dynast

Release 0.2.0

Kyung Hoi (Joseph) Min

May 09, 2022

INTRODUCTION:

Warning: Dynast is currently in beta-testing and is still under active development. Usage may change without warning.

CHAPTER

ONE

GETTING STARTED

Welcome to dynast!

Dynast is a command-line pipeline that preprocesses data from metabolic labeling scRNA-seq experiments and quantifies the following four mRNA species: unlabeled unspliced, unlabeled spliced, labeled unspliced and labeled spliced. In addition, dynast can perform statistical estimation of these species through expectation maximization (EM) and Bayesian inference. Please see *Statistical estimation* for more details on how the statistical estimation is performed.

1.1 Installation

The latest stable release of dynast is available on the Python Package Index (Pypi) and can be installed with pip:

```
pip install dynast-release
```

To install directly from the Git repository:

```
pip install git+https://github.com/aristoteleo/dynast-release
```

To install the latest *development* verson:

```
pip install git+https://github.com/aristoteleo/dynast-release@devel
```

Please note that not all features may be stable when using the development version.

1.2 Command-line structure

Dynast consists of four commands that represent four steps of the pipeline: `ref`, `align`, `consensus`, `count`, `estimate`. This modularity allows users to add additional preprocessing between steps as they see fit. For instance, a user may wish to run a custom step to mark and remove duplicates after the `align` step.

Command	Description
`ref`	Build a STAR index from a reference genome FASTA and GTF.
`align`	Align FASTQs into an alignment BAM.
`consensus`	Call consensus sequence for each sequenced mRNA molecule.
`count`	Quantify unlabeled and labeled RNA.
`estimate`	Estimate the fraction of labeled RNA via statistical estimation.

1.3 Basic usage

Prerequisites:

- FASTQ files from a metabolic labeling scRNA-seq experiment
- **[Optional]** STAR genome index for the appropriate organism. Skip the first step if you already have this.

1.3.1 Build the STAR index

First, we must build a STAR index for the genome of the organism that was used in the experiment. For the purpose of this section, we will be using the mouse (Mus musculus) as an example. Download the **genome (DNA) FASTA** and **gene annotations GTF**. If you already have an appropriate STAR index, you do not need to re-generate it and may skip to the next step.

```
dynast ref -i STAR Mus_musculus.GRCm38.dna.primary_assembly.fa.gz Mus_musculus.GRCm38.
↪102.gtf.gz
```

where `STAR` is the directory to which we will be saving the STAR index.

1.3.2 Align FASTQs

Next, we align the FASTQs to the genome.

```
dynast align -i STAR -o align -x TECHNOLOGY CDNA_FASTQ BARCODE_UMI_FASTQ
```

where `align` is the directory to which to save alignment files, and `TECHNOLOGY` is a scRNA-seq technology. A list of supported technologies can be found by running `dynast --list`. `BARCODE_UMI_FASTQ` is the FASTQ containing the barcode and UMI sequences, whereas the `CDNA_FASTQ` is the FASTQ containing the biological cDNA sequences.

1.3.3 [Optional] Consensus

Optionally, we can call consensus sequences for each sequenced mRNA molecule.

```
dynast consensus -g Mus_musculus.GRCm38.102.gtf.gz --barcode-tag CB --umi-tag UB -o␣
↪consensus align/Aligned.sortedByCoord.out.bam
```

where `consensus` is the directory to which to save the consensus-called BAM. Once the above command finishes, the `consensus` directory will contain a new BAM file that can be used as input to the following step.

1.3.4 Quantify

Finally, we quantify the four RNA species of interest. Note that we re-use the gene annotations GTF.

```
dynast count -g Mus_musculus.GRCm38.102.gtf.gz --barcode-tag CB --umi-tag UB -o count --
↪barcodes align/Solo.out/Gene/filtered/barcodes.tsv --conversion TC align/Aligned.
↪sortedByCoord.out.bam
```

where `count` is the directory to which to save RNA quantifications. We provide a filtered barcode list `align/Solo.out/Gene/filtered/barcodes.tsv`, which was generated from the previous step, so that only these barcodes are processed during quantification. We specify the experimentally induced conversion with `--conversion`. In this example, our experiment introduces T-to-C conversions.

Once the above command finishes, the `count` directory will contain an `adata.h5ad` AnnData file containing all quantification results.

1.3.5 [Optional] Estimate

Optionally, we can estimate the unlabeled and labeled counts by statistically modelling the labeling dynamics (see *Statistical estimation*).

```
dynast estimate -o estimate count
```

where `estimate` is the directory to which to save RNA quantifications. We provide the directory that contains the quantification results (i.e. `-o` option of `dynast count`).

Once the above command finishes, the `estimate` directory will contain an `adata.h5ad` AnnData file containing all quantification and estimation results.

CHAPTER TWO

PIPELINE USAGE

This sections covers basic usage of dynast.

2.1 Building the STAR index with `ref`

Internally, dynast uses the STAR RNA-seq aligner to align reads to the genome [Dobin2013]. Therefore, we must construct a STAR index to use for alignment. The `dynast ref` command is a wrapper around the STAR's `--runMode genomeGenerate` command, while also providing useful default parameters to limit memory usage, similar to Cell Ranger. Existing STAR indices can be used interchangeably with ones generated through dynast. A genome FASTA and gene annotation GTF are required to build the STAR index.

```
usage: dynast ref [-h] [--tmp TMP] [--keep-tmp] [--verbose] [-t THREADS] -i INDEX [-m␣
↪MEMORY] fasta gtf

Build a STAR index from a reference

positional arguments:
  fasta       Genomic FASTA file
  gtf         Reference GTF file

optional arguments:
  -h, --help  Show this help message and exit
  --tmp TMP   Override default temporary directory
  --keep-tmp  Do not delete the tmp directory
  --verbose   Print debugging information
  -t THREADS  Number of threads to use (default: 8)
  -m MEMORY   Maximum memory used, in GB (default: 16)

required arguments:
  -i INDEX    Path to the directory where the STAR index will be generated
```

2.2 Aligning FASTQs with `align`

The `dynast align` command is a wrapper around STARsolo [Dobin2013]. Dynast automatically formats the arguments to STARsolo to ensure the resulting alignment BAM contains information necessary for downstream processing.

Additionally, `align` sets a more lenient alignment score cutoff by setting `--outFilterScoreMinOverLread 0.3 --outFilterMatchNminOverLread 0.3` because the reads are expected to have experimentally-induced conversions. The STARsolo defaults for both are `0.66`. The `--STAR-overrides` argument can be used to pass arguments directly to STAR.

`dynast align` outputs a different set of BAM tags in the alignment BAM depending on the type of sequencing technology specified. These are described in the following subsections.

```
usage: dynast align [-h] [--tmp TMP] [--keep-tmp] [--verbose] [-t THREADS] -i INDEX [-o␣
↪OUT] -x TECHNOLOGY
                    [--strand {forward,reverse,unstranded}] [-w WHITELIST] [--overwrite]
                    [--STAR-overrides ARGUMENTS]
                    fastqs [fastqs ...]

Align FASTQs

positional arguments:
  fastqs                FASTQ files. If `-x smartseq`, this is a single manifest CSV␣
↪file where the first
                        column contains cell IDs and the next two columns contain paths␣
↪to FASTQs (the third
                        column may contain a dash `-` for single-end reads).

optional arguments:
  -h, --help            Show this help message and exit
  --tmp TMP             Override default temporary directory
  --keep-tmp            Do not delete the tmp directory
  --verbose             Print debugging information
  -t THREADS            Number of threads to use (default: 8)
  --strand {forward,reverse,unstranded}
                        Read strandedness. (default: `forward`)
  -w WHITELIST          Path to file of whitelisted barcodes to correct to. If not␣
↪provided, all barcodes are
                        used.
  --overwrite           Overwrite existing alignment files
  --STAR-overrides ARGUMENTS
                        Arguments to pass directly to STAR.

required arguments:
  -i INDEX              Path to the directory where the STAR index is located
  -o OUT                Path to output directory (default: current directory)
  -x TECHNOLOGY         Single-cell technology used. `dynast --list` to view all␣
↪supported technologies
```

2.2.1 UMI-based technologies

For UMI-based technologies (such as Drop-seq, 10X Chromium, scNT-seq), the following BAM tags are written to the alignment BAM.

