pyroe

Release 0.8.1

Dongze He, Rob Patro

Jun 23, 2023

CONTENTS:

CHAPTER

ONE

WHAT IS PYROE?

The pyroe package provides useful functions for analyzing single-cell or single-nucleus RNA-sequencing data using *alevin-fry*.

1.1 Installing pyroe

The `pyroe` package can be accessed from its github repository, installed via pip. To install the `pyroe` package via `pip` use the command:

```
pip install pyroe
```

Alternatively, `pyroe` can be installed via `bioconda`, which will automatically install the variant of the package including `load_fry`, and will also install `bedtools` to enable faster construction of the expanded references. This installation can be performed with the following shell command:

```
conda install pyroe -c bioconda
```

1.2 Preparing an expanded transcriptome reference for quantification with alevin-fry

The USA mode in alevin-fry requires an expanded index reference, in which sequences represent spliced and unspliced transcripts. Pyroe provides CLI programs and python functions to build the pre-defined expanded references, the spliced + intronic (*splici*) reference, which includes the spliced transcripts plus the (merged and collapsed) intronic sequences of each gene and the spliced + unspliced (*spliceu*) reference, which consists of the spliced transcripts plus the unspliced transcript (genes' entire genomic interval) of each gene. The `make_splici_txome()` and `make_spliceu_txome()` python functions are designed to make the *splici* and *spliceu* reference by taking a genome FASTA file and a gene annotation GTF file as the input. Furthermore, the

1.2.1 Preparing a *spliced+intronic* transcriptome reference

The *splici* index reference of a given species consists of the transcriptome of the species, i.e., the spliced transcripts and the intronic sequences of the species. Within a gene, if the flanked intronic sequences overlap with each other, the overlapped intronic sequences will be collapsed as a single intronic sequence to make sure each base will appear only once in the intronic sequences of each gene.

To prepare a *splici* reference using pyroe, in addition to a genome FASTA file and a gene annotation GTF file, you also need to specify the read length argument `read_length` of the experiment you are working on. Besides, you can provide a flank trimming length `flank_trim_length`. Then, a final flank length will be computed as the difference between the `read_length` and flank trimming length and will be attached to the ends of each intron to absorb the intron-exon junctional reads. Details about *splici* can be found in the supplementary section S2 of the alevin-fry paper.

Following is an example of calling `pyroe` in the command line to make the *splici* index reference. The final flank length is calculated as the difference between the read length and the flank_trim_length, i.e., 5-2=3. This program allows you to add extra spliced and unspliced sequences to the *splici* index reference, which will be useful when some unannotated sequences, such as mitochondrial genes, are important for your experiment. **Note** : To make *pyroe* work more quickly, it is recommended to have the latest version of bedtools (Aaron R. Quinlan and Ira M. Hall, 2010) installed.

```
pyroe make-spliced+intronic \
extdata/small_example_genome.fa \
extdata/small_example.gtf 5 splici_txome \
--flank-trim-length 2 \
--filename-prefix splici \
--dedup-seqs
```

The *pyroe make-spliced+intronic* program writes three files to your specified output directory *splici_txome*. They are

- A FASTA file that stores the extracted splici sequences.
- A three-column transcript-name-to-gene-name file that stores the name of each reference sequence in the splici index reference, their corresponding gene name, and the splicing status (*S* for spliced and *U* for unspliced) of those transcripts.
- A two-column TSV file that maps gene ids (used as the keys in eventual alevin-fry output) to gene names. This can later be used with the `pyroe convert` command line program to convert gene ids to gene names in the count matrix.

Full usage

```
usage: pyroe make-spliced+intronic [-h] [--filename-prefix FILENAME_PREFIX]
                                  [--flank-trim-length FLANK_TRIM_LENGTH]
                                  [--extra-spliced EXTRA_SPLICED]
                                  [--extra-unspliced EXTRA_UNSPLICED]
                                  [--bt-path BT_PATH] [--no-bt]
                                  [--dedup-seqs] [--no-flanking-merge]
                                  [--write-clean-gtf]
                                  genome-path gtf-path read-length output-dir


positional arguments:
  genome-path           The path to a genome fasta file.
  gtf-path              The path to a gtf file.
  read-length           The read length of the single-cell experiment being
                        processed (determines flank size).
  output-dir            The output directory where splici reference files will
```

