necessary if you intend to use the Pyicoteolib. To do so decompress the folder and run the setup.py script with administrator privileges:

```
python setup.py install
```

1.2 Check installation

To test that the software was installed correctly, start a python console and try importing it by typing:

```
python
>>> import pyicoteolib
>>> import pyicoteolib.core
```

Also, you should find the tools in your command line.

Tip: If you are in a GNU/Linux system, type "pyico" in the command line and tap TAB twice. You should see something like this:

```
pyicoclip        pyicoller        pyicos
pyicoenrich      pyicoregion      pyicotrocol
```

1.3 Dependencies

In order to make it simple for the community, Pyicoteo basic functionality has no dependencies other than Python 2.6 or higher. However, there are 2 optional libraries you could install.

1.3.1 Matplotlib

For plotting capabilities, it is neccesary to install Matplotlib (> 1.0).

1.3.2 Samtools

Also, for BAM reading, while we offer a native python implementation, you can ask Pyicoteo to read BAM using samtools with the flag `--samtools`. Pyicoteo is not compatible with Python 3.

1.4 The bedpk format

Some Pyicoteo tools (Pyicos, Pyicaller and Pyicoclip) default experiment and output formats is a derivative of UCSC Bed format called bedpk. It follows the same starting fields "chromosome/tag start end" but it uses some of the original optional fields to include extra information. It is a cluster oriented format that aims to recollect information of a cluster of reads in a single comprehensive line.

Figure 1.1: The bedpk format specification. It is exactly like a BED6 format, but using the 4th column to store extra information about how the cluster was built.

1.4.1 bedpk Column definition

1. Chromosome
2. Start coordinate
3. End coordinate
4. Profile: This field summarizes the accumulation of reads per nucleotide of the cluster. The first number is the number of bases covered, while the second will be the number of reads in those bases. See the example above
5. Height: The maximum height of the cluster. In this case, 3.
6. Strand: if ALL reads that are positive strand +, if they are all negative −. Otherwise .
7. Summit: The position where the maximum height is found. The binding site is expected to be close to the summit.
8. Area: The area covered by the cluster.
9. p-value: The significance of the cluster calculated by the poisson operation based on peak heights or numbers of reads.

CHAPTER

TWO

PYICOTEOLIB

A python library that is the building blocks of all the other tools and a useful tool for python scripting.

Read more about it at:

2.1 Pyicoteolib.core

Pyicoteolib is the library and the building blocks of the Pyicoteo suite. The pyicoteolib.core library contains the main holders of data in the

2.1.1 ReadCluster

A ReadCluster object may contain one read or a group of **overlapping** reads. It can read both tag like (bed, sam, bam..) and histogram like (wig, bed_pk...) formats. Instances of the ReadCluster object can be added, compared, subtracted to other readCluster objects with standard python syntax.

The ReadCluster object is optimized in order to deal with millions of overlaps, and has been tested with multiple different HTS datasets. The optimization consists in 2 main principles:

Common python operators

All the following standard operators are supported:

Adding

Adding combines the signal of 2 different ReadClusters, with nucleotide precision:

```
cluster1 = ReadCluster(read=PK)
cluster2 = ReadCluster(read=PK)
cluster1.read_line('chr1 1 45 9:2.00|41:3.00|50:2.00|45:1.00')
cluster2.read_line('chr1 1 125 9:4.00|41:3.00|30:2.00|45:1.00')
result = cluster1 + cluster2

result.write_line()
```

```
        chr1    1       145     50:6.00|30:4.00|20:3.00|25:2.00|20:1.00 6.0     .       25      550.0
```

Subtracting

Substracts the signal of 2 different ReadClusters, with nucleotide precision:

```
cluster1 = ReadCluster(read=SAM)
cluster2 = ReadCluster(read=PK)
cluster1.read_line('SL-XAJ_1_FC305HJAAXX:2:21:872:1402  0       chr1    1       50      36M     *
cluster2.read_line('chr1 1 125 9:4.00|41:3.00|30:2.00|45:1.00')
result = cluster2 - cluster1

result.write_line()
```

Length

Returns the length of the read cluster:

```
c = Cluster(name="chrX", start=1, end=10000)
len(c)

10000
```

Comparison operators (< > == !=)

This indicates which read cluster is before another in a chromosome:

```
c1 = Cluster(name="chr1", start=100, end=1000)
c1_copy = Cluster(name="chr1", start=100, end=1000)
c2 = Cluster(name="chr1", start=50000, end=100000)

c1 > c2
False
c1 == c1_copy
True
```

Lets see some usage examples.