- `MD`
- `HI`, `AS` for alignment index and score
- `CR`, `CB` for raw and corrected barcodes
- `UR`, `UB` for raw and corrected UMIs

2.2.2 Plate-based technologies

For plate-based technologies (such as Smart-Seq), the following BAM tags are written to the alignment BAM. See

- `MD`
- `HI`, `AS` for alignment index and score
- `RG` indicating the sample name

2.3 Calling consensus sequences with `consensus`

`dynast consensus` parses the alignment BAM to generate consensus sequences for each sequenced mRNA molecule (see *Consensus procedure*).

```
usage: dynast consensus [-h] [--tmp TMP] [--keep-tmp] [--verbose] [-t THREADS] -g GTF [-
↪o OUT] [--umi-tag TAG]
                        [--barcode-tag TAG] [--gene-tag TAG] [--strand {forward,reverse,
↪unstranded}]
                        [--quality QUALITY] [--barcodes TXT] [--add-RS-RI]
                        bam

Generate consensus sequences

positional arguments:
  bam                   Alignment BAM file that contains the appropriate UMI and barcode␣
↪tags, specifiable with
                        `--umi-tag`, and `--barcode-tag`.

optional arguments:
  -h, --help            Show this help message and exit
  --tmp TMP             Override default temporary directory
  --keep-tmp            Do not delete the tmp directory
  --verbose             Print debugging information
  -t THREADS            Number of threads to use (default: 8)
  -o OUT                Path to output directory (default: current directory)
  --umi-tag TAG         BAM tag to use as unique molecular identifiers (UMI). If not␣
↪provided, all reads are assumed
                        to be unique. (default: None)
  --barcode-tag TAG     BAM tag to use as cell barcodes. If not provided, all reads are␣
↪assumed to be from a single
```

(continues on next page)

(continued from previous page)

```
                        cell. (default: None)
  --gene-tag TAG        BAM tag to use as gene assignments (default: GX)
  --strand {forward,reverse,unstranded}
                        Read strandedness. (default: `forward`)
  --quality QUALITY     Base quality threshold. When generating a consensus nucleotide␣
↪at a certain position, the base
                        with smallest error probability below this quality threshold is␣
↪chosen. If no base meets this
                        criteria, the reference base is chosen. (default: 27)
  --barcodes TXT        Textfile containing filtered cell barcodes. Only these barcodes␣
↪will be processed.
  --add-RS-RI           Add custom RS and RI tags to the output BAM, each of which␣
↪contain a semi-colon delimited list
                        of read names (RS) and alignment indices (RI) of the reads and␣
↪alignments from which the
                        consensus is derived. This option is useful for debugging.

required arguments:
  -g GTF                Path to GTF file used to generate the STAR index
```

The resulting BAM will contain a collection of consensus alignments and a subset of original alignments (for those alignments for which a consensus could not be determined). The latter are identical to those in the original BAM, while the names of the former will be seemingly random sequences of letters and numbers (in reality, these are SHA256 checksums of the grouped read names). They will also contain the following modified BAM tags

- `AS` is now the *sum* of the alignment scores of the reads
- `HI`, the alignment index, is always 1

and the follwing additional BAM tags.

- `RN` indicating how many reads were used to generate the consensus
- `RS`, `RI` each containing a semicolon-delimited list of read names and their corresponding alignment indices (`HI` tag in the original BAM) that were used to generate the consensus (only added if `--add-RS-RI` is provided)

2.4 Quantifying counts with `count`

`dynast count` parses the alignment BAM and quantifies the four RNA species (unlabeled unspliced, unlabeled spliced, labeled unspliced, labeled spliced) and outputs the results as a ready-to-use AnnData `H5AD` file. In order to properly quantify the above four species, the alignment BAM must contain specific BAM tags, depending on what sequencing technology was used. If `dynast align` was used to generate the alignment BAM, dynast automatically configures the appropriate BAM tags to be written.

```
usage: dynast count [-h] [--tmp TMP] [--keep-tmp] [--verbose] [-t THREADS] -g GTF --
↪conversion CONVERSION [-o OUT]
                    [--umi-tag TAG] [--barcode-tag TAG] [--gene-tag TAG] [--strand
↪{forward,reverse,unstranded}]
                    [--quality QUALITY] [--snp-threshold THRESHOLD] [--snp-min-coverage␣
↪THRESHOLD] [--snp-csv CSV]
                    [--barcodes TXT] [--no-splicing | --exon-overlap {lenient,strict}] [-
↪-control]
                    [--dedup-mode {auto,conversion,exon}] [--overwrite]
```

(continues on next page)

(continued from previous page)

```
                    bam

Quantify unlabeled and labeled RNA

positional arguments:
  bam                   Alignment BAM file that contains the appropriate UMI and barcode␣
↪tags, specifiable with
                        `--umi-tag`, and `--barcode-tag`.

optional arguments:
  -h, --help            Show this help message and exit
  --tmp TMP             Override default temporary directory
  --keep-tmp            Do not delete the tmp directory
  --verbose             Print debugging information
  -t THREADS            Number of threads to use (default: 8)
  -o OUT                Path to output directory (default: current directory)
  --umi-tag TAG         BAM tag to use as unique molecular identifiers (UMI). If not␣
↪provided, all reads are assumed
                        to be unique. (default: None)
  --barcode-tag TAG     BAM tag to use as cell barcodes. If not provided, all reads are␣
↪assumed to be from a single
                        cell. (default: None)
  --gene-tag TAG        BAM tag to use as gene assignments (default: GX)
  --strand {forward,reverse,unstranded}
                        Read strandedness. (default: `forward`)
  --quality QUALITY     Base quality threshold. Only bases with PHRED quality greater␣
↪than this value will be
                        considered when counting conversions. (default: 27)
  --snp-threshold THRESHOLD
                        Conversions with (# conversions) / (# reads) greater than this␣
↪threshold will be considered a
                        SNP and ignored. (default: no SNP detection)
  --snp-min-coverage THRESHOLD
                        For a conversion to be considered as a SNP, there must be at␣
↪least this many reads mapping to
                        that region. (default: 1)
  --snp-csv CSV         CSV file of two columns: contig (i.e. chromosome) and genome␣
↪position of known SNPs
  --barcodes TXT        Textfile containing filtered cell barcodes. Only these barcodes␣
↪will be processed.
  --no-splicing, --transcriptome-only
                        Do not assign reads a splicing status (spliced, unspliced,␣
↪ambiguous) and ignore reads that
                        are not assigned to the transcriptome.
  --exon-overlap {lenient,strict}
                        Algorithm to use to detect spliced reads (that overlap exons).␣
↪May be `strict`, which assigns
                        reads as spliced if it only overlaps exons, or `lenient`, which␣
↪assigns reads as spliced if it
                        does not overlap with any introns of at least one transcript.␣
↪(default: lenient)
  --control             Indicate this is a control sample, which is used to detect SNPs.
```

(continues on next page)

(continued from previous page)

```
  --dedup-mode {auto,conversion,exon}
                        Deduplication mode for UMI-based technologies (required `--umi-
↪tag`). Available choices are:
                        `auto`, `conversion`, `exon`. When `conversion` is used, reads␣
↪that have at least one of the
                        provided conversions is prioritized. When `exon` is used, exonic␣
↪reads are prioritized. By
                        default (`auto`), the BAM is inspected to select the appropriate␣
↪mode.
  --overwrite           Overwrite existing files.

required arguments:
  -g GTF                Path to GTF file used to generate the STAR index
  --conversion CONVERSION
                        The type of conversion(s) introduced at a single timepoint.␣
↪Multiple conversions can be
                        specified with a comma-delimited list. For example, T>C and A>G␣
↪is TC,AG. This option can be
                        specified multiple times (i.e. dual labeling), for each labeling␣
↪timepoint.
```

2.4.1 Basic arguments

The `--barcode-tag` and `--umi-tag` arguments are used to specify what BAM tags should be used to differentiate cells (barcode) and RNA molecules (UMI). If the former is not specified, all BAM alignments are assumed to be from a single cell, and if the latter is not specified, all aligned reads are assumed to be unique (i.e. no read deduplication is performed). If `align` was used to generate the alignment BAM, then `--barcode-tag CB --umi-tag UB` is recommended for UMI-based technologies (see *UMI-based technologies*), and `--barcode-tag RG` is recommended for Plate-based technologies (see *Plate-based technologies*).

The `--strand` argument can be used to specify the read strand of the sequencing technology. Usually, the default (`forward`) is appropriate, but this argument may be of use for other technologies.

The `--conversion` argument is used to specify the type of conversion that is experimentally introduced as a two-character string. For instance, a T>C conversion is represented as `TC`, which is the default. Multiple conversions can be specified as a comma-delimited list, and `--conversion` may be specified multiple times to indicate multiple-indexing experiments. For example, for an experiment that introduced T>C mutations at timepoint 1 and A>G and C>G mutations at timepoint 2, the appropriate options would be `--conversion TC --conversion AG,CG`.

2.4.2 Detecting and filtering SNPs

`dynast count` has the ability to detect single-nucleotide polymorphisms (SNPs) by calculating the fraction of reads with a mutation at a certain genomic position. `--snp-threshold` can be used to specify the proportion threshold greater than which a SNP will be called at that position. All conversions/mutations at the genomic positions with SNPs detected in this manner will be filtered out from further processing. In addition, a CSV file containing known SNP positions can be provided with the `--snp-csv` argument. This argument accepts a CSV file containing two columns: contig (i.e. chromosome) and genomic position of known SNPs.

2.4.3 Read deduplication modes

The `--dedup-mode` option is used to select how duplicate reads should be deduplicated for UMI-based technologies (i.e. `--umi-tag` is provided). Two different modes are supported: `conversion` and `exon`. The former prioritizes reads that have at least one conversions provided by `--conversion`. The latter prioritizes exonic reads. See *quant* for a more technical description of how deduplication is performed. Additionally, see *Consensus procedure* to get an idea of why selecting the correct option may be important.