(continues on next page)

(continued from previous page)

```
                        be written.

options:
  -h, --help            show this help message and exit
  --filename-prefix FILENAME_PREFIX
                        The file name prefix of the generated output files.
  --flank-trim-length FLANK_TRIM_LENGTH
                        Determines the amount subtracted from the read length
                        to get the flank length.
  --extra-spliced EXTRA_SPLICED
                        The path to an extra spliced sequence fasta file.
  --extra-unspliced EXTRA_UNSPLICED
                        The path to an extra unspliced sequence fasta file.
  --bt-path BT_PATH     The path to beadtools v2.30.0 or greater.
  --no-bt               A flag indicates whether bedtools will be used for
                        generating splici reference files.
  --dedup-seqs          A flag indicates whether identical sequences will be
                        deduplicated.
  --no-flanking-merge   A flag indicates whether flank lengths will be
                        considered when merging introns.
  --write-clean-gtf     A flag indicates whether a clean gtf will be written
                        if encountered invalid records.
```

The pyroe `make-spliced+intronic` command line program calls the `make_splici_txome()` python function under the hood. One can also directly call this function from a python instance to build a *splici* index. Here we provide helping messages of the `make_splici_txome()` python function.

```
Construct the splici (spliced + introns) transcriptome for alevin-fry.

Required Parameters
genome_path : str
    The path to a genome fasta file.

gtf_path : str
    The path to a gtf file.

read_length : int
    The read length of the single-cell experiment being processed.

output_dir : str
    The output directory, where the splici reference files will be written.

Optional Parameters
flank_trim_length : int (default: 5)
    The flank trimming length. The final flank length is obtained by subtracting the␣
↪flank_trim_length from the read_length.

filename_prefix : str (default: splici)
    The file name prefix of the generated output files. The derived flank length will be␣
↪automatically appended to the provided prefix.

extra_spliced : str
```

(continues on next page)

(continued from previous page)

```
    A path to a fasta file. The records in this fasta file will be regarded as spliced␣
↪transcripts.

extra_unspliced : str
    The path to a fasta file. The records in this fasta file will be regarded as introns.

dedup_seqs : bool (default: False)
    If True, the repeated sequences in the splici reference will be deduplicated.

no_bt : bool (default: False)
    If true, biopython, instead of bedtools, will be used for generating splici␣
↪reference files.

bt_path : str
    The path to bedtools v2.30.0 or greater if it is not in the environment PATH.

no_flanking_merge : bool (default: False)
    If true, overlapping introns caused by the added flanking length will not be merged.

Returns
Nothing will be returned. The splici reference files will be written to disk.
```

1.2.2 Preparing a *spliced+unspliced* transcriptome reference

Recently, He et al., 2023 introduced the spliced + unspliced (*spliceu*) index in alevin-fry. This requires the spliced + unspliced transcriptome reference, where the unspliced transcripts of each gene represent the entire genomic interval of that gene. Details about the *spliceu* can be found in the preprint. To make the spliceu reference using pyroe, one can call the `make_spliceu_txome()` python function or `pyroe make-spliced+unspliced` or its alias `pyroe make-spliceu` from the command line. The following example shows the shell command of building a spliceu reference from a given reference set in the directory `spliceu_txome`.

```
pyroe make-spliced+unspliced \
extdata/small_example_genome.fa \
extdata/small_example.gtf \
spliceu_txome \
--filename-prefix spliceu
```

Full usage

```
usage: pyroe make-spliced+unspliced [-h] [--filename-prefix FILENAME_PREFIX]
                                    [--extra-spliced EXTRA_SPLICED] [--extra-unspliced␣
↪EXTRA_UNSPLICED]
                                    [--bt-path BT_PATH] [--no-bt] [--dedup-seqs]
                                    genome-path gtf-path output-dir

positional arguments:
  genome-path           The path to a genome fasta file.
  gtf-path              The path to a gtf file.
  output-dir            The output directory where Spliceu reference files will be␣
```

(continues on next page)

