RiboPy

Release 0.0.0

Dec 20, 2019

Contents:

RiboPy is a set of tools to work with ribo files. Ribo is a file format designed specifically for ribosome profiling data. Ribo files contain essential quantitative entities such as metagene coverage and 5'UTR, CDS and 3'UTR counts. Ribo files are used for downstream analysis of ribosome profiling data. RiboPy provides interfaces to read and write ribo files as well as generating plots.

CHAPTER 1

Installation

1.1 General Advice

We strongly recommend that users install conda first. Then, using conda, one can install RiboPy in a separate conda environment.

For example, the following command will install RiboPy inside a conda environment named *ribo*.

```
conda env create -f environment.yaml
```

The environment file above, *environment.yaml*, can be obtained from the RiboFlow repository.

Also, we highly encourage using conda for the installation of scientific Python packages: numpy, pandas and h5py.

1.2 Requirements

RiboPy is a python package and requires Python 3.6 or a later version. The python packages, that RiboPy requires, are automatically installed by pip. The command line interface (CLI) requires terminal access.

1.3 Using pip

```
pip install ribopy
```

1.4 From Github

Alternatively, you can install the latest version from github.

```
pip install git+https://github.com/ribosomeprofiling/ribopy.git
```

CHAPTER 2

Quickstart

There are two interfaces, available in RiboPy, to work with ribo files: Command Line Interface(CLI) and Application PRogramming Interface (API).

A walk-through of Python API

A walk-through of Python CLI

Also, for R users, a walkthrough of RiboR is in this link.

CHAPTER 3

Ribo File Format

Ribo files contain ribosome profiling data in a compact form. Ribo format is built on top of HDF5 . A ribo file can hold data coming from multiple experiments.

3.1 Reference & Annotation

Ribo files work on the transcriptomic coordinates. Thus, sequencing data must be mapped to a transcriptome reference where each reference entry is a transcript. Thus, in this context we use the terms 'transcript' and 'reference' interchangeably. Typically, one representative transcript is picked for each gene for eukoryatic organisms.

3.1.1 Transcript Lengths

A ribo file requires a list of the transcript names and transcript lengths. This information is stored under *reference* data group.

Transcript names and lengths are provided in a tab separated file at the creation of a ribo file. The first column contains transcript names and the second column contains corresponding transcript lengths. All data will be stored and reported according to this given transcript order.

Transcript List Example:

TRANSCRIPT_1	1512
TRANSCRIPT_2	1387

All quantified entities (region counts, metagene, ribosome footprint coverage and transcript abundance) are stored per transcript.

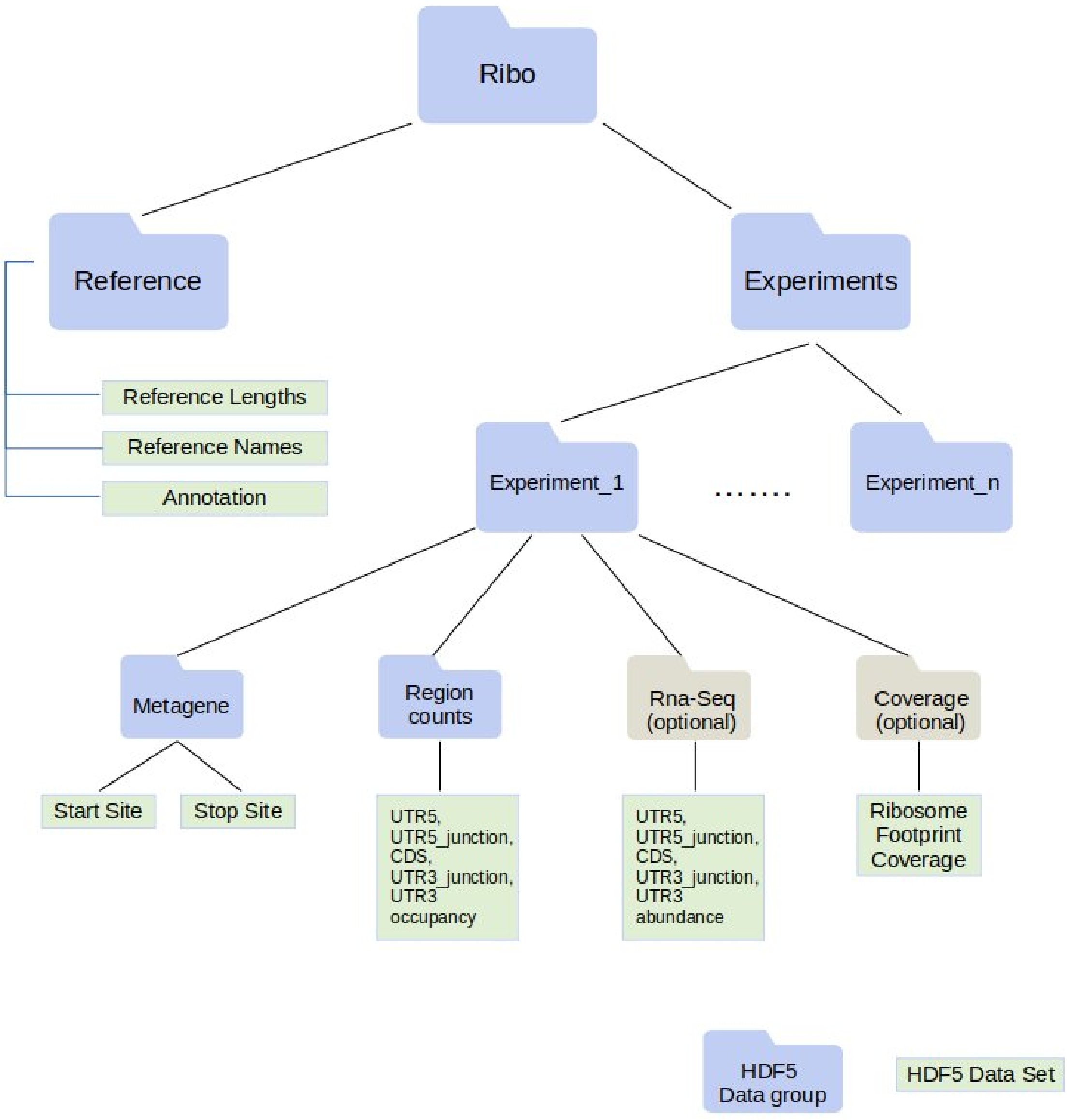

Fig. 1: Ribo File Organization

3.1.2 Annotation

Annotation of a transcript is the coordinates of the regions UTR5 (untranslated region on the five prime end), CDS (coding sequence) and UTR3 (untranslated region on the three prime end). Annotation information of each transcript is stored in a ribo file where region boundaries are stored in an array.