By default, the `--dedup-mode` is set to `auto`, which sets the deduplication mode to `exon` if the input BAM is detected to be a consensus-called BAM (a BAM generated with `dynast consensus`). Otherwise, it is set to `conversion`. This option has no effect for non-UMI technologies.

2.5 Estimating counts with `estimate`

The fraction of labeled RNA is estimated with the `dynast estimate` command. Whereas `dynast count` produces naive UMI count matrices, `dynast estimate` statistically models labeling dynamics to estimate the true fraction of labeled RNA (and then in turn uses this fraction to split the total UMI counts into unlabeled and labeled RNA). See *Statistical estimation* of a technical overview of this process. In this section, we will simply be describing the command-line usage of this command.

```
usage: dynast estimate [-h] [--tmp TMP] [--keep-tmp] [--verbose] [-t THREADS]
                       [--reads {total,transcriptome,spliced,unspliced}] [-o OUT] [--
↪groups CSV]
                       [--ignore-groups-for-pi] [--genes TXT] [--cell-threshold COUNT] [-
↪-cell-gene-threshold COUNT]
                       [--downsample NUM] [--downsample-mode MODE] [--control] [--p-e P_
↪E]
                       count_dirs [count_dirs ...]

Estimate fraction of labeled RNA

positional arguments:
  count_dirs            Path to directory that contains `dynast count` output. When␣
↪multiple are provided, the
                        barcodes in each of the count directories are suffixed with `-i`␣
↪where i is a 0-indexed
                        integer.

optional arguments:
  -h, --help            Show this help message and exit
  --tmp TMP             Override default temporary directory
  --keep-tmp            Do not delete the tmp directory
  --verbose             Print debugging information
  -t THREADS            Number of threads to use (default: 8)
  --reads {total,transcriptome,spliced,unspliced}
                        Read groups to perform estimation on. This option can be used␣
↪multiple times to estimate
                        multiple groups. (default: all possible reads groups)
  -o OUT                Path to output directory (default: current directory)
  --groups CSV          CSV containing cell (barcode) groups, where the first column is␣
↪the barcode and the second is
                        the group name the cell belongs to. Cells will be combined per␣
↪group for estimation of
```

(continues on next page)

(continued from previous page)

```
                        parameters specified by `--groups-for`.
  --ignore-groups-for-pi
                        Ignore cell groupings when estimating the fraction of labeled␣
↪RNA. This option only has an
                        effect when `--groups` is also specified.
  --genes TXT           Textfile containing list of genes to use. All other genes will␣
↪be treated as if they do not
                        exist.
  --cell-threshold COUNT
                        A cell must have at least this many reads for correction.␣
↪(default: 1000)
  --cell-gene-threshold COUNT
                        A cell-gene pair must have at least this many reads for␣
↪correction. (default: 16)
  --downsample NUM      Downsample the number of reads (UMIs). If a decimal between 0␣
↪and 1 is given, then the number
                        is interpreted as the proportion of remaining reads. If an␣
↪integer is given, the number is
                        interpreted as the absolute number of remaining reads.
  --downsample-mode MODE
                        Downsampling mode. Can be one of: `uniform`, `cell`, `group`. If␣
↪`uniform`, all reads (UMIs)
                        are downsampled uniformly at random. If `cell`, only cells that␣
↪have more reads than the
                        argument to `--downsample` are downsampled to exactly that␣
↪number. If `group`, identical to
                        `cell` but per group specified by `--groups`.
  --control             Indicate this is a control sample, only the background mutation␣
↪rate will be estimated.
  --p-e P_E             Textfile containing a single number, indicating the estimated␣
↪background mutation rate
```

2.5.1 Estimation thresholds

The `--cell-threshold` and `--cell-gene-threshold` arguments control the minimum number of reads that a cell and cell-gene combination must have for accurate estimation. By default, these are `1000` and `16` respectively. Any cells with reads less than the former are excluded from estimation, and the same goes for any genes within a cell that has less reads than the latter. If `--groups` is also provided, then these thresholds apply to each cell **group** instead of each cell individually. Internally, `--cell-threshold` is used to filter cells before estimating the average conversion rate in labeled RNA (see *Induced rate estimation (p_c)*), and `--cell-gene-threshold` is used to filter cell-gene combinations before estimating the fraction of new RNA (see *Bayesian inference (\pi_g)*).

2.5.2 Estimation on a subset of RNA species

The `--reads` argument controls which RNA species to run the estimation procedure on. By default, all possible RNA species, minus `ambiguous` reads, are used. This argument can take on the following values: `total`, `transcriptome`, `spliced`, `unspliced` (see *Read groups*). The value of this argument specifies which group of unlabeled/labeled RNA counts will be estimated. For instance, `--reads spliced` will run statistical estimation on unlabeled/labeled spliced reads. This option may be provided multiple times to run estimation on multiple groups. The procedure involves estimating the conversion rate of unlabeled and labeled RNA, and modeling the fraction of new RNA as a binomial mixture model (see *Statistical estimation*).

2.5.3 Grouping cells

Sometimes, grouping read counts across cells may provide better estimation results, especially in the case of droplet-based methods, which result in fewer reads per cell and gene compared to plate-based methods. The `--groups` argument can be used to provide a CSV of two columns: the first containing the cell barcodes and the second containing group names that each cell belongs to. Estimation is then performed on a per-group basis by combining the read counts across all cells in each group. This strategy may be applied across different samples, simply by specifying multiple input directories. In this case, the number of group CSVs specified with `--groups` must match the number of input directories. For example, when providing two input directories `./input1` and `./input2`, with the intention of grouping cells across these two samples, two group CSVs, `groups1.csv` and `groups2.csv` must be provided where the former are groups for barcodes in the first sample, and the latter are groups for barcodes in the second sample. The group names may be shared across samples. The output AnnData will still contain reads per cell.

Cell groupings provided this way may be ignored for estimation of the fraction of labeled RNA (see *Bayesian inference (\pi_g)*) by providing the `--ignore-groups-for-pi` flag. This flag may be used only in conjunction with `--groups`, and when it is provided, estimation of the fraction of labeled RNA is performed per cell, while estimation of background and induced mutation rates are still done per group.

2.5.4 Downsampling

Downsampling UMIs uniformly, per cell, or per cell group may be useful to significantly reduce runtime while troubleshooting pipeline parameters (or just to quickly get some preliminary results). Dynast can perform downsampling when the `--downsample` argument is used. The value of this argument may either be an integer indicating the number of UMIs to retain or a proportion between 0 and 1 indicating the proportion of UMIs to retain. Additionally, the downsampling mode may be specified with the `--downsample-mode` argument, which takes one of the following three parameters: `uniform`, `cell`, `group`. `uniform` is the default that downsamples UMIs uniformly at random. When `cell` is provided, the value of `--downsample` may only be an integer specifying the threshold to downsample cells to. Only cells with UMI counts greater than this value will be downsampled to exactly this value. `group` works the same way, but for cell groups and may be used only in conjunction with `--groups`.

2.6 Control samples

Control samples may be used to find common SNPs and directly estimate the conversion rate of unlabeled RNA (see *Background estimation (p_e)*). Normally, the latter is estimating using the reads directly. However, it is possible to use a control sample (prepared in absence of the experimental introduction of conversions) to calculate this value directly. In addition, SNPs can be called in the control sample, and these called SNPs can be used when running the test sample(s) (see *Detecting and filtering SNPs* for SNP arguments). Note that SNP calling is done with `dynast count`.

A typical workflow for a control sample is the following.

```
dynast count --control --snp-threshold 0.5 [...] -o control_count --conversion TC -g GTF.
↪gtf CONTROL.bam
dynast estimate --control -o control_estimate control_count
```

Where `[...]` indicates the usual options that would be used for `dynast count` if this were not control samples. See *Basic arguments* for these options.

The `dynast count` command detects SNPs from the control sample and outputs them to the file `snps.csv` in the output directory `control_count`. The `dynast estimate` calculates the background conversion rate of unlabeled RNA to the file `p_e.csv` in the output directory `control_estimate`. These files can then be used as input when running the test sample.

```
dynast count --snp-csv control_count/snps.csv -o test_count [...] INPUT.bam
dynast estimate --p-e control_estimate/p_e.csv -o test_estimate test_count
```

The above set of commands runs quantification and estimation on the test sample using the SNPs detected from the control sample (`control_count/snps.csv`) and the background conversion rate estimated from the control sample (`control_estimate/p_e.csv`).

CHAPTER

THREE

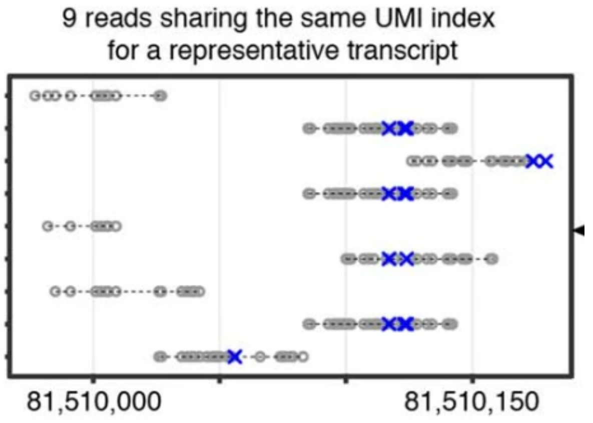
TECHNICAL INFORMATION

This section details technical information of the quantification and statistical estimation procedures of the `dynast consensus`, `dynast count` and `dynast estimate` commands. Descriptions of `dynast ref` and `dynast align` commands are in *Pipeline Usage*.