(continued from previous page)

```
↪written.

options:
  -h, --help            show this help message and exit
  --filename-prefix FILENAME_PREFIX
                        The file name prefix of the generated output files.
  --extra-spliced EXTRA_SPLICED
                        The path to an extra spliced sequence fasta file.
  --extra-unspliced EXTRA_UNSPLICED
                        The path to an extra unspliced sequence fasta file.
  --bt-path BT_PATH     The path to bedtools v2.30.0 or greater.
  --no-bt               A flag indicates whether bedtools will be used for generating␣
↪Spliceu reference
                        files.
  --dedup-seqs          A flag indicates whether identical sequences will be␣
↪deduplicated.
```

The `pyroe make-spliced+unspliced` command line program calls the `make_spliceu_txome()` python function under the hood. One can also directly call this function from a python instance to build a *spliceu* index. Here we provide helping messages of the `make_spliceu_txome()` python function.

```
Construct the spliceu (spliced + unspliced) transcriptome for alevin-fry.

Required Parameters
genome_path : str
    The path to a genome fasta file.

gtf_path : str
    The path to a gtf file.

output_dir : str
    The output directory, where the spliceu reference files will be written.

Optional Parameters
filename_prefix : str (default: spliceu)
    The file name prefix of the generated output files. The derived flank length will be␣
↪automatically appended to the provided prefix.

extra_spliced : str
    A path to a fasta file. The records in this fasta file will be regarded as spliced␣
↪transcripts.

extra_unspliced : str
    The path to a fasta file. The records in this fasta file will be regarded as introns.

dedup_seqs : bool (default: False)
    If True, the repeated sequences in the spliceu reference will be deduplicated.

no_bt : bool (default: False)
    If true, biopython, instead of bedtools, will be used for generating spliceu␣
↪reference files.

```

(continues on next page)

(continued from previous page)

```
bt_path : str
    The path to bedtools v2.30.0 or greater if it is not in the environment PATH.

Returns
Nothing will be returned. The spliceu reference files will be written to disk.

Notes
The input GTF file will be processed before extracting unspliced sequences. If pyroe␣
↪finds invalid records, a clean_gtf.gtf file will be generated in the specified output␣
↪directory. **Note** : The features extracted in the spliced + unspliced transcriptome␣
↪will not necessarily be those present in the clean_gtf.gtf file - as this command will␣
↪prefer the input in the user-provided file wherever possible. More specifically:
If the required metadata fields contain missing values, pyroe will impute them if␣
↪possible, or return an error if not.
**Pyroe will always extract unspliced sequences according to the boundaries defined in␣
↪the transcript/gene feature records unless there is no transcript/gene feature record␣
↪in the GTF file.** In this case, pyroe imputes all transcripts/genes boundaries as the␣
↪bounds of the corresponding exons to extract unspliced sequences.
If the transcript/gene feature records do not match their exon feature records, pyroe␣
↪will still use transcript/gene feature records, but correct those transcript/gene␣
↪feature records in the celan_grf.gtf according to exon feature records.
If using bedtools, a temp.bed and a temp.fa will be created and then deleted. These two␣
↪files encode the introns of each gene and the exons of each transcript of each gene.
```

1.2.3 Notes on the input gene annotation GTF files for building an expanded reference

Pyroe builds expanded transcriptome references, the spliced + intronic (*splici*) and the spliced + unspliced (*spliceu*) transcriptome reference, based on a genome build FASTA file and a gene annotation GTF file.

The input GTF file will be processed before extracting unspliced sequences. If pyroe finds invalid records, a `clean_gtf.gtf` file will be generated in the specified output directory. **Note** : The features extracted in the spliced + unspliced transcriptome will not necessarily be those present in the `clean_gtf.gtf` file — as this command will prefer the input in the user-provided file wherever possible. One can rerun pyroe using the `clean_gtf.gtf` file if needed. More specifically:

1. The non-gene level records, those whose `feature` field value is not "gene, " must have a valid `transcript_id`. If this is not satisfied, pyroe returns an error and writes only the records with a valid `transcript_id` to the `clean_gtf.gtf` file. One can rerun pyroe using the *clean_gtf.gtf* file to ignore those invalid records if needed.