Annotation is given in bed file format where each region is defined in a separate line. Regions are defined by specifying their boundaries. Note that a bed file is 0-based and the start coordinate is included whereas the end coordinate is excluded.

Bed file entry to define a region is of the following form.

TRANSCRIPT_NAME	START	END	REGION_NAME	0	+

In the above entry, the last two columns are not used by RiboPy but they are still maintained for bed file compatibility.

Warning: RiboPy is not designed to work with transcripts with no CDS or UTR3 regions. In other words, all transcripts must have non-zero length CDS and UTR3 regions. Therefore the following annotation, in bed format, is **INVALID** because no UTR3 region is defined,

TRANSCRIPT_3	0	700	UTR5	0	+
TRANSCRIPT_3	100	1000	CDS	0	+

Warning: All nucleotide positions of a transcript must be annotated. Equivalently, there can NOT be gaps in the annotation of a transcript. Therefore, the following annotation is **INVALID** as the nucleotide positions 32, 33 and 34 don't belong to any region.

TRANSCRIPT_1	0	32	UTR5	0	+
TRANSCRIPT_1	35	920	CDS	0	+
TRANSCRIPT_1	920	1000	UTR3	0	+

Hint: It is possible to extract the annotation information from a ribo file. For example:

```
ribopy dump annotation sample.ribo
```

3.2 Read Length

Ribo format allows the user to store ribosome profiling data for a given range of lengths. If the minimum length is set to N and maximum length is set to M, then metagene, region counts, coverage and RNA-Seq (if any) data are stored for each length from N to M (N and M are included).

Note that length is the number of nucleotides of the ribosome protected RNA footprint.

3.3 Metagene

Metagene analysis is the study of ribosome footprint coverage around transcription start and stop sites.

First, a radius size, r, is fixed. Second, for each transcript, the coverage data from r nucleotides to the left and to the right side of the start site is obtained. Then, these coverage vectors are summed up to obtain an overall coverage around start / stop sites.

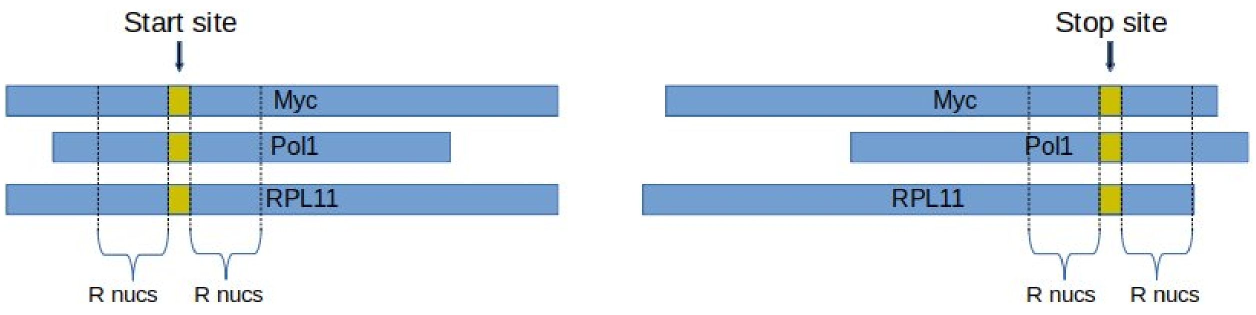

Fig. 2: Metagene for Start and Stop Sites

3.4 Region Counts

Start site is the position of the nucleotide on which ribosome starts translation. Stop site is the position of the first nucleotide on which ribosome terminates translation. Typically, a transcript can be split into three regions via start and stop sites: UTR5 (nucleotides on the 5' end of the start site), CDS (nucleotides between start and stop sites) and UTR3 (nucleotides to the right of stop site).

For the purpose of calculating ribosome footprints per region, we define two additional regions around start and stop sites. The definition of these additional regions also modify the conventional definition of UTR5, CDS and UTR3.

First we fix two integers **left span** = *l* and **left span** = *r*.

UTR5 Junction: This is the part of the transcript consisting of *l* nucleotides to the left of the start site and *r* nucleotides to the right of the start site, including the start site.

UTR3 Junction: This is the part of the transcript consisting of *l* nucleotides to the left of the stop site and *r* nucleotides to the right of the stop site, including the stop site.

Using UTR5 and UTR3 junctions, we re-define UTR5, CDS and UTR3 as follows.

UTR5: Nucleotides to the left of UTR5 Junction.

CDS: Nucleotides between UTR5 junction and UTR3 junction.

UTR3: Nucleotides to the right of UTR3 Junction.

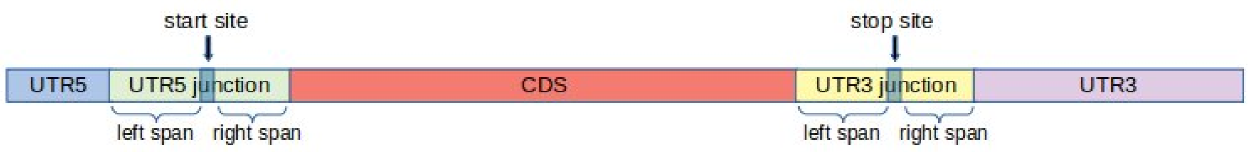

Fig. 3: Region Definitions

3.5 RNA-Seq

Storing RNA-Seq of an experiment is optional in ribo files.

?? More explanation is needed ??

3.6 Coverage

This is an optional part of ribo format.