Read a .bed file, print the chromosome and the length of each read

```
from pyicoteolib.core import ReadCluster, BED

bed_file = open("/path/to/myfile.bed")

for line in bed_file:
        rc = ReacCluster(read_as=BED)
        rc.read_line(l)
        print len(rc), rc.area()
```

Read some .bed lines, cluster them, output a wiggle file

cluster = Cluster(read=BED) cluster.read_line('chr1 1 20000 666 hola +') cluster.read_line('chr1 1 20000 666 hola +') cluster.read_line('chr1 1 20000 666 hola +') cluster.read_line('chr1 1001 20000 666 hola +') cluster.write_line()

ReadRegion

A ReadRegion object holds a genomic region that may contain ReadClusters

2.1.2 pyicoteolib.utils

2.1.3 BAM reader

2.1.4 Credit

- Developers: Juan González-Vallinas, Ferran Lloret
- Unit and beta Testing: Juan González-Vallinas, Ferran Lloret
- Supervision: Eduardo Eyras

CHAPTER

THREE

COMMAND-LINE BASED TOOLS

3.1 pyicos command line operations

Pyicos is a command line utility for the conversion and manipulation of genomic coordinates files. It follows a command/sub-command structure

In the interactive help you can visualize the available commands list:

```
pyicos -h
```

If you are interested in the usage of a particular command (for example, 'extend') and the meaning of its flags type:

```
pyicos extend -h
```

Here we explain briefly what each subcommand does and we give some examples:

3.1.1 convert

Converting a file from one format to another format. Currently supported format are:

experiment: Bed, Wiggle files (bed_wiggle), SAM, BAM, Eland, bedpk (Pyicos default compressed format), bedspk (Pyicos stranded compressed format)

output: Bed, Wiggle files (bed_wiggle, variable_wiggle), SAM, BAM, Eland, bedpk (Pyicos default compressed format), bedspk (Pyicos stranded compressed format)

This operation is useful if you only want to convert your data to another format. Other operations already include a conversion if you specify different experiment and output formats.

Convert a bed file to a half-open variable wig file:

```
pyicos convert my_experiment.bed my_experiment.wig -f bed -F variable_wig -O
```

Convert all pk files in a folder to bed wig files:

```
pyicos convert my_experiment_folder/ outputfolder/ -f pk -F bed_wig
```

3.1.2 remregions

Remove regions that overlap with the regions in the "black list" file.

Example:

```
pyicos remregions my_experiment.bed regions.bed my_result.bed --experiment-format bed --open-experime
```

3.1.3 remduplicates

Remove the duplicated reads in a file. A duplicate is a read with the same start position as a read that has already been seen. You can choose how many duplicates you want to tolerate. If you want to keep only one read for a start position, set the duplicates to 0.

Example:

Here we tolerate 1 duplicate so a read can not occur more often than twice:

```
pyicos remduplicates my_experiment.bed my_experiment_1dupl.bed --duplicates 1 -f bed -o -F bed
```

3.1.4 strcorr (Strand Correlation)

Finds the optimal extension value by finding the "gap" between groups of positive and negative cluster of reads by performing a pearson correlation test.

3.1.5 extend

Extend the reads to the estimated fragment length, taking into consideration if they map to the forward or reverse strand of the reference genome.

Examples:

We have a bed file (half open) with reads between 30 and 50 nucleotides long. We want to extend them all to 150 nucleotides and write the output in bedpk-format to accelerate the successive operations:

```
pyicos extend my_experiment.bed my_experiment_ext.bedpk 150 -f bed -o
```

We do the same with the control:

```
pyicos extend control.bed control_ext.bedpk 150 -f bed -o
```

To visualize the data in a genome browser we set the output to be half-open bed_wig:

```
pyicos extend my_experiment.bed my_experiment_ext.bed_wig 150 -f bed -o -F bed_wig -O
```