2. For `gene_id` and `gene_name` metadata field,

   - If these two fields are entirely missing in the GTF file, An error will be returned. At the same time, in the `clean_gtf.gtf`, the two fields will be imputed using the `transcript_id` field.
   - If one of `gene_id` and `gene_name` is completely missing, pyroe will print a warning, impute the missing field using the other one, and move to the next step with the imputed data.
   - if some records have missing `gene_id`, `gene_name`, or both, pyroe will print a warning and move to the next step after imputing the missing values by the following rules: For records missing `gene_id` or `gene_name`, pyroe imputes the missing one using the other one; If both are missing, pyroe imputes both of them using its `transcript_id`, which cannot be missing.

3. If the GTF file does not contain transcript or gene level records, those whose `feature` field value is "transcript" or "gene", pyroe will print a warning and impute those missing records using the exon level records of transcripts

and genes, in which the `Start` and `End` fields will be imputed as the bounds of the corresponding exons.

4. If the boundaries of transcripts/genes defined in the "transcript" or "gene" level records – those whose `feature` field value is either "transcript" or "gene" – do not match those implied by their exons' feature records, or the transcript/gene level records of some transcripts/genes' are missing, pyroe will report a warning, fix all those gene/transcript level records using their exon level records and write them to the `clean_gtf.gtf` file, but still extract unspliced sequences based on the existing transcript/gene level records.

1.3 Processing alevin-fry quantification result

The quantification result of alevin-fry can be loaded into python by the `load_fry()` function. This function takes a output directory returned by `alevin-fry quant` command as the minimum input, and load the quantification result as an `AnnData` object. When processing USA mode result, it assumes that the data comes from a single-cell RNA-sequencing experiment. If one wants to process single-nucleus RNA-sequencing data or prepare the single-cell data for RNA-velocity analysis, the `output_format` argument should be set as `snRNA` or `velocity` correspondingly. One can also define customized output format, see the Full Usage section for detail.

1.4 `load_fry()` full Usage

load alevin-fry quantification result into an AnnData object

1.4.1 Required Parameters

frydir

[`str`] The path to a output directory returned by alevin-fry quant command. The directory containing the alevin-fry quantification (i.e. the the quant.json file & alevin subdirectory).

1.4.2 Optional Parameters

output_format

[`str` or `dict`] A string represents one of the pre-defined output formats, which are "scRNA", "snRNA" and "velocity". If a customized format of the returned *AnnData* is needed, one can pass a Dictionary. See Notes section for details.

quiet

[`bool` (default: `True`)] True if function should be quiet.False if messages (including error messages) should be printed out.

nonzero

[`bool` (default: `False`)] True if cells with non-zero expression value across all genes should be filtered in each layer.False if unexpressed genes should be kept.

1.4.3 *load_fry* Notes

The `output_format` argument takes either a dictionary that defines the customized format or a string that represents one of the pre-defined format of the returned `AnnData` object.

Each of the pre-defined formats contains a `X` field and some optional extra `AnnData.layers` obtained from the sub-matrices representing unspliced (U), spliced (S) and ambiguous (A) counts returned by alevin-fry.

The following formats are defined:

- **"scRNA":**
  This format is recommended for single cell RNA-sequencing experiments. It returns a *X* field that contains the S+A count of each gene in each cell , and an extra *unspliced* field that contains the U count of each gene in each cell.
- **"snRNA", "U+S+A", "all":**
  These formats are recommended for single nucleus RNA-sequencing experiments. Furthermore, these formats match the behaviors of Cell Ranger 7, which by default includes all intronic reads in the output gene count matrix for both single-cell and single-nucleus experiments.These formats return a *X* field that contains the U+S+A count of each gene in each cell without any extra layers.
- **"raw":**
  This format uses the S count matrix as the *X* field and put the U, S, and A counts into three separate layers, which are *unspliced*, *spliced* and *ambiguous*.
- **"S+A":**
  This format uses the U + S counts as the *X* field without any extra layers.
- **"velocity":**
  This format is the same as "scRNA", except it contains a *spliced* layer, which contains the S+A counts.