Though metagene and region counts should be sufficient for most analysis, ribosome footprint coverage per nucleotide position for each length might be necessary in some cases. So, the user has the option to store this data in the ribo file. Storing coverage increases the size of the ribo file significantly,

3.7 Metadata

Metadata is a n optional part of ribo format. During or after the creation of a ribo file, users can provide metadata.

Metadata is provided in pairs of the form **key: value** in yaml format.

Metadata can be provided for two entities: ribo file and experiment.

3.7.1 Ribo File Metadata

This is stored at the root data group of the ribo file. Mapping or pipeline parameters or other metadata common to the experiments can be stored here.

3.7.2 Experiment Metadata

This part is intended to hold experiment specific metadata such as cell line, treatment, library preparation method etc.

3.8 Ribo File Attributes

Upon the creation of a ribo file, the following parameters are stored and they can NOT be changed thereafter. These attributes are created by RiboPy and they can NOT be specified or changed by the user.

- Metagene Radius
- Left Span & Right Span
- Minimum and Maximum Read Length
- Reference Name

Hint: These attributes can be viewed using the *info* command.

```
ribopy info sample.ribo
```

CHAPTER 4

Python API

A walk-through of Python API

4.1 Quick reference

4.1.1 Ribo Attributes

Essential Ribo class attributes

ribopy.Ribo(ribo_file[, alias, file_mode])	Ribo is an interface to ribo files.
ribopy.Ribo.experiments	List of experiments in the ribo object
ribopy.Ribo.minimum_length	Reads, used to generate the ribo file, are of at least minumum_length
ribopy.Ribo.maximum_length	Reads, used to generate the ribo file, are of at least minumum_length
ribopy.Ribo.metagene_radius	#nucleotides to the left and right of start / stop sites in metagene coverage.
ribopy.Ribo.left_span	#Nucleotides to the left of start / stop sites to define UTR5 & UTR3 junction regions
ribopy.Ribo.right_span	#Nucleotides to the right of start / stop sites to define UTR5 & UTR3 junction regions
ribopy.Ribo.format_version	Ribo file format version

4.1.2 Getter Functions

Methods for reading ribosome profiling data.

ribopy.Ribo.get_metagene(site_type[, ...])	Returns metagene data at start / stop site

Continued on next page

Table 2 – continued from previous page

ribopy.Ribo.get_region_counts(region_name[, ...])	Returns number of reads mapping to UTRs or CDS.
ribopy.Ribo.get_coverage(experiment[, ...])	Returns coverage at nucleotide resolution.
ribopy.Ribo.get_rnaseq([experiments])	Returns region counts coming from RNA-Seq data.

4.1.3 Plot Functions

Some essential plots for ribosome profiling analysis,

ribopy.Ribo.plot_metagene(site_type[, ...])	Generates coverage plots around start / stop sites.
ribopy.Ribo.plot_lengthdist(region_type, ...)	Generates distribution of the reads according to length
ribopy.Ribo.plot_region_counts(experiments)	Generates bar plots of region counts

4.2 Ribo

class ribopy.**Ribo**(*ribo_file*, *alias=None*, *file_mode='r'*)
Ribo is an interface to ribo files.

It provides access to ribo file attributes, metadata and ribosome profiling data in a ribo file.

Parameters

- **ribo_file** (*str, BytesIO*) – Path to a ribo file or a handle to a ribo file
- **file_mode** (*str , [choices: "r", "r+"]*) – h5py file mode. Ribo file must exist. So only read ("r") or read & write ("r+") modes are allowed. Be extremely careful with the r+ option. For most use cases, "r" (read-only) option should be sufficient. You are strongly discouraged to delete or modify existing attributes or data tables as this can corrupt the file.

experiments
List of experiments in the ribo object

format_version
Ribo file format version

get_coverage(*experiment*, *range_lower=0*, *range_upper=0*, *alias=False*)
Returns coverage at nucleotide resolution.

Note that RNA-Seq data is an optional entity of Ribo File.

get_length_dist(*region_name*, *experiments=[]*)
Returns the number of reads for each length for a given region

Parameters **region_name** (*str, list*) –

get_metadata(*experiment=None*)
Returns user defined metadata in dictionary form

Parameters **experiment** (*str (default = None)*) – If None, the metadata of the ribo file itself is returned. Otherwise, returns the metadata of the given experiment.

Returns result

Return type dict

get_metagene(*site_type*, *experiments=[]*, *sum_lengths=True*, *sum_references=True*, *range_lower=0*, *range_upper=0*, *alias=False*)

Returns metagene data at start / stop site

Metagene data is reported for a range of read lengths. Every read length in this range is included in the returned data frame. This range is provided as an interval by specifying range_lower and range_upper, both of which are included. If range is not specified, the minimum and maximum read lengths from the ribo file are taken as the range definition.

Parameters

- **site_type** (*str [choices: 'start' , 'stop']*) – Determines the site, around which metagene coverage is going to be reported.
- **sum_lengths** (*bool [default: True]*) – If True, metagene data is summed up across the given length range. If False, metagene data is reported for each length in the given range.
- **sum_references** (*bool [default: False]*) – If True, metagene data is summed across references (transcripts). If False, metagene data is reported for each transcript.
- **experiments** (*list, str*) – List of experiment(s). If empty, all experiments are included.

Returns **metagene_df** – This data frame contains coverage around start or stop site. The column labels are the relative positions, with respect to the start or stop site. The index of the dataframe depends on the two parameters: sum_lengths and sum_references. Whichever of these parameters are set to True, won't be in the index. Therefore, if sum_lengths is True, there won't be a length index in the metagene_df because the values are summed across the lengths.

Return type pandas.DataFrame

get_region_counts(*region_name*, *sum_lengths=True*, *sum_references=True*, *range_lower=0*, *range_upper=0*, *experiments=[]*, *alias=False*)

Returns number of reads mapping to UTRs or CDS.

Region counts are reported for a range of read lengths. Every read length in this range is included in the returned data frame. This range is provided as an interval by specifying range_lower and range_upper, both of which are included. If range is not specified, the minimum and maximum read lengths from the ribo file are taken as the range definition.