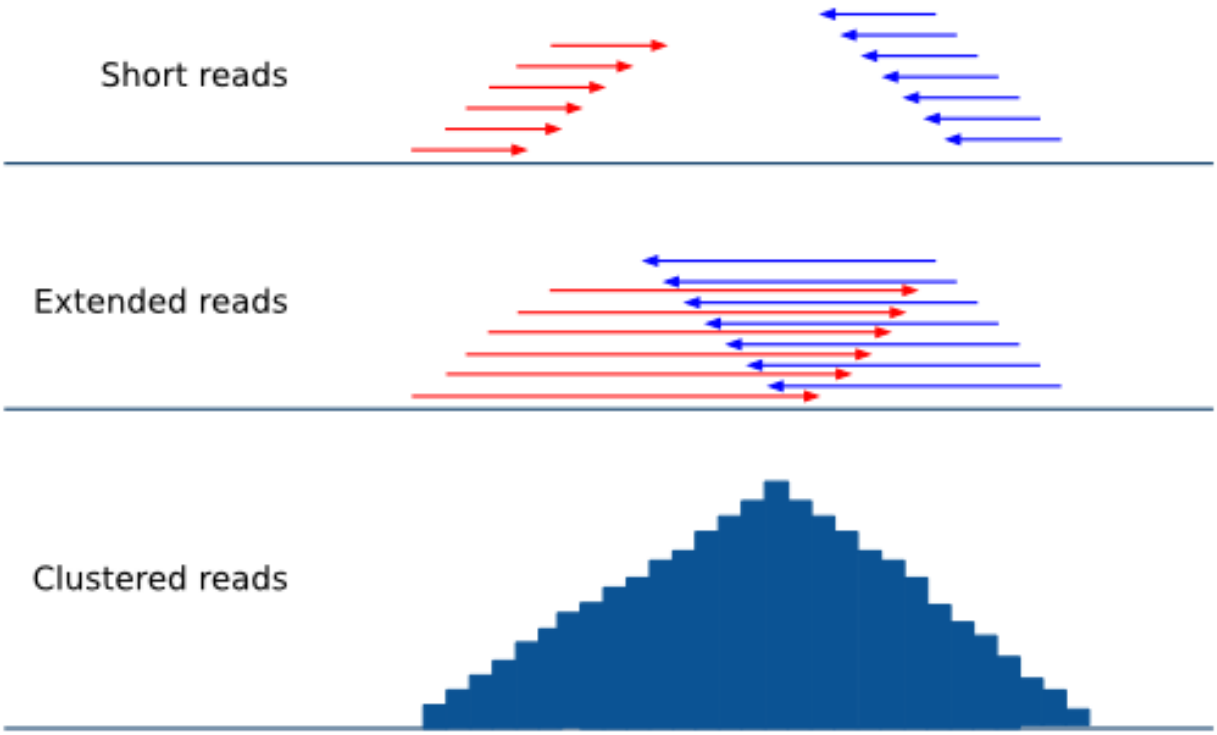
3.1.6 subtract

Subtract the reads in one file from the reads in another file. Using background data (control) improves the results because the background distribution is not supposed to be normal, and statistical approaches to obtain this have a limited reach.

The most straightforward approach is to subtract the control from the sample. Make sure the sample has been **normalized** to the control beforehand! Pyicos maintains a 1bp resolution by subtracting the reads nucleotide by nucleotide, rather than doing a statistical approximation. Operating with directories will only give appropiate results if the files and the control are paired in alphabetical order.

Example:

Subtract the control from the experiment (both have already been extended, converted to bedpk and normalized):


Short reads
Extended reads
Clustered reads

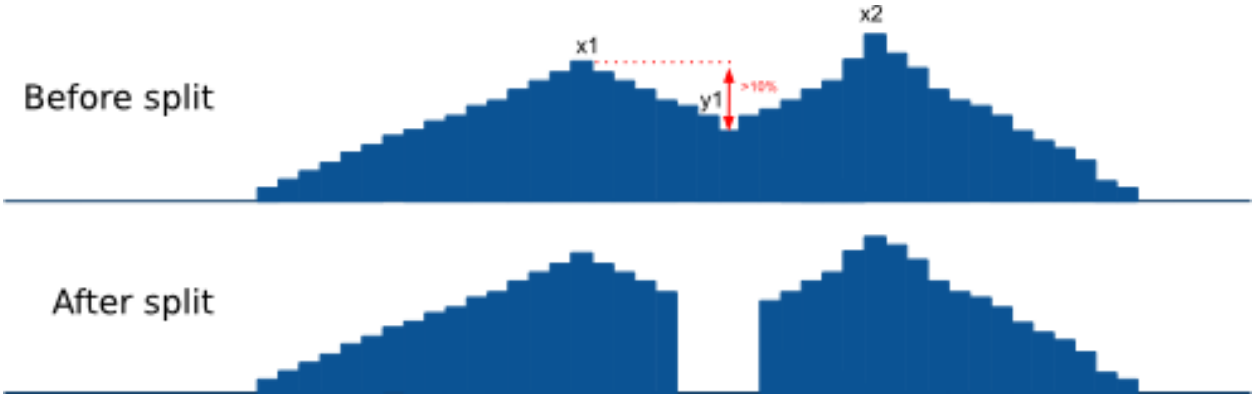
```
pyicos subtract my_experiment_ext_norm.bedpk control_ext.bedpk my_experiment_ext_norm_subtr.bedpk
```