3.1 Consensus procedure

`dynast consensus` procedure generates consensus sequences for each mRNA molecule that was present in the sample. It relies on sequencing the same mRNA molecule (often distinguished using UMIs for UMI-based technologies, or start and end alignment positions for non-UMI technologies) multiple times, to obtain a more confident read sequence.

Why don't we just perform UMI-deduplication (by just selecting a single read among the reads that share the same UMI) and call it a day? Though it seems counterintuitive, reads sharing the same UMI may originate from different regions of the same mRNA, as [Qiu2020] (scNT-seq) observed in Extended Data Fig.1b.



Therefore, simply selecting one read and discarding the rest will cause a bias toward unlabeled reads because the selected read may happen to have no conversions, while all the other (discarded) reads do. Therefore, we found it necessary to implement a consensus-calling procedure, which works in the following steps. Here, we assume cell barcodes are available (`--barcode-tag` is provided), but the same procedure can be performed in the absence of cell barcodes by assuming all reads were from a single cell. Additionally, we will use the term *read* and *alignment* interchangeably because only a single alignment (see the note below) from each read will be considered.

1. Alignments in the BAM are grouped into UMI groups. In the case of UMI-based technologies (`--umi-tag` is provided), a UMI group is defined as the set of alignments that share the same cell barcode, UMI, and gene. For alignments with the `--gene-tag` tag, assigning these into a UMI group is trivial. For alignments without this tag, it is assigned to the gene whose gene body fully contains the alignment. If multiple genes are applicable, the

alignment is not assigned a UMI group and output to the resulting BAM as-is. For non-UMI-based technologies, the start and end positions of the alignment are used as a pseudo-UMI.

2. For each UMI group, the consensus base is taken for every genomic location that is covered by at least one alignment in the group. The consensus base is defined as the base with the highest sum of quality scores of that base among all alignments in the group. Loosely, this is proportional to the conditional probability of each base being a sequencing error. If the consensus base score does not exceed the score specified with `--quality`, then the reference base is taken instead. Once this is done for every covered genomic location, the consensus alignment is output to the BAM, and the UMI group is discarded (i.e. not written to the BAM).

Note: Only primary, not-duplicate, mapped BAM entries are considered (equivalent to the `0x4`, `0x100`, `0x400` BAM flags being unset). For paired reads, only properly paired alignments (`0x2` BAM flag being set) are considered. Additionally, if `--barcode-tag` or `--umi-tag` are provided, only BAM entries that have these tags are considered. Any alignments that do not satisfy all of these conditions are not written to the output BAM.

3.2 Count procedure

`dynast count` procedure consists of three steps:

1. *parse*
2. *snp*
3. *quant*

3.2.1 `parse`

1. All gene and transcript information are parsed from the gene annotation GTF (`-g`) and saved as Python pickles `genes.pkl.gz` and `transcripts.pkl.gz`, respectively.
2. All aligned reads are parsed from the input BAM and output to `conversions.csv` and `alignments.csv`. The former contains a line for every conversion, and the latter contains a line for every alignment. Note that no conversion filtering (`--quality`) is performed in this step. Two `.idx` files are also output, corresponding to each of these CSVs, which are used downstream for fast parsing. Splicing types are also assigned in this step if `--no-splicing` was not provided.

Note: Only primary, not-duplicate, mapped BAM entries are considered (equivalent to the `0x4`, `0x100`, `0x400` BAM flags being unset). For paired reads, only properly paired alignments (`0x2` BAM flag being set) are considered. Additionally, if `--barcode-tag` or `--umi-tag` are provided, only BAM entries that have these tags are considered.

3.2.2 `snp`

This step is skipped if `--snp-threshold` is not specified.

1. Read coverage of the genome is computed by parsing all aligned reads from the input BAM and output to `coverage.csv`.
2. SNPs are detected by calculating, for every genomic position, the fraction of reads with a conversion at that position over its coverage. If this fraction is greater than `--snp-threshold`, then the genomic position and the specific conversion is written to the output file `snps.csv`. Any conversion with PHRED quality less than or equal to `--quality` is not counted as a conversion. Additionally, `--snp-min-coverage` can be used to specify

the minimum coverage any detected SNP must have. Any sites that have less than this coverage are ignored (and therefore not labeled as SNPs).

3.2.3 `quant`

1. For every read, the numbers of each conversion (A>C, A>G, A>T, C>A, etc.) and nucleotide content (how many of A, C, G, T there are in the region that the read aligned to) are counted. Any SNPs provided with `--snp-csv` or detected from the *snp* step are not counted. If both are present, the union is used. Additionally, Any conversion with PHRED quality less than or equal to `--quality` is not counted as a conversion.
2. For UMI-based technologies, reads are deduplicated by the following order of priority: 1) reads that have at least one conversion specified with `--conversion`, 2) read that aligns to the transcriptome (i.e. exon-only), 3) read that has the highest alignment score, and 4) read with the most conversions specified with `--conversion`. If multiple conversions are provided, the sum is used. Reads are considered duplicates if they share the same barcode, UMI, and gene assignment. For plate-based technologies, read deduplication should have been performed in the alignment step (in the case of STAR, with the `--soloUMIdedup Exact`), but in the case of multimapping reads, it becomes a bit more tricky. If a read is multimapping such that some alignments map to the transcriptome while some do not, the transcriptome alignment is taken (there can not be multiple transcriptome alignments, as this is a constraint within STAR). If none align to the transcriptome and the alignments are assigned to multiple genes, the read is dropped, as it is impossible to assign the read with confidence. If none align to the transcriptome and the alignments are assigned multiple velocity types, the velocity type is manually set to `ambiguous` and the first alignment is kept. If none of these cases are true, the first alignment is kept. The final deduplicated/de-multimapped counts are output to `counts_{conversions}.csv`, where `{conversions}` is an underscore-delimited list of all conversions provided with `--conversion`.

Note: All bases in this file are relative to the forward genomic strand. For example, a read mapped to a gene on the reverse genomic strand should be complemented to get the actual bases.

3.2.4 Output Anndata

All results are compiled into a single AnnData `H5AD` file. The AnnData object contains the following:

- The *transcriptome* read counts in `.X`. Here, *transcriptome* reads are the mRNA read counts that are usually output from conventional scRNA-seq quantification pipelines. In technical terms, these are reads that contain the BAM tag provided with the `--gene-tag` (default is `GX`).
- Unlabeled and labeled *transcriptome* read counts in `.layers['X_n_{conversion}']` and `.layers['X_l_{conversion}']`.

The following layers are also present if `--no-splicing` or `--transcriptome-only` was *NOT* specified.

- The *total* read counts in `.layers['total']`.
- Unlabeled and labeled *total* read counts in `.layers['unlabeled_{conversion}']` and `.layers['labeled_{conversion}']`.
- Spliced, unspliced and ambiguous read counts in `.layers['spliced']`, `.layers['unspliced']` and `.layers['ambiguous']`.
- Unspliced unlabeled, unspliced labeled, spliced unlabeled, spliced labeled read counts in `.layers['un_{conversion}']`, `.layers['ul_{conversion}']`, `.layers['sn_{conversion}']` and `.layers['sl_{conversion}']` respectively.

The following equalities always hold for the resulting Anndata.

- `.layers['total'] == .layers['spliced'] + .layers['unspliced'] + .layers['ambiguous']`

The following additional equalities always hold for the resulting Anndata in the case of single labeling (`--conversion` was specified once).

- `.X == .layers['X_n_{conversion}'] + .layers['X_l_{conversion}']`
- `.layers['spliced'] == .layers['sn_{conversion}'] + .layers['sl_{conversion}']`
- `.layers['unspliced'] == .layers['un_{conversion}'] + .layers['ul_{conversion}']`

Tip: To quantify splicing data from conventional scRNA-seq experiments (experiments without metabolic labeling), we recommend using the kallisto | bustools pipeline.

3.3 Estimate procedure

`dynast estimate` procedure consists of two steps:

1. *aggregate*
2. *estimate*

3.3.1 aggregate

For each cell and gene and for each conversion provided with `--conversion`, the conversion counts are aggregated into a CSV file such that each row contains the following columns: cell barcode, gene, conversion count, nucleotide content of the original base (i.e. if the conversion is T>C, this would be T), and the number of reads that have this particular barcode-gene-conversion-content combination. This procedure is done for all read groups that exist (see *Read groups*).

3.3.2 estimate

1. The background conversion rate p_e is estimated, if `--p-e` was not provided (see *Background estimation (p_e)*). If `--p-e` was provided, this value is used and estimation is skipped. p_e.
2. The induced conversion rate p_c is estimated using an expectation maximization (EM) approach, for each conversion provided with `--conversion` (see *Induced rate estimation (p_c)*). p_c where `{conversion}` is an underscore-delimited list of each conversion (because multiple conversions can be introduced in a single timepoint). This step is skipped for control samples with `--control`.
3. Finally, the fraction of labeled RNA per cell π_c and per cell-gene π_g are estimated. The resulting fractions are written to CSV files named `pi_c_xxx.csv` and `pi_xxx.csv`, where the former contains estimations per cell and the latter contains estimations per cell-gene.