Parameters

- **region_name** (*str [choices: "UTR5", "UTR5_junction", "CDS", "UTR3_junction", "UTR3"]*) – For the definition of these regions, check the documentation at https://ribopy.readthedocs.io The region definitions are coming from the annotation in the ribo file.
- **sum_lengths** (*int [default: True]*) – If True, sum region counts across read lengths. If False, report region counts for each length.
- **sum_references** (*int [default: True]*) – If True, sum region counts across references (transcripts). If False, report region counts for each reference.
- **range_lower** (*int*) – Minimum read length to be included in the result. range_upper: int

  Maximum read length to be included in the result.

experiments: list, str [default: []] List of experiment(s). If empty, all experiments are included.

Returns **region_counts** – The index of the dataframe depends on the parameters sum_lengths and sum_references. More precisely, if sum_lengths is False, there will be an index for read length. If True, there won't be such an index as data is aggregated by read lengths. Similarly, if sum_references is False, there will be an index for reference (transcript) names. If True, there won't be such an index. The columns correspond to experiments.

Return type pd.DataFrame

`get_rnaseq`(*experiments=None*)

Returns region counts coming from RNA-Seq data.

Note that RNA-Seq data is an optional entity of Ribo File.

In contrast to region counts for ribosome profiling data, the resulting data frame do not have separate entries for read lengths. The columns of the data frame correspond to regions. The index of the Data Frame has two levels: experiment, transcript.

Parameters

- **`experiments`** (*`list, str`*) – List of experiment(s) whose RNA-Seq data is to be reported
- **`rnaseq_df`** (*`pd.DataFrame`*) – Data Frame containing RNA-Seq counts for each region.

`has_coverage`(*experiment*)

return has_coverage_data(self._handle, experiment_name)

Parameters **`experiment`** (*`str`*) – Named of the experiment whose coverage data is inquired.

Returns **result**

Return type bool

`has_metadata`(*experiment=None*)

Checks if Ribo has metadata

Parameters **`experiment`** (*`str (default = None)`*) – If None, the metadata of the ribo file itself is inquired. Otherwise, checks if the given experiment has metadata

Returns **result**

Return type bool

`has_rnaseq`(*experiment*)

Does experiment has RNA-Seq data?

Parameters **`experiment`** (*`str`*) – Name of the experiment

Returns **rnaseq_exists**

Return type bool

`info`

All Ribo attributes packed in a dictionary.

`left_span`

#Nucleotides to the left of start / stop sites to define UTR5 & UTR3 junction regions

`maximum_length`

Reads, used to generate the ribo file, are of at least minumum_length

metagene_radius
#nucleotides to the left and right of start / stop sites in metagene coverage.

minimum_length
Reads, used to generate the ribo file, are of at least minumum_length

plot_lengthdist(*region_type, experiments, title=", normalize=False, output_file=", colors=['blue', 'red', 'green', 'brown', 'orange', 'violet', 'black']*)
Generates distribution of the reads according to length

The x-axis is the read length and the y-axis is the number of reads mapping to the particular region.

Parameters

- **region_name** (*str [choices: "UTR5", "UTR5_junction", "CDS", "UTR3_junction", "UTR3"]*) – For the definition of these regions, check the documentation at https://ribopy.readthedocs.io The region definitions are coming from the annotation in the ribo file.
- **experiments** (*list, str*) – List of experiments to be plotted
- **title** (*str*) – Title of the plot
- **normalize** (*bool (default = False)*) – Normalize each experiment by total number of reads.
- **output_file** (*str (default = "")*) – If provided, the output will be saved in this path.
- **colors** (*(default: ["blue", "red", "green", "brown", "orange", "violet", "black"])*) – Colors of lines.

plot_metagene(*site_type, title=", experiments=[], range_lower=0, range_upper=0, normalize=False, output_file=", colors=['blue', 'red', 'green', 'brown', 'orange', 'violet', 'black']*)
Generates coverage plots around start / stop sites.

"metagene_radius" many nucleotides are taken on either side of the site type. The axis is the positions of the nucleotides relative to "site_type" and the y-axis is their frequency.

Setting normalize = True might be helpful when plotting more than one experiment. Values are normalized by total number of mapped reads in the experiment,

Parameters

- **site_type** (*str [choices: "start", "stop"]*) – Coverage plot is centered around the site type.
- **title** (*str, default = ""*) – Title of the plot
- **experiments** (*list, str*) – List of experiments to be plotted
- **range_lower** (*int*) – Minimum read length to be included in the metagene coverage data.
- **range_upper** (*int*) – Maximum read length to be included in the metagene coverage data.
- **normalize** (*bool (default = False)*) – Normalize metagene data by number of mapped reads in the experiment
- **output_file** (*str, (default = "")*) – If non-empty, the plot will be saved in the provided path.

- **colors** (*list, (default: ["blue", "red", "green", "brown", "orange", "violet", "black"])*) – Colors of lines.

plot_region_counts(*experiments*, *title=''*, *range_lower=0*, *range_upper=0*, *horizontal=True*, *output_file=''*)

Generates bar plots of region counts

The bar plot coming from the percentages of the counts of the regions: UTR5, CDS and UTR3

Parameters

- **experiments** (*list, str*) – List of experiments to be plotted
- **title** (*str*) – Title of the plot
- **range_lower** (*int*) – Minimum read length to be included in the bar plot data.
- **range_upper** (*int*) – Maximum read length to be included in the bar plot data.
- **horizontal** (*bool (default = True)*) – Generates bar plots horizontally. Especially for long experiment names, this is the preferred method.
- **output_file** (*str (default = "")*) – If provided, the output will be saved in this path.

print_info(*return_str=False*)

Prints Ribo file information in string format.

Parameters **return_str** (*bool*) – If True, retuns the info_str and does not print. if False, prints the info_str and returns none.