3.1.7 split

Split peaks into subpeaks in case they fulfill the criteria.

Criteria: peak has at least two neighboring maxima between which the coverage of reads falls below the threshold. The threshold can be set by the user and it reflects a proportion of the lower maximum.

Output: bedpk or Wiggle files



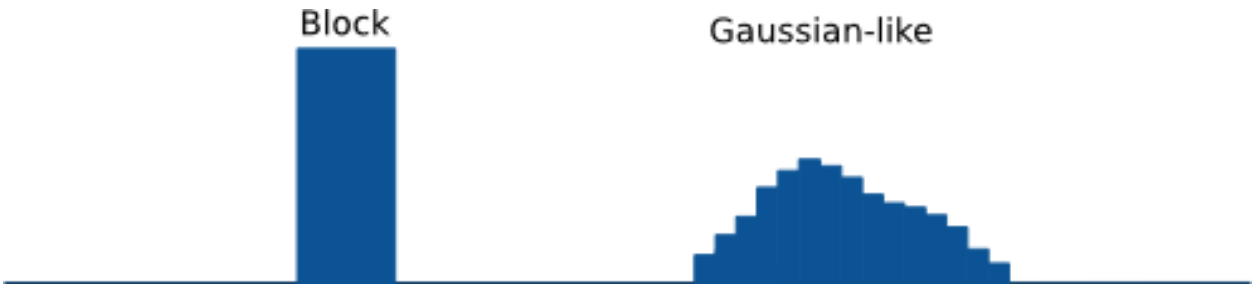
Example:

```
pyicos split peaks.bedpk peaks_split.bedpk --split-proportion 0.9
```

3.1.8 discard

Discards peaks that look like artifacts due to the sequencing bias. Here we refer to peaks that look like blocks that result from duplicates (reads with the same start position).

Output: bedpk or Wiggle files



Example:

```
pyicos discard peaks.bedpk peaks_discA.bedpk
```

3.1.9 poisson

This is the test to assess significance of peaks along the whole genome (as for ChIP-Seq). We do 3 different global poisson statistical tests for each chromosome in a file:

Max height analysis

Lambda is calculated from the maximum heights of the clusters by calculating the average height of a cluster in a given region. Pyicos will obtain the p-value_height of one cluster having a height k by chance.

Number of reads analysis

Lambda is obtained from the number of reads in clusters.

Nucleotide analysis

Lambda is obtained from the number of nucleotides in a cluster.

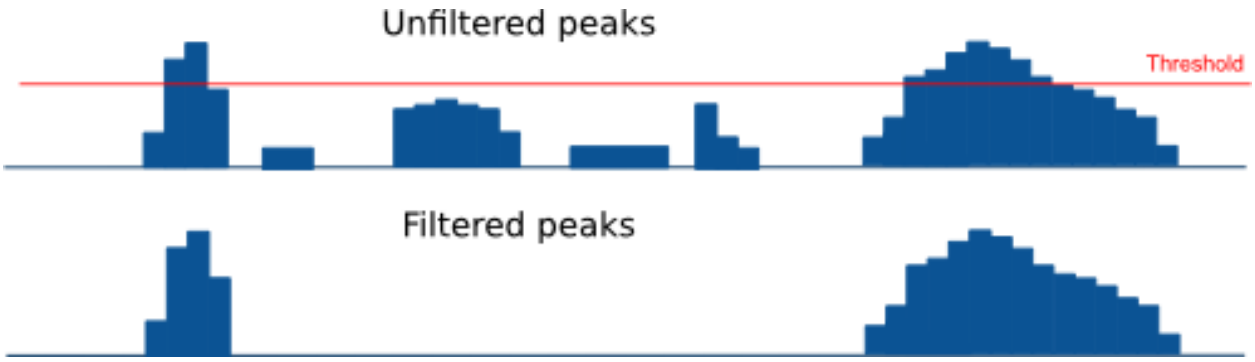
Example:

```
pyicos poisson peaks.bedpk
```

3.1.10 filter

Detect and select significant clusters in the file. There are two steps in this operation: Through the poisson operation the thresholds are determined. Next the peaks are filtered according to these thresholds.