A custom output format can be defined using a Dictionary specifying the desired format of the output `Anndata` object. If the input is not a USA mode quantification directory, this parameter is ignored and the count matrix is returned in the *X* field of the returned `AnnData` object. If the input quantification directory contains a USA mode quantification, then there are 3 sub-matrices that can be referenced in the dictionary; 'U', 'S', 'A' containing, respectively, unspliced, spliced and ambiguous counts. The dictionary should have entries of the form `key` (str) : `value` (list[str]). The following constraints apply : there should be one key-value pair with the key `X`, the resulting counts will be returned in the `X` field of the AnnData object. There can be an arbitrary number of other key-value pairs, but each will be returned as a layer of the resulting AnnData object. Within the key-value pairs, the key refers to the layer name that will be given to the combined count matrix upon output, and the value should be a subset of `['U', 'S', 'A']` that defines which sub-matrices should be summed. For example: `{'X' : ['S', 'A'], 'unspliced' :  ['U']}` will result in a return AnnData object where the X field has a matrix in which each entry corresponds to the summed spliced and ambiguous counts for each gene in each cell, and there is an additional "unspliced" layer, whose counts are taken directly from the unspliced sub-matrix.

1.4.4 Returns

An AnnData object with X and layers corresponding to the requested `output_format`.

1.5 Fetching and loading preprocessed quantification results

The raw data for many single-cell and single-nucleus RNA-seq experiments is publicly available. However, certain datasets are used *again and again*, to demonstrate data processing in tutorials, as benchmark datasets for novel methods (e.g. for clustering, dimensionality reduction, cell type identification, etc.). In particular, 10x Genomics hosts various publicly available datasets generated using their technology and processed via their Cell Ranger software on their website for download.

We have created a Nextflow-based `alevin-fry` workflow that one can use to easily quantify single-cell RNA-sequencing data in a single workflow. The pipeline can be found here. To test out this initial pipeline, we have begun to reprocess the publicly-available datasets collected from the 10x website. We have focused the initial effort on standard single-cell and single-nucleus gene-expression data generated using the Chromium v2 and v3 chemistries, but hope to expand the pipeline to more complex protocols soon (e.g. feature barcoding experiments) and process those data as well. We note that these more complex protocols can already be processed with `alevin-fry` (see the alevin-fry tutorials), but these have just not yet been incorprated into the automated Nextflow-based workflow linked above.

We provide two python functions:

- `fetch_processed_quant()` can fetch the quantification result of one or more available datasets according to the provided `dataset_ids` vector, and store them to a local folder.
- `load_processed_quant()` can fetch the quantification result of one or more available dataset as `fetch_processed_quant()`, and load them into python as `AnnData` objects. We also provide a CLI for fetching quantification results.

```
pyroe fetch-quant 1 3 6
```

or start python, and run

```
import pyroe

# fetch, decompress and load the quantification result of dastset #1, 3 and 6
pq_dict = pyroe.load_processed_quant([1,3,6])

# get the ProcessedQuant class object for dataset #1 and #3
pq_ds1 = pq_dict[1]
pq_ds3 = pq_dict[3]

# get the dataset name
pq_ds1.dataset_name
pq_ds3.dataset_name


# get the path to the quantification result
pq_ds1.quant_path
pq_ds3.quant_path

# get the AnnData
pq_ds1.anndata
pq_da3.anndata
```