Returns **info_str** – Ribo File Summary String

Return type str

Notes

If this information is needed in a structured form, use the "info" attribute of the ribo object.

reference_name

Name of the reference (transacript assembly & annotation)

ribopy_version

Version of the ribopy used to creat the ribo file

right_span

#Nucleotides to the right of start / stop sites to define UTR5 & UTR3 junction regions

transcript_index

Transcript indices in dictionary form.

transcript_name -> transcript_length

The transcript index coming from the order of Transcripts in the ribo file,

transcript_lengths

Transcript lengths in dictionary form.

transcript_name -> transcript_length

transcript_names

Transcripts Names in the ribo reference

`transcript_offsets`

Transcript offsets

This gives us the initial position of the transcript in determining coverage. If the transcripts are linearly lined up according to the order in the ribo file, then this offset is the position of the first nucleotide of the transcript.

CHAPTER 5

Python CLI

A walk-through of Python CLI

5.1 CLI Reference

```
CLI Reference [OPTIONS] COMMAND [ARGS]...
```

Options

--debug, --no-debug

5.1.1 create

Creates a ribo file from a given reference, annotation and alignment file.

The resulting ribo file contains a single experiment whose name is provided in "–name". If "–name" is not provided, the experiment name will be extracted from the ribo file path after stripping the file extension.

RiboPy works on transcriptomic coordinates. In other words, all coordinates are relative to transcripts where the first nucleotide of the transcript is always zero. Each entry in the alignment reference must come from a single transcript. Ribo-Seq data aligned against genomic coordinates is NOT usable with RiboPy.

A note on transcript <-> reference: Sequencing reads are mapped against a reference to generate alignment files in bed or bam format. Each entry of this reference is coming from a transcript. Therefore, in this context, refrence and transcript correspond to the same entity. Thus, the names 'reference' and 'transcript' are used interchangebly. For example 'reference names' and 'transcript names' refer to the same list of names.

Reference Lengths File: Reference lengths must be provided in the 'lengths' option. This must be a tab separated file where transcript names are in the first coulmn and the transcript lengths are in the seocond column.

Example:

TRANSCRIPT_1	1512
TRANSCRIPT_2	1387

Annotation Examples: Below are some valid and invalid annotation file examples. ribopy will report an error and fail to generate a ribo file if an invalid annotation file is given.

VALID ANNOTATION: (Assuming length of TRANSCRIPT_1 is 1252)

TRANSCRIPT_1	0	21	UTR5	0	+
TRANSCRIPT_1	21	1041	CDS	0	+
TRANSCRIPT_1	1041	1252	UTR3	0	+

INVALID ANNOTATION: (Assuming length of TRANSCRIPT_2 is 1000. Nucleotide positions 32,33 and 34 are not annotated)

TRANSCRIPT_1	0	32	UTR5	0	+
TRANSCRIPT_1	35	920	CDS	0	+
TRANSCRIPT_1	920	1000	UTR3	0	+

INVALID ANNOTATION: (Assuming length of TRANSCRIPT_3 is 1200) (There is no CDS region.)

TRANSCRIPT_3	0	50	UTR5	0	+
TRANSCRIPT_3	50	1200	UTR3	0	+

INVALID ANNOTATION: (Assuming length of TRANSCRIPT_4 is 1000) (There is no UTR3 region.)

TRANSCRIPT_3	0	700	UTR5	0	+
TRANSCRIPT_3	100	1000	CDS	0	+

Region Counts & Left / Right Span: For each transcript, the number of reads aligning to each region (UTR5, CDS, and UTR3) are stored. For this quantification, we exclude nucleotides in some proximity of start and stop sites.

This proximity is defined by taking 'leftspan' many nucleotides to the left of start / stop sites and 'rightspan' many nucleotides to the right of start / stop sites. The regions around start and stop sites are called UTR5_junction and UTR3_junction respectively.

UTR5_juction: Nucleotides 'around' start site. This region is between UTR5 and CDS.

UTR3_juction: Nucleotides 'around' stop site. This region is between CDS and UTR3.

Note that 'around' is precisely defined by 'leftspan' and 'rightspan' arguments.

Read Length Range: Quantified Ribo-Seq data (metagene coverage, region counts, transcript ocoverage (if any)) is stored for each read length for a given range. This range is defined by –lengthmin and –lengthmax. Both values are inclusive. So, for example, for –lengthmin 19 –lengthmax 21, region counts are computed and stored for RNA fragments of length 19,20 and 21 separetely. For human Ribo-Seq data, a range from 15 to 35 can be sufficient for conventional experiments.

Metadata: Metadata, either for the ribo file or for any experiment, is an optional argument. Metadata must be provided in yaml format.

Users can provide metadata for

i) ribo file

–ribometa

ii) experiment

–expmeta

Metadata is provided in pairs of the form 'label: value'

Example Metadata:

cell-line:	HEK
enzyme:	RNASEI

Examples: Below are some ribo file generation examples in different settings.

1) Some command line exaples to create ribo files. Create a file named WT.ribo in the current directory. Provide alignment data in a zipped bed file. Do NOT store coverage data.

```
ribopy create --name WT \
   --alignmentfile WT.bed.gz \
   --reference appris_human_v1 \
   --lengths appris_len.tsv \
   --annotation appris_regions.bed \
   --radius 50 \
   -l 35 -r 15 \
   --lengthmin 15 --lengthmax 35 \
   --nocoverage \
   WT.ribo
```

2) Create a file named Treatment_1.ribo in the current directory. Provide alignment data in a zipped bam file. Store coverage data.

```
ribopy create --name Treatment_1 \
    --alignmentfile Treatment_1.bam \
    --format bam \
    --reference appris_human_v1 \
    --lengths appris_len.tsv \
    --annotation appris_regions.bed \
    --radius 50 \
    -l 35 -r 15 \
    --lengthmin 15 --lengthmax 35 \
   Treatment_1.ribo
```

3) Create a file named WT_2.ribo in the current directory. Provide alignment data in a zipped bed file. Do NOT store coverage data. Read metadata of this experiment from WT_2_meta.yaml

```
ribopy create --name WT_2 \
    --alignmentfile WT_2.bed.gz \
    --reference appris_human_v1 \
    --lengths appris_len.tsv \
    --annotation appris_regions.bed \
    --radius 50 \
    -l 35 -r 15 \
    --lengthmin 15 --lengthmax 35 \
    --nocoverage \
    --expmeta WT_2_meta.yaml \
    WT_2.ribo
```

```
CLI Reference create [OPTIONS] RIBOFILE
```

Options

-a, --alignmentfile `<alignmentfile>`
Aligned Ribo-Seq Data File If no file is provided, it is read from standard input.Alignment file can be in bed or bam format.Bed files with '.gz' or '.gzip'extension are assumed to be gzipped.So they will be automatiocally unzipped.If no aligment file is given,it is read from the standard input.Thus, the user can pipe the output of some processto ribopy to produce ribo files.