Output: bedpk or Wiggle files



Example:

```
pyicos filter peaks.bedpk significant_peaks.bedpk 150
```

3.1.11 push

Push the reads in the corresponding strand. If a read doesn't have a strand, it will be pushed from left to right.

This operation requires tag-like files (bed, eland, sam).

Example:

```
pyicos push my_experiment.bed my_experiment_pushed100.bed 100 -f bed -F bed
```

3.1.12 Credit

- Developer: Juan González-Vallinas
- Beta Testing: Eneritz Agirre, Sonja Althammer, Juan González-Vallinas
- Supervision: Eduardo Eyras

3.2 Pyicoller

This peak caller is a combinations of some of Pyicos commands (extend, normalize, subtract, remove, poisson and filter) for the task of calling peaks from a ChIP-Seq experiment (with narrow peaks). A control file is optional but recommended.

Example:

```
pyicoller my_experiment.bed significant_peaks.bedpk -f bed -o --control control.bed --control-format
```

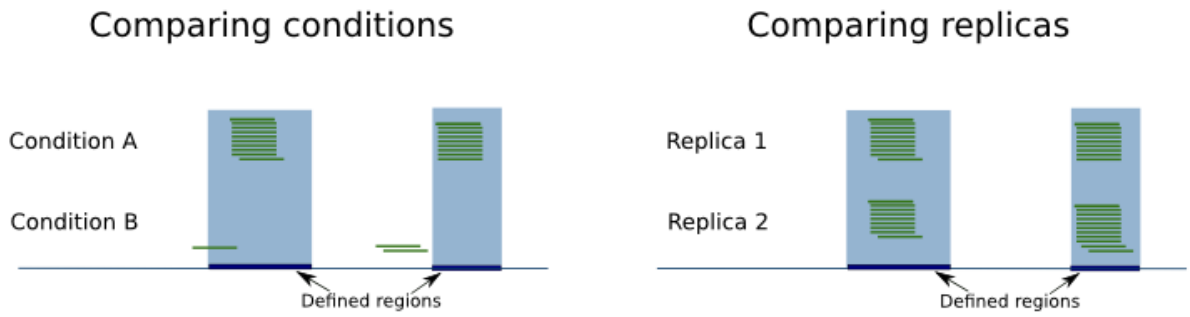
3.2.1 Credit

- Developer: Juan González-Vallinas
- Beta testing: Sonja Althammer, Eneritz Agirre, Nuria Conde Pueyo
- Benchmarking against other peak callers: Sonja Althammer
- Performance benchmarking: Juan González-Vallinas

3.3 Pyicoenrich

3.3.1 Introduction

Enrichment analysis can be applied on any type of -seq data. Pyicoenrich performs enrichment analysis on sequenced reads from two conditions. Like this you can find out how significant the difference of these two conditions is, in terms of the number/density of reads overlapping a region of interest.



3.3.2 MA Plot

Pyicoenrich is based on the MA-Plot.

(Figura)

3.3.3 Region exploration

If a region file is provided, Pyicoenrich returns for each region a Z-Score (See counts file description) which indicates the enrichment/depletion of condition A over condition B. If no region file is provided, Pyicoenrich provides the

options to take the union of reads from both conditions as a region and gives back Z-Scores for the generated regions. As regions with 0 reads in one condition might be especially interesting.

In order to decide what regions are to be explored, you have 3 main options:

Generate a file with the –region-magic flag and GTF file

See *Pyicoregion* for examples on how to use `--region-magic` flag to automatically explore exons, introns and the whole genome using sliding windows automatically from GTF files.