1.5.1 fetch-quant full usage

```
usage: pyroe fetch-quant [-h] [--fetch_dir FETCH_DIR] [--force] [--delete_tar]
                         [--quiet]
                         dataset-ids [dataset-ids ...]

positional arguments:
  dataset-ids           The ids of the datasets to fetch

optional arguments:
  -h, --help            show this help message and exit
  --fetch_dir FETCH_DIR
                        The path to a directory for storing fetched datasets.
  --force               A flag indicates whether existing datasets will be redownloaded␣
↪by force.
  --delete_tar          A flag indicates whether fetched tar files will be deleted.
  --quiet               A flag indicates whether help messaged should not be printed.

1. 500 Human PBMCs, 3' LT v3.1, Chromium Controller
2. 500 Human PBMCs, 3' LT v3.1, Chromium X
3. 1k PBMCs from a Healthy Donor (v3 chemistry)
4. 10k PBMCs from a Healthy Donor (v3 chemistry)
5. 10k Human PBMCs, 3' v3.1, Chromium X
6. 10k Human PBMCs, 3' v3.1, Chromium Controller
7. 10k Peripheral blood mononuclear cells (PBMCs) from a healthy donor, Single Indexed
8. 10k Peripheral blood mononuclear cells (PBMCs) from a healthy donor, Dual Indexed
9. 20k Human PBMCs, 3' HT v3.1, Chromium X
10. PBMCs from EDTA-Treated Blood Collection Tubes Isolated via SepMate-Ficoll Gradient␣
↪(3' v3.1 Chemistry)
11. PBMCs from Heparin-Treated Blood Collection Tubes Isolated via SepMate-Ficoll␣
↪Gradient (3' v3.1 Chemistry)
12. PBMCs from ACD-A Treated Blood Collection Tubes Isolated via SepMate-Ficoll Gradient␣
↪(3' v3.1 Chemistry)
13. PBMCs from Citrate-Treated Blood Collection Tubes Isolated via SepMate-Ficoll␣
↪Gradient (3' v3.1 Chemistry)
14. PBMCs from Citrate-Treated Cell Preparation Tubes (3' v3.1 Chemistry)
15. PBMCs from a Healthy Donor: Whole Transcriptome Analysis
16. Whole Blood RBC Lysis for PBMCs and Neutrophils, Granulocytes, 3'
17. Peripheral blood mononuclear cells (PBMCs) from a healthy donor - Manual (channel 5)
18. Peripheral blood mononuclear cells (PBMCs) from a healthy donor - Manual (channel 1)
19. Peripheral blood mononuclear cells (PBMCs) from a healthy donor - Chromium Connect␣
↪(channel 5)
20. Peripheral blood mononuclear cells (PBMCs) from a healthy donor - Chromium Connect␣
↪(channel 1)
21. Hodgkin's Lymphoma, Dissociated Tumor: Whole Transcriptome Analysis
22. 200 Sorted Cells from Human Glioblastoma Multiforme, 3’ LT v3.1
23. 750 Sorted Cells from Human Invasive Ductal Carcinoma, 3’ LT v3.1
24. 2k Sorted Cells from Human Glioblastoma Multiforme, 3’ v3.1
25. 7.5k Sorted Cells from Human Invasive Ductal Carcinoma, 3’ v3.1
26. Human Glioblastoma Multiforme: 3’v3 Whole Transcriptome Analysis
27. 1k Brain Cells from an E18 Mouse (v3 chemistry)
28. 10k Brain Cells from an E18 Mouse (v3 chemistry)
29. 1k Heart Cells from an E18 mouse (v3 chemistry)
```

(continues on next page)

(continued from previous page)

```
30. 10k Heart Cells from an E18 mouse (v3 chemistry)
31. 10k Mouse E18 Combined Cortex, Hippocampus and Subventricular Zone Cells, Single␣
↪Indexed
32. 10k Mouse E18 Combined Cortex, Hippocampus and Subventricular Zone Cells, Dual␣
↪Indexed
33. 1k PBMCs from a Healthy Donor (v2 chemistry)
34. 1k Brain Cells from an E18 Mouse (v2 chemistry)
35. 1k Heart Cells from an E18 mouse (v2 chemistry)
```

1.5.2 The ProcessedQuant class

To store the information of a dataset, we provide the `ProcessedQuant` class, which can be simply instantiated using a dataset id, for example, `ProcessedQuant(2)` will return an instance of the `ProcessedQuant` class containing the detail of dataset #2, 500 Human PBMCs, 3' LT v3.1, Chromium X. This class contains methods for fetching, decompressing and loading the quantification result of the corresponding dataset. After getting an instance of the class, i.e., running `pq = ProcessedQuant(dataset_id)`, one can run the following commands to fetch, decompress and/or load the quantification result of the dataset:

- `pq.fetch_quant()` fetches the compressed quantification result of the corresponding dataset into a local directory and stores the path in its `tar_path` attribute.
- `pq.decompress_quant()` decompresses the fetched quantification result into a local directory and stores the path in its `quant_path` attribute.
- `pq.load_quant()` loads the decompressed quantification result into python as an *AnnData* object and stores the object in its `anndata` attribute.

Besides, we have some helper function for printing and loading the information of the available datasets:

- `ProcessedQuant.get_available_dataset_df()` returns the detail of available datasets as a pandas dataframe.
- `ProcessedQuant.print_available_datasets()` prints the index and name of the available datasets.