-f, --alignmentformat `<alignmentformat>`
Alignment File Format [default: bed]

Options bed|bam

--name `<name>`
Experiment name If name is not provided, experiment name is extracted from the ribo file path. If –name parameter is omitted, experiment name is determined using the ribo file path. For example if the ribo file path is /home/user/data/WT.ribo, then, experiment name is set to 'WT'. Experiment name can only contain alphanumeric characters, '_', '-' and '.'.

--reference `<reference>`
Reference name It is good practice to give a unique name to a particular transcriptome and annotation pair and use it consistently.For convenience, the user has the freedomto choose a name for the transcript reference and the annotation, used to create a ribo file,If used consistently, one can tell whether two ribo files are coming from the same annotation or not.Note that ribo files having different reference names can not be merged. [required]

--lengthsfile, --lengths `<lengthsfile>`
A tab-separated file containing ref. name and ref. lengths [required]

--annotationfile, --annotation `<annotationfile>`
A bed file defining UTR5, CDS and UTR3 regions.Each transcript must be annotated. More explicitly, the coordinates of the regions UTR5, CDS and UTR3 must be provided for each transcript. This annotation is provided in a BED file. The annotation can not contain gaps. All nucleotides of a transcript must belong to a region. If a transcript does not have a CDS or UTR3 region, it must be excluded from the lengths file and annotation. Note that bed files are 0-based, start position is included and the end position is excluded. The order and transcript names in the lengths file must match the order in the annotation file. Annotation data is kept in ribo files. It is possible to extract the annotation from a ribo file using the dump command. [required]

--metageneradius, --radius `<metageneradius>`
Number of nucleotides on either side of start / stop sites for metagene analysis.

Aligning transcripts by their start / stop sitesand aggregating the coverage gives us metagene data.The number of nucleotides to be taken on either side ofstart / stop site is given by the metagene radius.

-l, --leftspan `<leftspan>`
Number of nucleotides to the left of start / stop sites for defining UTR5 / UTR3 junction regions. [default: 35]

-r, --rightspan `<rightspan>`
Number of nucleotides to the right of start / stop sites for defining UTR5 / UTR3 junction regions. [default: 15]

--lengthmin, --min `<lengthmin>`
Minimum read length to be counted [default: 15]

--lengthmax, --max `<lengthmax>`
Maximum read length to be counted [default: 35]

--ribometa `<ribometa>`
Metadata file, for the ribo file, in yaml format.

--expmeta `<expmeta>`
Metadata file, for the experiment, in yaml format.

--nocoverage
Do not store coverage. By default, coverage IS stored at nucleotide resolution, of each transcript, for each read length. Turning this flag on decreases size of ribo file at the cost of coverage data. From the alignment data, five prime end of the reads, mapping to each nucleotide position, of each transcript is computed for each read length. We call this coverage. Coverage is used for metagene analysis and region counts. By default, ribopy stores coverage data in the ribo file. Keeping coverage increases the ribo file size considerably. Users can choose NOT to keep coverage by setting this flag.

-n, --nprocess `<nprocess>`
Number of cores to be used. [default: 1]

Arguments

RIBOFILE
Required argument

5.1.2 dump

Dump selected parts of ribo files to particular formats

```
CLI Reference dump [OPTIONS] COMMAND [ARGS]...
```

annotation

Gives UTR5, CDS and UTR3 annotation in bed format.

Annotation is written to standard output if no output file is given.

Examples:

1) Store the annotation in a bed file

```
ribopy dump annotation -o regions.bed sample.ribo
```

2) Print annotation to standard output

```
ribopy dump annotation sample.ribo
```

```
CLI Reference dump annotation [OPTIONS] RIBO
```

Options

-o, --out `<out>`
Output file in bed format

Arguments

RIBO
Required argument

coverage

Prints the coverage of each transcript.

This command prints the coverage of each transcript, at nucleotide resolution, for a given read length range. For a single read length, set 'lowerlength' equal to 'upperlength'. The values for the given range are aggregated by summation.

Append ".gz" to the output file name to have the output in zipped form.

File format: options: bg, tsv

bg: Bedgraph The columns of the bedgraph file are of the form transcript_name location_start location_end coverage_value Only the nonzero values are reported in the bedgraph file. Bedgraph is 0-based and location_start is inclusive and location_end is exclusive.

See https://genome.ucsc.edu/goldenPath/help/bedgraph.html for a detailed description of bedgraph file format.

tsv: Tab separated File In this format, coverage values for all nucleotide positions, regardless of their value (0 or not) are reported.

The columns of the file are of the form transcript_name comma_separated_coverage

Examples:

1) Get coverage data, of WT, for read length 21 in zipped bedgraph format

```
ribopy --lowerlength 21 --upperlength 21 \
    -o coverage.bg.gz \
    --fromat bg sample.ribo WT
```

2) Get coverage data, of treatment_1, for range 26 to 30 in tsv format

```
ribopy --lowerlength 26 --upperlength 30
      -o coverage.tsv
      --format tsv
      sample.ribo treatment_1
```

```
CLI Reference dump coverage [OPTIONS] RIBO EXPERIMENT
```

Options

-o, --out <out>
Output file in csv format

--lowerlength <lowerlength>
Minimum read length to take

--upperlength <upperlength>
Maximum read length to take

--format <format>
Output file format

Options bg|tsv

Arguments

RIBO
Required argument

EXPERIMENT
Required argument

metagene

Dumps metagene data to a csv file or standard output.

Metagene data is obtained from coverage around start / stop site in a pre-defined radius.

Examples: Below are some examples for different scenarios.