3.3.3 Output Anndata

All results are compiled into a single AnnData `H5AD` file. The AnnData object contains the following:

- The *transcriptome* read counts in `.X`. Here, *transcriptome* reads are the mRNA read counts that are usually output from conventional scRNA-seq quantification pipelines. In technical terms, these are reads that contain the BAM tag provided with the `--gene-tag` (default is `GX`).

- Unlabeled and labeled *transcriptome* read counts in `.layers['X_n_{conversion}']` and `.layers['X_l_{conversion}']`. If `--reads transcriptome` was specified, the estimated counts are in `.layers['X_n_{conversion}_est']` and `.layers['X_l_{conversion}_est']`. `{conversion}` is an underscore-delimited list of each conversion provided with `--conversion` when running `dynast count`.
- Per cell estimated parameters in corresponding columns of `.obs`. These include the estimated p_e in `.obs['p_e']`, p_c in `.obs['p_c_{conversion}']`, and per cell estimated fractions of labeled RNA in `.obs['pi_c_{group}_{conversion}']`. There is one column for each possible read group. For instance, if `transcriptome` and `spliced` read groups are available, two columns with the names `pi_c_transcriptome_{conversion}` and `pi_c_spliced_{conversion}` are added.

The following layers are also present if `--no-splicing` or `--transcriptome-only` was *NOT* specified when running `dynast count`.

- The *total* read counts in `.layers['total']`.
- Unlabeled and labeled *total* read counts in `.layers['unlabeled_{conversion}']` and `.layers['labeled_{conversion}']`. If `--reads total` is specified, the estimated counts are in `.layers['unlabeled_{conversion}_est']` and `.layers['labeled_{conversion}_est']`.
- Spliced, unspliced and ambiguous read counts in `.layers['spliced']`, `.layers['unspliced']` and `.layers['ambiguous']`.
- Unspliced unlabeled, unspliced labeled, spliced unlabeled, spliced labeled read counts in `.layers['un_{conversion}']`, `.layers['ul_{conversion}']`, `.layers['sn_{conversion}']` and `.layers['sl_{conversion}']` respectively. If `--reads spliced` and/or `--reads unspliced` was specified, layers with estimated counts are added. These layers are suffixed with `_est`, analogous to *total* counts above.

In addition to the equalities listed in the *quant* section, the following inequalities always hold for the resulting Anndata.

- `.X >= .layers['X_n_{conversion}_est'] + .layers['X_l_{conversion}_est']`
- `.layers['spliced'] >= .layers['sn_{conversion}_est'] + .layers['sl_{conversion}_est']`
- `.layers['unspliced'] >= .layers['un_{conversion}_est'] + .layers['ul_{conversion}_est']`

Tip: To quantify splicing data from conventional scRNA-seq experiments (experiments without metabolic labeling), we recommend using the kallisto | bustools pipeline.

3.3.4 Caveats

The statistical estimation procedure described above comes with some caveats.

- The induced conversion rate (p_c) can not be estimated for cells with too few reads (defined by the option `--cell-threshold`).
- The fraction of labeled RNA (π_g) can not be estimated for cell-gene combinations with too few reads (defined by the option `--cell-gene-threshold`).

For statistical definitions of these variables, see *Statistical estimation*.

Therefore, for low coverage data, we expect many cell-gene combinations to not have any estimations in the Anndata layers prefixed with `_est`, indicated with zeros. It is possible to construct a boolean mask that contains `True` for cell-gene combinations that were successfully estimated and `False` otherwise. Note that we are using *total* reads.

```
estimated_mask = ((adata.layers['labeled_{conversion}'] + adata.layers['unlabeled_
↪{conversion}']) > 0) & \
    ((adata.layers['labeled_{conversion}_est'] + adata.layers['unlabeled_{conversion}_est
↪']) > 0)
```

Similarly, it is possible to construct a boolean mask that contains `True` for cell-gene combinations for which estimation failed (either due to having too few reads mapping at the cell level or the cell-gene level) and `False` otherwise.

```
failed_mask = ((adata.layers['labeled_{conversion}'] + adata.layers['unlabeled_
↪{conversion}']) > 0) & \
    ((adata.layers['labeled_{conversion}_est'] + adata.layers['unlabeled_{conversion}_est
↪']) == 0)
```

The same can be done with other *Read groups*.

3.4 Read groups

Dynast separates reads into read groups, and each of these groups are processed together.

- `total`: All reads. Used only when `--no-splicing` or `--transcriptome-only` is not used.
- `transcriptome`: Reads that map to the transcriptome. These are reads that have the `GX` tag in the BAM (or whatever you provide for the `--gene-tag` argument). This group also represents all reads when `--no-splicing` or `--transcriptome-only` is used.
- `spliced`: Spliced reads
- `unspliced`: Unspliced reads
- `ambiguous`: Ambiguous reads

The latter three groups are mutually exclusive.

3.5 Statistical estimation

Dynast can statistically estimate unlabeled and labeled RNA counts by modeling the distribution as a binomial mixture model [Jürges2018]. Statistical estimation can be run with `dynast estimate` (see *estimate*).

3.5.1 Overview

First, we define the following model parameters. For the remainder of this section, let the conversion be T>C. Note that all parameters are calculated per barcode (i.e. cell) unless otherwise specified.

$$\begin{aligned}
p_e &: \text{average conversion rate in unlabeled RNA} \\
p_c &: \text{average conversion rate in labeled RNA} \\
\pi_g &: \text{fraction of labeled RNA for gene } g \\
y &: \text{number of observed T>C conversions (in a read)} \\
n &: \text{number of T bases in the genomic region (a read maps to)}
\end{aligned}$$

Then, the probability of observing k conversions given the above parameters is

$$\mathbb{P}(k; p_e, p_c, n, \pi) = (1-\pi_g)B(k; n, p_e) + \pi_g B(k; n, p_c)$$

where $B(k, n, p)$ is the binomial PMF. The goal is to calculate π_g, which can be used the split the raw counts to get the estimated counts. We can extract k and n directly from the read alignments, while calculating p_e and p_c is more complicated (detailed below).

3.5.2 Background estimation (p_e)

If we have control samples (i.e. samples without the conversion-introducing treatment), we can calculate p_e directly by simply calculating the mutation rate of T to C. This is exactly what dynast does for `--control` samples. All cells are aggregated when calculating p_e for control samples.

Otherwise, we need to use other mutation rates as a proxy for the real T>C background mutation rate. In this case, p_e is calculated as the average conversion rate of all non-T bases to any other base. Mathematically,

$$p_e = average(r(A, C), r(A, G), \cdots, r(G, T))$$

where $r(X, Y)$ is the observed conversion rate from X to Y, and $average$ is the function that calculates the average of its arguments. Note that we do not use the conversion rates of conversions that start with a T. This is because T>C is our induced mutation, and this artificially deflates the T>A, T>G mutation rates (which can skew our p_e estimation to be lower than it should). In the event that multiple conversions are of interest, and they span all four bases as the initial base, then p_e estimation falls back to using all other conversions (regardless of start base).

3.5.3 Induced rate estimation (p_c)

p_c is estimated via an expectation maximization (EM) algorithm by constructing a sparse matrix A where each element $a_{k,n}$ is the number of reads with k T>C conversions and n T bases in the genomic region that each read align to. Assuming $p_e < p_c$, we treat $a_{k,n}$ as missing data if greater than or equal to 1% of the count is expected to originate from the p_e component. Mathematically, $a_{k,n}$ is excluded if

$$e_{k,n} = B(k, n, p_e) \cdot \sum_{k' \geq k} a_{k',n} > 0.01 a_{k,n}$$

Let $X = \{(k_1, n_1), \cdots\}$ be the excluded data. The E step fills in the excluded data by their expected values given the current estimate $p_c^{(t)}$,

$$a_{k,n}^{(t+1)} = \frac{\sum_{(k',n) \notin X} B(k, n, p_c^{(t)}) \cdot a_{k',n}}{\sum_{(k',n) \notin X} B(k', n, p_c^{(t)})}$$

The M step updates the estimate for p_c

$$p_c^{(t+1)} = \frac{\sum_{k,n} k a_{k,n}^{(t+1)}}{\sum_{k,n} n a_{k,n}^{(t+1)}}$$

3.5.4 Bayesian inference (π_g)

The fraction of labeled RNA per cell π_c and per cell-gene π_g are estimated with Bayesian inference using the binomial mixture model described above. A Markov chain Monte Carlo (MCMC) approach is applied using the p_e, p_c, and the matrix A found/estimated in previous steps. This estimation procedure is implemented with pyStan, which is a Python interface to the Bayesian inference package Stan. The Stan model definition is here.