1.6 Generating a gene id to gene name mapping

It is often useful to perform analyses with gene *names* rather than gene *identifiers*. The convert command of `pyroe` allows you to specify an id to name mapping so that the converted output matrix will be labeled with gene names rather than identifiers. However, you must provide it with a 2-column tab-separated file mapping IDs to names. This command can help you with that task.

The `id-to-name` command takes as input 2 parameters, an annotation file and the location where an output file should be written. The annotation file can be either a GTF or GFF3 file (which, optionally, can be gzipped). The program will attempt to figure out the format automatically from the file suffix(es); if it cannot, you may also provide the parameter `--format` taking the argument either `gtf` or `gff3` to specify the format of the input. The `id-to-name` command will use the PyRanges package to parse the file and extract the gene id to gene name mapping, and will write this mapping at the provided output path.

Note: The `id-to-name` sub-command assumes that the input annotation file has a field named `gene_id` and another named `gene_name`. Please make sure the input has these fields to produce a proper output mapping. We may allow specifying a custom `gene_id` identifier and `gene_name` identifier in the future.

1.6.1 `id-to-name` full usage

```
usage: pyroe id-to-name [-h] [--format FORMAT] gtf_file output

positional arguments:
gtf_file         The GTF input file.
output           The path to where the output tsv file will be written.

optional arguments:
-h, --help       show this help message and exit
--format FORMAT  The input format of the file (must be either GTF or GFF3). This will be␣
↪inferred from the filename, but if that fails it can be provided explicitly.
```

1.7 Converting quantification results

The `convert` sub-command of `pyroe` can convert the output of *alevin-fry* into several common formats, such as the native *AnnData* format (`h5ad`). Further, when performing this conversion, it can organize the unspliced, spliced, and ambiguous counts as desired by the user.

The sub-command takes as input a quantification directory produced by `alevin-fry`, and an output location. Additionally, the user should pass in command line parameters to describe the desired output structure, and output format. The output structure defines how the `U`, `S`, and `A` layers of the input quantification should be represented in the converted matrix. The syntax for this flag exactly mimics the `output_format` argument of the `load_fry` function, which you can read about here. Note that, if you pass in a custom output structure, you should enclose your format description in quotes. For example, to output to an object where the "main" layer (`X`) contains the sum of `U`, `S`, and `A`, and where there is an additional layer named *unspliced* having just the unspliced counts, you would pass `--output-structure '{ "X" :  ["U", "S", "A"], "unspliced" :  ["U"]}'`.

If you do not explicitly provide an `--output-format`, the default of `h5ad` will be used.

The *optional* `--geneid-to-name` parameter allows you to pass in a 2-column tab-separated filed mapping gene identifiers to gene names. If this is provided, then gene IDs will be converted to gene names in the output matrix. Gene names will be made unique using the `var_names_make_unique()` function of ScanPy. It is also possible that some gene IDs do not have a mapped name. In this case, the `convert` subcommand will also write out a JSON format file, at the provided output path, with the additional suffix `_unmapped_ids.json`. This file contains a list of the gene IDs that could not successfully be mapped to a name given the provided mapping.

1.7.1 `convert` command full usage

```
usage: pyroe convert [-h] [--output-structure OUTPUT_STRUCTURE] [--output-format OUTPUT_
↪FORMAT] [--geneid-to-name GENEID_TO_NAME] quant_dir output

positional arguments:
  quant_dir             The input quantification directory containing the matrix to be␣
↪converted.
  output                The output name where the quantification matrix should be␣
↪written. For `csvs` output format, this will be a directory. For all others, it will␣
↪be a file.

optional arguments:
  -h, --help            show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  --output-structure OUTPUT_STRUCTURE
                        The structure that U,S and A counts should occupy in the output␣
↪matrix.
  --output-format OUTPUT_FORMAT
                        The format in which the output should be written, one of {'zarr',
↪ 'loom', 'csvs', 'h5ad'}.
  --geneid-to-name GENEID_TO_NAME
                        A 2 column tab-separated list of gene ID to gene name mappings.␣
↪Providing this file will project gene IDs to gene names in the output.
```

1.8 License

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search