1) Get metagene data around START site for the experiment WT. Report results for read lengths from 28 to 32. Agrregate data by summing accros read lengths. Also, values are summed across transcripts. Save the results in start.csv.

```
ribopy dump metagene \
   --experiment WT \
   --site start \
   --out start.csv \
   --lowerlength 28 --upperlength 32 \
   --sumlengths \
   sample.ribo
```

2) Get metagene data around STOP site for the experiment Treatment_1. Report results for read length 31. Values are summed across transcripts. Print the results on the standard output.

```
ribopy dump metagene \
   --experiment WT \
   --site stop \
   --lowerlength 31 --upperlength 31 \
   sample.ribo
```

3) Get metagene data around START site for the experiment WT. Report results for read lengths from 30 to 32. Report results for each read length. Also, values are summed across transcripts. Save the results in start.csv.

```
ribopy dump metagene \
   --experiment WT \
   --site start \
   --out start.csv \
   --lowerlength 30 --upperlength 32 \
   sample.ribo
```

```
CLI Reference dump metagene [OPTIONS] RIBO
```

Options

-s, --site `<site>`
Site type. [required]

Options start|stop

-o, --out <out>
Output file in bed format

-e, --experiment <experiment>
Name of the experiment

-l, --lowerlength <lowerlength>
Minimum read length to be taken

-u, --upperlength <upperlength>
Maximum read length to be taken

--sumlengths
Sum accross lengths

--nosumtrans
Do NOT aggregate values accross transcripts

Arguments

RIBO
Required argument

reference-lengths

Gives transcript names and their lengths.

Annotation is written to standard output if no output file is given.

The first column corresponds to transcript names and the second column corresponds to transcript lengths.

Examples: Print the output to the terminal

```
ribopy dump reference-lengths sample.ribo
```

Save the output, in gzipped form, in lengths.csv.gz and use , to separate columns

```
ribopy dump reference-lengths -o lengths.csv.gz --sep "," sample.ribo
```

```
CLI Reference dump reference-lengths [OPTIONS] RIBO
```

Options

-o, --out <out>
Output file

-s, --sep <sep>
Column separator, default is t (tab)

Arguments

RIBO
Required argument

region-counts

Dumps a given region in csv format to a file or standard output.

Examples: Below are some examples for different scenarios.

1) Get number of reads mapping to the coding sequence (CDS) for the experiment WT. Report results for read lengths from 28 to 32. Agrregate data by summing accros read lengths. Also, values are summed across transcripts. Save the results in cds.csv.

```
ribopy dump region-counts \
   --experiment WT \
   --region CDS \
   --out cds.csv \
   --lowerlength 28 --upperlength 32 \
   --sumlengths \
   --sumtrans \
   sample.ribo
```

2) Get number of reads mapping to UTR3 for the experiment Treatment. Report results for read lengths from 30 to 32. Agrregate data by summing accros read lengths. CDS occupancy is reported for each transcript. Save the results in cds.csv.

```
ribopy dump region-counts \
   --experiment Treatment \
   --region UTR3 \
   --out cds.csv \
   --lowerlength 30 --upperlength 32 \
   --sumlengths \
   sample.ribo
```

```
CLI Reference dump region-counts [OPTIONS] RIBO
```

Options

-r, --region <region>
Site type. [required]

Options UTR5|UTR5_junction|CDS|UTR3|UTR3_junction

-o, --out <out>
Output file in csv format

-e, --experiment <experiment>
Name of the experiment

--lowerlength <lowerlength>
Minimum read length to be taken

--upperlength <upperlength>
Maximum read length to be taken

--sumlengths
Sum accross lengths

--sumtrans
Sum accross transcripts

Arguments

RIBO
Required argument

5.1.3 info

Displays a summary information about the given ribo file

```
CLI Reference info [OPTIONS] RIBOFILE
```

Arguments

RIBOFILE
Required argument

5.1.4 merge

Merges a set of given ribo files into one ribo file.

The input ribo files are merged into a new ribo file. The resulting ribo file has the union of the experiments of the input files

The ribo files to be merged must be compatible: They must have the same

1) Reference Name
2) Transcript Names & Transcript Lengths
3) Annotation
4) Ribo file parameters:
    a) Metagene Radius
    b) Left Span & Right Span
    c) Min & Max Read Length
5) They can not have overlapping experiment names

This option is especially useful for Ribo-Seq Data processed together and needs to be analyzed together.

```
CLI Reference merge [OPTIONS] OUT_RIBO_PATH [IN_RIBO_PATHS]...
```

Arguments

OUT_RIBO_PATH
Required argument

IN_RIBO_PATHS
Optional argument(s)

5.1.5 metadata

Display, set or delete user-defined metadata

If no name is given, the metadata of the ribo file is set, displayed or deleted.

```
CLI Reference metadata [OPTIONS] COMMAND [ARGS]...
```

delete

Deletes user-defined metadata of the ribo file or experiment

If no name is provided, metadata of the ribo file is deleted.

If name is given, metadata of the corresponding experiment is deleted.

```
CLI Reference metadata delete [OPTIONS] RIBO
```

Options

--name <name>
experiment name

--force
Set metadata without prompting user.

Arguments

RIBO
Required argument

get

Displays user-defined metadata of the ribo file or experiment

If no name is provided, metadata of the ribo file is displayed.

If name is given, metadata of the corresponding experiment is displayed.

```
CLI Reference metadata get [OPTIONS] RIBO
```

Options

--name <name>
experiment name

Arguments

RIBO
Required argument

set

Stores the metadata in the meta file.

If no name is given, the metadata of the ribo file is set.

If name is provided, the metadata of the corresponding experiment is set.

The metadata must be in yaml format.

Example Ribo Metadata File Contents:

pipeline_name:	RiboFlow
pipeline_version:	v1.0.2
project:	elongation blocker

Example Library Metadata File Contents:

Cell_Line:	Human ESC
Treatment:	Drug A
Enzyme:	RNASEI

```
CLI Reference metadata set [OPTIONS] RIBO
```

Options

--name <name>
experiment name

--meta <meta>
Metadata File [required]

--force
Set metadata without prompting user.

Arguments

RIBO
Required argument

5.1.6 plot

Generate some basic plots for ribo files.

```
CLI Reference plot [OPTIONS] COMMAND [ARGS]...
```

lengthdist

Plots the distribution of the ribosome footprint lengths.