CHAPTER

FOUR

DYNAST

4.1 Subpackages

4.1.1 dynast.benchmarking

Submodules

dynast.benchmarking.simulation

Module Contents

Functions

generate_sequence(k, seed=None)	Generate a random genome sequence of length *k*.
simulate_reads(sequence, p_e, p_c, pi, l=100, n=100, seed=None)	Simulate *n* reads of length *l* from a sequence.
initializer(model)	
estimate(df_counts, p_e, p_c, pi, estimate_p_e=False, estimate_p_c=False, estimate_pi=True, model=None, nasc=False)	
_simulate(p_e, p_c, pi, sequence=None, k=10000, l=100, n=100, estimate_p_e=False, estimate_p_c=False, estimate_pi=True, seed=None, model=None, nasc=False)	
simulate(p_e, p_c, pi, sequence=None, k=10000, l=100, n=100, n_runs=16, n_threads=8, estimate_p_e=False, estimate_p_c=False, estimate_pi=True, model=None, nasc=False)	
simulate_batch(p_e, p_c, pi, l, n, estimate_p_e, estimate_p_c, estimate_pi, n_runs, n_threads, model, nasc=False)	Helper function to run simulations in batches.
plot_estimations(X, Y, n_runs, means, truth, ax=None, box=True, tick_decimals=1, title=None, xlabel=None, ylabel=None)	

Attributes

__model
_pi_model

`dynast.benchmarking.simulation.`**`generate_sequence`**(*k*, *seed=None*)

Generate a random genome sequence of length *k*.

Parameters

- **k** (*int*) – length of the sequence
- **seed** (*int, optional*) – random seed, defaults to *None*

Returns a random sequence

Return type str

`dynast.benchmarking.simulation.`**`simulate_reads`**(*sequence*, *p_e*, *p_c*, *pi*, *l=100*, *n=100*, *seed=None*)

Simulate *n* reads of length *l* from a sequence.

Parameters

- **sequence** (*str*) – sequence to generate the reads from
- **p_e** (*float*) – background specific mutation rate. This is the rate a specific base mutates to another specific base (i.e. T>C, A>G, . . .)
- **p_c** (*float*) – T>C mutation rate in labeled RNA
- **pi** (*float*) – fraction of labeled RNA
- **l** (*int, optional*) – length of each read, defaults to *100*
- **n** (*int, optional*) – number of reads to simulate, defaults to *100*
- **seed** (*int, optional*) – random seed, defaults to *None*

Returns a dataframe with each read as a row and the number of conversions and base content as the columns

Return type pandas.DataFrame

`dynast.benchmarking.simulation.`**`__model`**

`dynast.benchmarking.simulation.`**`_pi_model`**

`dynast.benchmarking.simulation.`**`initializer`**(*model*)

`dynast.benchmarking.simulation.`**`estimate`**(*df_counts*, *p_e*, *p_c*, *pi*, *estimate_p_e=False*, *estimate_p_c=False*, *estimate_pi=True*, *model=None*, *nasc=False*)

`dynast.benchmarking.simulation.`**`_simulate`**(*p_e*, *p_c*, *pi*, *sequence=None*, *k=10000*, *l=100*, *n=100*, *estimate_p_e=False*, *estimate_p_c=False*, *estimate_pi=True*, *seed=None*, *model=None*, *nasc=False*) `dynast.benchmarking.simulation.`**`simulate`**(*p_e*, *p_c*, *pi*, *sequence=None*, *k=10000*, *l=100*, *n=100*, *n_runs=16*, *n_threads=8*, *estimate_p_e=False*, *estimate_p_c=False*, *estimate_pi=True*, *model=None*, *nasc=False*)

`dynast.benchmarking.simulation.`**`simulate_batch`**(*p_e*, *p_c*, *pi*, *l*, *n*, *estimate_p_e*, *estimate_p_c*, *estimate_pi*, *n_runs*, *n_threads*, *model*, *nasc=False*)

Helper function to run simulations in batches.

`dynast.benchmarking.simulation.`**`plot_estimations`**(*X*, *Y*, *n_runs*, *means*, *truth*, *ax=None*, *box=True*, *tick_decimals=1*, *title=None*, *xlabel=None*, *ylabel=None*)

4.1.2 `dynast.estimation`

Submodules

`dynast.estimation.p_c`

Module Contents

Functions

read_p_c(p_c_path, group_by=None)	Read p_c CSV as a dictionary, with *group_by* columns as keys.
binomial_pmf(k, n, p)	Numbaized binomial PMF function for faster calculation.
expectation_maximization_nasc(values, p_e, threshold=0.01)	NASC-seq pipeline variant of the EM algorithm to estimate average
expectation_maximization(values, p_e, p_c=0.1, threshold=0.01, max_iters=300)	Run EM algorithm to estimate average conversion rate in labeled RNA.
estimate_p_c(df_aggregates, p_e, p_c_path, group_by=None, threshold=1000, n_threads=8, nasc=False)	Estimate the average conversion rate in labeled RNA.

`dynast.estimation.p_c.`**`read_p_c`**(*p_c_path*, *group_by=None*)

Read p_c CSV as a dictionary, with *group_by* columns as keys.

Parameters

- **p_c_path** (*str*) – path to CSV containing p_c values
- **group_by** (*list, optional*) – columns to group by, defaults to *None*

Returns dictionary with *group_by* columns as keys (tuple if multiple)

Return type dictionary

`dynast.estimation.p_c.`**`binomial_pmf`**(*k*, *n*, *p*)

Numbaized binomial PMF function for faster calculation.

Parameters

- **k** (*int*) – number of successes

- **n** (*int*) – number of trials
- **p** (*float*) – probability of success

Returns probability of observing *k* successes in *n* trials with probability of success *p*

Return type float

`dynast.estimation.p_c.`**`expectation_maximization_nasc`**(*values*, *p_e*, *threshold=0.01*)

NASC-seq pipeline variant of the EM algorithm to estimate average conversion rate in labeled RNA.

Parameters

- **values** (*numpy.ndarray*) – array of three columns encoding a sparse array in (row, column, value) format, zero-indexed, where

  row: number of conversions column: nucleotide content value: number of reads
- **p_e** (*float*) – background mutation rate of unlabeled RNA
- **threshold** (*float, optional*) – filter threshold, defaults to *0.01*

Returns estimated conversion rate

Return type float

`dynast.estimation.p_c.`**`expectation_maximization`**(*values*, *p_e*, *p_c=0.1*, *threshold=0.01*, *max_iters=300*)

Run EM algorithm to estimate average conversion rate in labeled RNA.

This function runs the following two steps. 1) Constructs a sparse matrix representation of *values* and filters out certain

indices that are expected to contain more than *threshold* proportion of unlabeled reads.

2) Runs an EM algorithm that iteratively updates the filtered out data and stimation.

See https://doi.org/10.1093/bioinformatics/bty256.

Parameters

- **values** (*numpy.ndarray*) – array of three columns encoding a sparse array in (row, column, value) format, zero-indexed, where

  row: number of conversions column: nucleotide content value: number of reads
- **p_e** (*float*) – background mutation rate of unlabeled RNA
- **p_c** (*float, optional*) – initial p_c value, defaults to *0.1*
- **threshold** (*float, optional*) – filter threshold, defaults to *0.01*
- **max_iters** (*int, optional*) – maximum number of EM iterations, defaults to *300*

Returns estimated conversion rate

Return type float

`dynast.estimation.p_c.`**`estimate_p_c`**(*df_aggregates*, *p_e*, *p_c_path*, *group_by=None*, *threshold=1000*, *n_threads=8*, *nasc=False*)

Estimate the average conversion rate in labeled RNA.

Parameters

- **df_aggregates** (*pandas.DataFrame*) – Pandas dataframe containing aggregate values
- **p_e** (*float*) – background mutation rate of unlabeled RNA

- **p_c_path** (*str*) – path to output CSV containing p_c estimates
- **group_by** (*list, optional*) – columns to group by, defaults to *None*
- **threshold** (*int, optional*) – read count threshold, defaults to *1000*
- **n_threads** (*int, optional*) – number of threads, defaults to *8*
- **nasc** (*bool, optional*) – flag to indicate whether to use NASC-seq pipeline variant of the EM algorithm, defaults to *False*

Returns path to output CSV containing p_c estimates

Return type str

`dynast.estimation.p_e`

Module Contents

Functions

read_p_e(p_e_path, group_by=None)	Read p_e CSV as a dictionary, with *group_by* columns as keys.
estimate_p_e_control(df_counts, p_e_path, conversions=frozenset([('TC',)]))	Estimate background mutation rate of unlabeled RNA for a control sample
estimate_p_e(df_counts, p_e_path, conversions=frozenset([('TC',)]), group_by=None)	Estimate background mutation rate of unabeled RNA by calculating the
estimate_p_e_nasc(df_rates, p_e_path, group_by=None)	Estimate background mutation rate of unabeled RNA by calculating the

dynast.estimation.p_e.**read_p_e**(*p_e_path*, *group_by=None*)

Read p_e CSV as a dictionary, with *group_by* columns as keys.

Parameters

- **p_e_path** (*str*) – path to CSV containing p_e values
- **group_by** (*list, optional*) – columns to group by, defaults to *None*

Returns dictionary with *group_by* columns as keys (tuple if multiple)

Return type dictionary

dynast.estimation.p_e.**estimate_p_e_control**(*df_counts*, *p_e_path*, *conversions=frozenset([('TC',)])*)

Estimate background mutation rate of unlabeled RNA for a control sample by simply calculating the average mutation rate.