Provide a regions file

If a region file is provided, Pyicoenrich returns for each region a z-Score (among others) which indicates the enrichment/depletion of condition A over condition B. The region file should be in BED format. Also, you may consider only discontinuous regions by using the BED12 format:

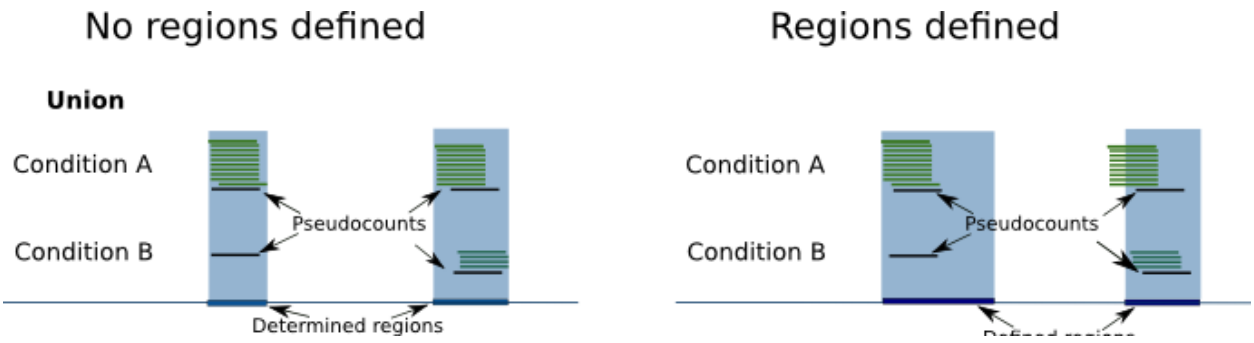
```
pyicoenrich -reads kidney1.bed liver1.bed -output Pyicoenrich_Kidney_Liver_result_Counts -f bed --reg
```

Do nothing

Don't really know where you want to look yet? If no region file is provided, Pyicoenrich will automatically generate one with taking he union of reads from both conditions as a region and gives back Z-Scores for the generated regions.

The flag `--proximity` controls the distance with which the regions are considered "joined". Default is 50nt:

```
pyicoenrich -reads kidney1.bed liver1.bed -output Pyicoenrich_Kidney_Liver_result -f bed --proximity
```



3.3.4 `--pseudocounts` flag

As regions with 0 reads in one condition might be especially interesting, Pyicoenrich can use pseudocounts, in order to avoid a division by 0: Pyicoenrich calculates the ratio of number of reads in both conditions. As there might not be any reads in a region, Pyicoenrich assumes that there is already 1 read in each region in each condition.

3.3.5 `--stranded` flag

3.3.6 Replica or technical control (swap)

To calculate the Z-Score, Pyicoenrich compares the differences between condition A and condition B with the differences between A and A' (while A' is the biological replica of A). If no biological replica is available, Pyicoenrich uses

a sample swap as a reference. With sample swap we mean that reads from condition A and B are mixed randomly and divided in two sets (with size of those of A and B). In the two resulting sets we do not expect any significant differences, just like in replicas.