The x-axis is the length of the protected ribosome footprints. The y-axis is the raw or normalized frequecies.

At most 7 experiments can be provided for a single plot.

Pdf and png output formats are supported. If "dump" option is provided, the data is written to the provided file path.

If the frequencies are normalized using the "–normalize" option, the y-axis becomes the percentages of the frequencies.

Examples: 1) Plot CDS length distribution of exp_1 and exp_2 and normalize the frequencies.

```
ribopy plot lengthdist  \
   -o multiple_dist.pdf \
   -r CDS --normalize \
   project.ribo exp_1 exp_2
```

2)Plot only main_exp and write the data to out.csv.

```
ribopy plot lengthdist \
    -d out.csv \
    -o main_exp.pdf \
    -r CDS \
    project.ribo  main_exp
```

```
CLI Reference plot lengthdist [OPTIONS] RIBO [EXPERIMENTS]...
```

Options

-r, --region <region>
Region type. [required]

Options UTR5|UTR5_junction|CDS|UTR3_junction|UTR3

-o, --out <out>
Output file in bed format [required]

-t, --title <title>
Plot title.

--normalize
Normalize by total metagene site coverage

-d, --dump <dump>
Dump the data to csv file

Arguments

RIBO
Required argument

EXPERIMENTS
Optional argument(s)

metagene

Generates metagene plots.

The x-axis is the relative nucleotide positions and the start /stop site is at the center (origin at 0). The y-axis is the raw or normalized coverage.

At most 7 experiments can be provided for a single plot.

For a given start or stop site, the coverage around the site of interest is plotted.

The supported output extensions are pdf and png.

If a length range is not provided, the minimum and the maximum ranges from the ribo file are read and used as the range. The values in the length range are aggregated to generate the metagene plot.

Examples:

1) Get coverage data, of WT, for read length 21 in zipped bedgraph format

```
ribopy plot metagene -s start -o hela_1.pdf project.ribo HeLa_1
```

2) Two experiments in one plot, stop site, with normalization

```
ribopy plot metagene -s start --normalize -o hela_1_2.pdf project.ribo␣
↪HeLa_1 HeLa_2
```

3) Plot start site usiong footprints of length 20,21,22 Dump the data to out.csv file.

```
ribopy plot metagene -s start \
  --lowerlength 20 upperlength 22 \
  -o hela_1.pdf \
  -d out.csv \
  project.ribo HeLa_1
```

```
CLI Reference plot metagene [OPTIONS] RIBO [EXPERIMENTS]...
```

Options

-s, --site <site>
Site type. [required]

Options start|stop

-o, --out <out>
Output file. [required]

--lowerlength <lowerlength>
Lower read length

--upperlength <upperlength>
Upper read length

-t, --title <title>
Plot title.

--normalize
Normalize by total metagene site coverage

-d, --dump <dump>
Dump the data to csv file

Arguments

RIBO
Required argument

EXPERIMENTS
Optional argument(s)

regioncounts

Generates barplots of the percentages of the UTR5, CDS and UTR3 counts.

The raw counts are saved to a csv file if "–dump" option is provided.

If a length range is not provided, then the range is determined using the mininum and the maximum read lengths in the ribo file.

Examples: 1) Plot the region counts for the experiment names sample for lengths from 29 to 31

```
ribopy plot regioncounts --lowerlength 29 --upperlength 31 \
    --out sample.region_counts.pdf test.ribo sample
```

2) Plot region counts for the two experiments WT and DrugA for all lengths combined

```
ribopy plot regioncounts --out wt_anddrug.pdf other.ribo WT DrugA
```

```
CLI Reference plot regioncounts [OPTIONS] RIBO [EXPERIMENTS]...
```

Options

-o, --out <out>
Output file [required]

--lowerlength <lowerlength>
Lower read length

--upperlength <upperlength>
Upper read length

-t, --title <title>
Plot title.

--horizontal
Draw bars horizontally.

-d, --dump <dump>
Dump the data to csv file

Arguments

RIBO
Required argument

EXPERIMENTS
Optional argument(s)

5.1.7 rnaseq

Display, set or delete RNA-Seq data

```
CLI Reference rnaseq [OPTIONS] COMMAND [ARGS]...
```

delete

Delete RNA-Seq data of a particular experiment

Example

——-

ribopy rnaseq delete –name WT test.ribo

```
CLI Reference rnaseq delete [OPTIONS] RIBO
```

Options

--name `<name>`
experiment name [required]

--force
Delete RNA-Seq without prompting user.

Arguments

RIBO
Required argument

get

Get transcript expression data of a given experiment

If no output parameter is provided, the results are printed to standard output.

Transcript expression is reported in two columns where the first column corresponds to transcript names and the second column corresponds to transcript expression.

Examples

———

Save the transcript expression in a tab separated file in gzipped form.

1) ribopy rnaseq get –name WT –out transcript_exp.tsv.gz test.ribo

Print the transcript expression on the screen

2) ribopy rnaseq get –name WT test.ribo

```
CLI Reference rnaseq get [OPTIONS] RIBO
```

Options

--name <name>
experiment name

--out <out>
Output File

--sep <sep>
Column Separator: Default is tab

Arguments

RIBO
Required argument

set

Store the transcript expression data of an experiment

?? MISSING DOCUMENTATION ??

```
CLI Reference rnaseq set [OPTIONS] RIBO
```

Options

-n, --name <name>
experiment name [required]

-a, --alignment <alignment>
RNASeq alignments in bed or bam format.

-c, --counts <counts>
Transcript Expression File

-f, --format <format>
RNA-Seq alignment format

Options bed|bam

--sep <sep>
Column Separator for counts file [default:]

--force
Set RNA-Seq without prompting user.

Arguments

RIBO
Required argument

CHAPTER 6

R API

A walkthrough of RiboR is in this link.

CHAPTER 7

Release notes

7.1 Latest

7.1.1 Enhancements

- Improved documentation

7.2 v0.0.0

Date: 09/19/2019

- Working initial version.

CHAPTER 8

Index

- genindex

Index