Parameters

- **df_counts** (*pandas.DataFrame*) – Pandas dataframe containing number of each conversion and nucleotide content of each read
- **p_e_path** (*str*) – path to output CSV containing p_e estimates
- **conversions** (*list, optional*) – conversion(s) in question, defaults to *frozenset([('TC',)])*

Returns path to output CSV containing p_e estimates

Return type str

`dynast.estimation.p_e.`**`estimate_p_e`**(*df_counts*, *p_e_path*, *conversions=frozenset([('TC',)])*, *group_by=None*)

Estimate background mutation rate of unabeled RNA by calculating the average mutation rate of all three nucleotides other than *conversion[0]*.

Parameters

- **df_counts** (*pandas.DataFrame*) – Pandas dataframe containing number of each conversion and nucleotide content of each read
- **p_e_path** (*str*) – path to output CSV containing p_e estimates
- **conversions** (*list, optional*) – conversion(s) in question, defaults to *frozenset([('TC',)])*
- **group_by** (*list, optional*) – columns to group by, defaults to *None*

Returns path to output CSV containing p_e estimates

Return type str

`dynast.estimation.p_e.`**`estimate_p_e_nasc`**(*df_rates*, *p_e_path*, *group_by=None*)

Estimate background mutation rate of unabeled RNA by calculating the average *CT* and *GA* mutation rates. This function imitates the procedure implemented in the NASC-seq pipeline (DOI: 10.1038/s41467-019-11028-9).

Parameters

- **df_counts** (*pandas.DataFrame*) – Pandas dataframe containing number of each conversion and nucleotide content of each read
- **p_e_path** (*str*) – path to output CSV containing p_e estimates
- **group_by** (*list, optional*) – columns to group by, defaults to *None*

Returns path to output CSV containing p_e estimates

Return type str

dynast.estimation.pi

Module Contents

Functions

read_pi(pi_path, group_by=None)	Read pi CSV as a dictionary.
initializer(model)	Multiprocessing initializer.
beta_mean(alpha, beta)	Calculate the mean of a beta distribution.
beta_mode(alpha, beta)	Calculate the mode of a beta distribution.
guess_beta_parameters(guess, strength=5)	Given a *guess* of the mean of a beta distribution, calculate beta
fit_stan_mcmc(values, p_e, p_c, guess=0.5, model=None, n_chains=1, n_warmup=1000, n_iters=1000, seed=None)	Run MCMC to estimate the fraction of labeled RNA.
estimate_pi(df_aggregates, p_e, p_c, pi_path, group_by=None, p_group_by=None, n_threads=8, threshold=16, seed=None, nasc=False, model=None)	Estimate the fraction of labeled RNA.

Attributes

_model

`dynast.estimation.pi.`**`read_pi`**`(`*pi_path*, *group_by=None*`)`

Read pi CSV as a dictionary.

Parameters

- **pi_path** (*str*) – path to CSV containing pi values
- **group_by** (*list, optional*) – columns that were used to group estimation, defaults to `None`

Returns dictionary with barcodes and genes as keys

Return type dictionary

`dynast.estimation.pi.`**`_model`**

`dynast.estimation.pi.`**`initializer`**`(`*model*`)`

Multiprocessing initializer. https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.ThreadPoolExecutor

This initializer performs a one-time expensive initialization for each process.

`dynast.estimation.pi.`**`beta_mean`**`(`*alpha*, *beta*`)`

Calculate the mean of a beta distribution. https://en.wikipedia.org/wiki/Beta_distribution

Parameters

- **alpha** (*float*) – first parameter of the beta distribution
- **beta** (*float*) – second parameter of the beta distribution

Returns mean of the beta distribution

Return type float

`dynast.estimation.pi.`**`beta_mode`**`(`*alpha*, *beta*`)`

Calculate the mode of a beta distribution. https://en.wikipedia.org/wiki/Beta_distribution

When the distribution is bimodal (*alpha*, *beta* < 1), this function returns *nan*.

Parameters

- **alpha** (*float*) – first parameter of the beta distribution
- **beta** (*float*) – second parameter of the beta distribution

Returns mode of the beta distribution

Return type float

`dynast.estimation.pi.`**`guess_beta_parameters`**`(`*guess*, *strength=5*`)`

Given a *guess* of the mean of a beta distribution, calculate beta distribution parameters such that the distribution is skewed by some *strength* toward the *guess*.

Parameters

- **guess** (*float*) – guess of the mean of the beta distribution
- **strength** (*int*) – strength of the skew, defaults to *5*

Returns beta distribution parameters (alpha, beta)

Return type (float, float)

`dynast.estimation.pi.`**`fit_stan_mcmc`**(*values*, *p_e*, *p_c*, *guess=0.5*, *model=None*, *n_chains=1*, *n_warmup=1000*, *n_iters=1000*, *seed=None*)

Run MCMC to estimate the fraction of labeled RNA.

Parameters

- **`values`** (*`numpy.ndarray`*) – array of three columns encoding a sparse array in (row, column, value) format, zero-indexed, where

  row: number of conversions column: nucleotide content value: number of reads
- **`p_e`** (*`float`*) – average mutation rate in unlabeled RNA
- **`p_c`** (*`float`*) – average mutation rate in labeled RNA
- **`guess`** (*`float, optional`*) – guess for the fraction of labeled RNA, defaults to *0.5*
- **`model`** (*`pystan.StanModel, optional`*) – pyStan model to run MCMC with, defaults to *None* if not provided, will try to use the *_model* global variable
- **`n_chains`** (*`int, optional`*) – number of MCMC chains, defaults to *1*
- **`n_warmup`** (*`int, optional`*) – number of warmup iterations, defaults to *1000*
- **`n_iters`** (*`int, optional`*) – number of MCMC iterations, excluding any warmups, defaults to *1000*
- **`seed`** (*`int, optional`*) – random seed used for MCMC, defaults to *None*

Returns (guess, alpha, beta, pi)

Return type (float, float, float, float)

`dynast.estimation.pi.`**`estimate_pi`**(*df_aggregates*, *p_e*, *p_c*, *pi_path*, *group_by=None*, *p_group_by=None*, *n_threads=8*, *threshold=16*, *seed=None*, *nasc=False*, *model=None*)

Estimate the fraction of labeled RNA.

Parameters

- **`df_aggregates`** (*`pandas.DataFrame`*) – Pandas dataframe containing aggregate values
- **`p_e`** (*`float`*) – average mutation rate in unlabeled RNA
- **`p_c`** (*`float`*) – average mutation rate in labeled RNA
- **`pi_path`** (*`str`*) – path to write pi estimates
- **`group_by`** (*`list, optional`*) – columns that were used to group cells, defaults to `None`
- **`p_group_by`** (*`list, optional`*) – columns that p_e/p_c estimation was grouped by, defaults to *None*
- **`n_threads`** (*`int, optional`*) – number of threads, defaults to *8*
- **`threshold`** (*`int, optional`*) – any conversion-content pairs with fewer than this many reads will not be processed, defaults to *16*
- **`seed`** (*`int, optional`*) – random seed, defaults to *None*
- **`nasc`** (*`bool, optional`*) – flag to change behavior to match NASC-seq pipeline. Specifically, the mode of the estimated Beta distribution is used as pi, defaults to *False*

- **model** (*pystan.StanModel, optional*) – pyStan model to run MCMC with, defaults to *None* if not provided, will try to compile the module manually

Returns path to pi output

Return type str

Package Contents

Functions

estimate_p_c(df_aggregates, p_e, p_c_path, group_by=None, threshold=1000, n_threads=8, nasc=False)	Estimate the average conversion rate in labeled RNA.
read_p_c(p_c_path, group_by=None)	Read p_c CSV as a dictionary, with *group_by* columns as keys.
estimate_p_e(df_counts, p_e_path, conversions=frozenset([('TC',)]), group_by=None)	Estimate background mutation rate of unabeled RNA by calculating the
estimate_p_e_control(df_counts, p_e_path, conversions=frozenset([('TC',)]))	Estimate background mutation rate of unlabeled RNA for a control sample
estimate_p_e_nasc(df_rates, p_e_path, group_by=None)	Estimate background mutation rate of unabeled RNA by calculating the
read_p_e(p_e_path, group_by=None)	Read p_e CSV as a dictionary, with *group_by* columns as keys.
estimate_pi(df_aggregates, p_e, p_c, pi_path, group_by=None, p_group_by=None, n_threads=8, threshold=16, seed=None, nasc=False, model=None)	Estimate the fraction of labeled RNA.
read_pi(pi_path, group_by=None)	Read pi CSV as a dictionary.

dynast.estimation.**estimate_p_c**(*df_aggregates, p_e, p_c_path, group_by=None, threshold=1000, n_threads=8, nasc=False*)

Estimate the average conversion rate in labeled RNA.

Parameters

- **df_aggregates** (*pandas.DataFrame*) – Pandas dataframe containing aggregate values
- **p_e** (*float*) – background mutation rate of unlabeled RNA
- **p_c_path** (*str*) – path to output CSV containing p_c estimates
- **group_by** (*list, optional*) – columns to group by, defaults to *None*
- **threshold** (*int, optional*) – read count threshold, defaults to *1000*
- **n_threads** (*int, optional*) – number of threads, defaults to *8*
- **nasc** (*bool, optional*) – flag to indicate whether to use NASC-seq pipeline variant of the EM algorithm, defaults to *False*

Returns path to output CSV containing p_c estimates

Return type str

dynast.estimation.**read_p_c**(*p_c_path, group_by=None*)

Read p_c CSV as a dictionary, with *group_by* columns as keys.