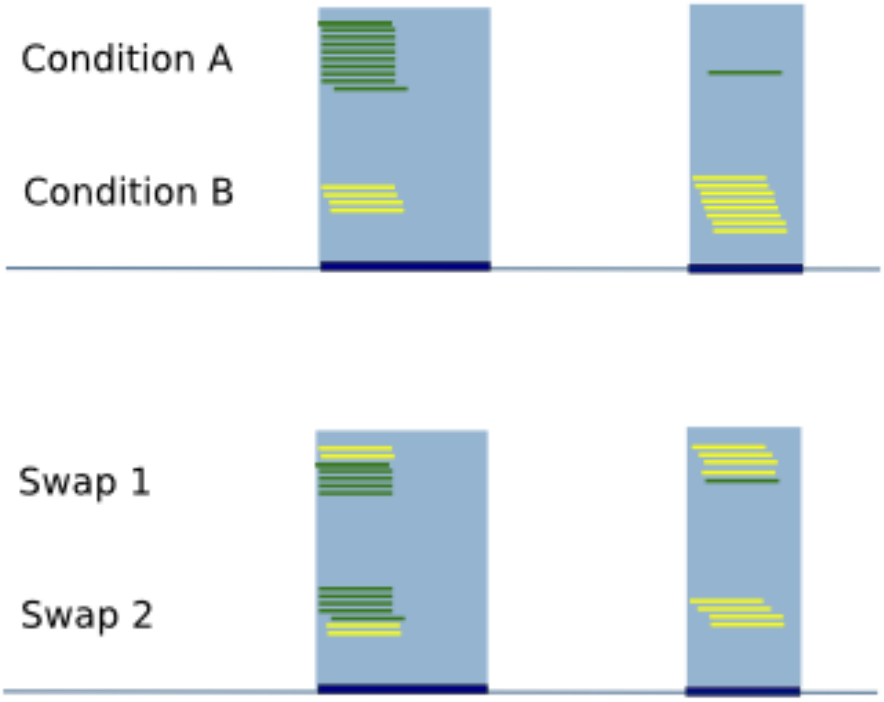


Figure 3.1: Technical replica (swap) illustration

Examples

With replica:

```
pyicoenrich -reads kidney1.bed liver1.bed -output n_norm.enrich -f bed --region genes.bed --replica
```

Using a swap:

```
pyicoenrich -reads kidney1.bed liver1.bed -output n_norm.enrich -f bed --region genes.bed
```

3.3.7 Description of the pyicoenrich counts file

Column description of enrichment result where each line describes a region:

```
TIP: If you want to provide pyicoenrich with your own counts file, you only need to provide up to co
```

1. name = chromosome of region
2. start = region start
3. end = region end

4. name2 = alternative label for the region, useful to put the gene name on it
5. score = Reserved by a "." as it is used by the UCSC browser for coloring.
6. strand = region strand
7. signal_a = Counts in experiment A (normalized if used)
8. signal_b = Counts in experiment B (normalized if used)
9. signal_prime_1 = Counts in experiment A (exactly the same as signal_a) or random background 1 (normalized if used)
10. signal_prime_2 = Counts in experiment replica A or random background 2 (normalized if used)
11. A = (log2(signal_a)+log2(signal_b))/2
12. M = log2(signal_a/signal_b)
13. total_reads_a = total number of reads in sample a
14. total_reads_b = total number of reads in sample b
15. num_tags_a = number of reads in sample a overlapping the region
16. num_tags_b = number of reads in sample b overlapping the region
17. A_prime = (log2(signal_prime_1)+log2(signal_prime_2))/2
18. M_prime = log2(signal_prime_1/signal_prime_2)
19. total_reads_a = total number of reads in sample a
20. total_reads_b = total number of reads in sample b
21. total_reads_prime_1 = total number of reads in sample prime 1
22. total_reads_prime_2 = total number of reads in sample prime 2
23. A_median = median of A values in window
24. mean = mean of M_prime values in window
25. sd = standard deviation of M_prime values in window
26. zscore = score for the significance of the difference of enrichment between condition a and b compared to prime 1 and prime 2

3.3.8 Normalization methods

Pyicoenrich included several popular normalization methods for the counts.

> **PUBLIC SERVICE ANNOUNCEMENT:** When dealing with normalization methods, one has to be very careful. There is no silver bullet, you need to understand your data and then apply the method that is appropriate for it. If you are in doubt, please consult your local statistician.

Total reads normalization (`--n-norm`)

This normalization will calculate the *number of reads per million reads* in each region and sample. This is a *very simple* normalization that tries to correct the bias of comparing different samples by total number of reads. You can activate it with the `--n-norm` flag.

Example. Using 2 reads files, calculate the enrichment normalizing by N

```
pyicoenrich -reads kidney1.bed liver1.bed -output n_norm.enrich -f bed --region genes.bed --n-norm
```

If you want to skip the total reads calculation step, you can provide the total number of reads with the following flags.

-total-reads-a

-total-reads-b

-total-reads-replica

Example:

```
pyicoenrich -reads kidney1.bed liver1.bed -output n_norm.enrich -f bed --region genes.bed --n-norm --
```

Region length normalization (--len-norm)

Calculates the number of reads per **region** kilobase. It aims to correct for regions with different lengths.

NOTE: If possible, try not to mix regions with different lengths.

pyicoenrich -reads kidney1.bed liver1.bed -output n_norm.enrich -f bed –region genes.bed –n-norm

RPKM (--len-norm and --n-norm)

The popular RPKM normalization is the combination of both `--n-norm` and `--len-norm`:

```
pyicoenrich -reads kidney1.bed liver1.bed -output rpkm_norm.enrich -f bed --region genes.bed --n-norm
```

Trimmed Means of M values normalization (--tmm-norm)

As proposed by EdgeR.

This calculates the weighted trimmed mean of the log expression ratios (trimmed mean of M values (TMM)). It is based on the hypothesis that most of your regions do not change, and calculates a normalization factor by excluding the total amount of data.

Important flags.

-a-trim
: Proportion of A values to be discarded when doing the

  TMM normalization. [Default 0.05]

-m-trim
: Proportion of M values to be discarded when doing the

  TMM normalization. [Default 0.25]

Example: TMM normalization calculated discarding the 20% smaller A (less read coverage) and 5% of the regions with the biggest differences (up and down):

```
pyicoenrich -reads kidney1.bed liver1.bed -output rpkm_norm.enrich -f bed --region genes.bed --tmm-no
```

Full quantile normalization (--quant-norm)

This method is suitable when your samples have too much variability. As eloquently put by Simplystatistics

3.3.9 --interesting-regions

Providing a list of interesting regions matching a the 4th column of the region or count file will highlight them in the MA plot.

(Falta figura)

Example:

```
"""
Region file (regions.bed)
chr1 1 100     region1 0 .
chr1 1000 1100 region2 0 .
chr2 1 100     region3 0 .
...
chrN x y       regionN 0 .

Interesting regions file (interreg.txt)
region4
region10
...
regionZ
"""
pyicoenrich -reads kidney1.bed liver1.bed -output rpkm_norm.enrich -f bed --region genes.bed --tinter
```

3.3.10 Credit

- Developers: Juan González-Vallinas, Ferran Lloret
- Beta Testing: Sonja Althammer, Eneritz Agirre, Nuria Conde Pueyo, Juan González-Vallinas
- Benchmarking against other DE methods: Sonja Althammer
- Speed and memory performance benchmarking: Juan González-Vallinas
- Supervision: Eduardo Eyras

3.4 pyicoclip

3.4.1 Introduction

Pyicoclip is an implementation of the modified False Discovery Rate algorithm proposed by Yeo et al. to determine which clusters are significant in a list of genomic regions (like genes or transcripts). This method is typically used in CLIP-Seq data that doesn't have a valid control experiment to compare against.

Theoretically, it could be used for any other kind of experiment that involves short reads and doesn't have a valid control.

3.4.2 Region file

You can provide your own region file, or otherwise you can provide a –region-magic description with a GTF file.

Example:

```
pyicoclip my_experiment.bed my_regions.bed output.pk -f bed
```

3.4.3 Credit

- Developer: Juan González-Vallinas
- Beta Testing: Mireya Plass, Juan González-Vallinas
- Supervision: Eduardo Eyras

3.5 Pyicoregion

Pyicoregion is a Pyicoteo module for processing region files.

It uses GFF files as specified in http://www.sanger.ac.uk/resources/software/gff/spec.html

3.5.1 Pyicoregion arguments

-region-magic

exons [position]
Returns all the exons in the region file.

If the optional argument `[position]` is specified (possible values: *first*, *last*), it will only return the first or last exon of every gene.

introns [position]
Returns all the introns in the region file.

If the optional argument `[position]` is specified (possible values: *first*, *last*), it will only return the first or last intron of every gene.

slide <window_size> <window_step> <region_type> [chromlen_file_path]
Searches for intergenic and intragenic regions using sliding windows.

Mandatory arguments are `<window_size>` (the size of the sliding window), `<window_step>` (the distance between the start position of every consecutive window. It must be lower than or equal to the window size) and `<region_type>` (must be *inter*, for intergenic, or *intra*, for intragenic regions).

The optional argument `[chromlen_file_path]` is used to specify the path to the file containing the chromosome lengths (Pyicoteo's own chromlen files can be found in pyicoteolib/chromlen/). If it is not specified for intergenic regions, the results for the last regions of the chromosomes might be wrong.

Note: if the last segment of a region is shorter than the window size, the step distance is decreased by the difference (the window size stays the same).

Note: regions shorter than the window size are ignored.

tss <add_start> <add_end>
Returns the TSS for every transcript in the region file.

Due to a TSS being a single point, the arguments *add_start* and *add_end* specify the values added to the start and end of every TSS (taking into consideration the strand). For pyicoregion to work correctly, they must be non-negative integers.

-gff-file `<gff_file_path>`
Used to specify the path of the GFF file containing the regions. This argument is mandatory for all operations involving regions.

CHAPTER

FOUR

PROTOCOL FILES

A configuration file based tool that exposes most functionality of the Pyicoteo suite, making it very useful when trying to combine different tools (for example, Pyicos and Pyicosenrich functionality)

Read more about it at *protocoldocs*