Parameters

- **p_c_path** (`str`) – path to CSV containing p_c values
- **group_by** (`list, optional`) – columns to group by, defaults to *None*

Returns dictionary with *group_by* columns as keys (tuple if multiple)

Return type dictionary

`dynast.estimation.`**`estimate_p_e`**(*df_counts*, *p_e_path*, *conversions=frozenset([('TC',)])*, *group_by=None*)

Estimate background mutation rate of unabeled RNA by calculating the average mutation rate of all three nucleotides other than *conversion[0]*.

Parameters

- **df_counts** (`pandas.DataFrame`) – Pandas dataframe containing number of each conversion and nucleotide content of each read
- **p_e_path** (`str`) – path to output CSV containing p_e estimates
- **conversions** (`list, optional`) – conversion(s) in question, defaults to *frozenset([('TC',)])*
- **group_by** (`list, optional`) – columns to group by, defaults to *None*

Returns path to output CSV containing p_e estimates

Return type str

`dynast.estimation.`**`estimate_p_e_control`**(*df_counts*, *p_e_path*, *conversions=frozenset([('TC',)])*)

Estimate background mutation rate of unlabeled RNA for a control sample by simply calculating the average mutation rate.

Parameters

- **df_counts** (`pandas.DataFrame`) – Pandas dataframe containing number of each conversion and nucleotide content of each read
- **p_e_path** (`str`) – path to output CSV containing p_e estimates
- **conversions** (`list, optional`) – conversion(s) in question, defaults to *frozenset([('TC',)])*

Returns path to output CSV containing p_e estimates

Return type str

`dynast.estimation.`**`estimate_p_e_nasc`**(*df_rates*, *p_e_path*, *group_by=None*)

Estimate background mutation rate of unabeled RNA by calculating the average *CT* and *GA* mutation rates. This function imitates the procedure implemented in the NASC-seq pipeline (DOI: 10.1038/s41467-019-11028-9).

Parameters

- **df_counts** (`pandas.DataFrame`) – Pandas dataframe containing number of each conversion and nucleotide content of each read
- **p_e_path** (`str`) – path to output CSV containing p_e estimates
- **group_by** (`list, optional`) – columns to group by, defaults to *None*

Returns path to output CSV containing p_e estimates

Return type str

`dynast.estimation.read_p_e`(*p_e_path*, *group_by=None*)

Read p_e CSV as a dictionary, with *group_by* columns as keys.

Parameters

- **p_e_path** (*str*) – path to CSV containing p_e values
- **group_by** (*list, optional*) – columns to group by, defaults to *None*

Returns dictionary with *group_by* columns as keys (tuple if multiple)

Return type dictionary

`dynast.estimation.estimate_pi`(*df_aggregates*, *p_e*, *p_c*, *pi_path*, *group_by=None*, *p_group_by=None*, *n_threads=8*, *threshold=16*, *seed=None*, *nasc=False*, *model=None*)

Estimate the fraction of labeled RNA.

Parameters

- **df_aggregates** (*pandas.DataFrame*) – Pandas dataframe containing aggregate values
- **p_e** (*float*) – average mutation rate in unlabeled RNA
- **p_c** (*float*) – average mutation rate in labeled RNA
- **pi_path** (*str*) – path to write pi estimates
- **group_by** (*list, optional*) – columns that were used to group cells, defaults to `None`
- **p_group_by** (*list, optional*) – columns that p_e/p_c estimation was grouped by, defaults to *None*
- **n_threads** (*int, optional*) – number of threads, defaults to *8*
- **threshold** (*int, optional*) – any conversion-content pairs with fewer than this many reads will not be processed, defaults to *16*
- **seed** (*int, optional*) – random seed, defaults to *None*
- **nasc** (*bool, optional*) – flag to change behavior to match NASC-seq pipeline. Specifically, the mode of the estimated Beta distribution is used as pi, defaults to *False*
- **model** (*pystan.StanModel, optional*) – pyStan model to run MCMC with, defaults to *None* if not provided, will try to compile the module manually

Returns path to pi output

Return type str

`dynast.estimation.read_pi`(*pi_path*, *group_by=None*)

Read pi CSV as a dictionary.

Parameters

- **pi_path** (*str*) – path to CSV containing pi values
- **group_by** (*list, optional*) – columns that were used to group estimation, defaults to `None`

Returns dictionary with barcodes and genes as keys

Return type dictionary

4.1.3 `dynast.preprocessing`

Submodules

`dynast.preprocessing.aggregation`

Module Contents

Functions

read_rates(rates_path)	Read mutation rates CSV as a pandas dataframe.
read_aggregates(aggregates_path)	Read aggregates CSV as a pandas dataframe.
merge_aggregates(*dfs)	Merge multiple aggregate dataframes into one.
calculate_mutation_rates(df_counts, rates_path, group_by=None)	Calculate mutation rate for each pair of bases.
aggregate_counts(df_counts, aggregates_path, conversions=frozenset([('TC',)]))	Aggregate conversion counts for each pair of bases.

`dynast.preprocessing.aggregation.`**`read_rates`**(*rates_path*)

Read mutation rates CSV as a pandas dataframe.

Parameters **`rates_path`** (*str*) – path to rates CSV

Returns rates dataframe

Return type pandas.DataFrame

`dynast.preprocessing.aggregation.`**`read_aggregates`**(*aggregates_path*)

Read aggregates CSV as a pandas dataframe.

Parameters **`aggregates_path`** (*str*) – path to aggregates CSV

Returns aggregates dataframe

Return type pandas.DataFrame

`dynast.preprocessing.aggregation.`**`merge_aggregates`**(**dfs*)

Merge multiple aggregate dataframes into one.

Parameters **`*dfs`** – dataframes to merge

Returns merged dataframe

Return type pandas.DataFrame

`dynast.preprocessing.aggregation.`**`calculate_mutation_rates`**(*df_counts*, *rates_path*, *group_by=None*)

Calculate mutation rate for each pair of bases.

Parameters

- **`df_counts`** (*pandas.DataFrame*) – counts dataframe, with complemented reverse strand bases
- **`rates_path`** (*str*) – path to write rates CSV
- **`group_by`** (*list*) – column(s) to group calculations by, defaults to *None*, which combines all rows

Returns path to rates CSV

Return type str

`dynast.preprocessing.aggregation.`**`aggregate_counts`**(*df_counts*, *aggregates_path*, *conversions=frozenset([('TC',)])*)

Aggregate conversion counts for each pair of bases.

Parameters

- **`df_counts`** (*`pandas.DataFrame`*) – counts dataframe, with complemented reverse strand bases
- **`aggregates_path`** (*`str`*) – path to write aggregate CSV
- **`conversions`** (*`list, optional`*) – conversion(s) in question, defaults to *frozenset([('TC',)])*

Returns path to aggregate CSV that was written

Return type str

`dynast.preprocessing.bam`

Module Contents

Functions

read_alignments(alignments_path, *args, **kwargs)	Read alignments CSV as a pandas DataFrame.
read_conversions(conversions_path, *args, **kwargs)	Read conversions CSV as a pandas DataFrame.
select_alignments(df_alignments)	Select alignments among duplicates. This function performs preliminary
parse_read_contig(counter, lock, bam_path, contig, gene_infos=None, transcript_infos=None, strand='forward', umi_tag=None, barcode_tag=None, gene_tag='GX', barcodes=None, temp_dir=None, update_every=2000, nasc=False, velocity=True, strict_exon_overlap=False)	Parse all reads mapped to a contig, outputing conversion
get_tags_from_bam(bam_path, n_reads=100000, n_threads=8)	Utility function to retrieve all read tags present in a BAM.
check_bam_tags_exist(bam_path, tags, n_reads=100000, n_threads=8)	Utility function to check if BAM tags exists in a BAM within the first
check_bam_is_paired(bam_path, n_reads=100000, n_threads=8)	Utility function to check if BAM has paired reads.
check_bam_contains_secondary(bam_path, n_reads=100000, n_threads=8)	
check_bam_contains_unmapped(bam_path)	
check_bam_contains_duplicate(bam_path, n_reads=100000, n_threads=8)	
sort_and_index_bam(bam_path, out_path, n_threads=8, temp_dir=None)	Sort and index BAM.
split_bam(bam_path, n, n_threads=8, temp_dir=None)	Split BAM into n parts.
parse_all_reads(bam_path, conversions_path, alignments_path, index_path, gene_infos, transcript_infos, strand='forward', umi_tag=None, barcode_tag=None, gene_tag='GX', barcodes=None, n_threads=8, temp_dir=None, nasc=False, control=False, velocity=True, strict_exon_overlap=False, return_splits=False)	Parse all reads in a BAM and extract conversion, content and alignment

Attributes

CONVERSION_CSV_COLUMNS
ALIGNMENT_COLUMNS

`dynast.preprocessing.bam.`**`CONVERSION_CSV_COLUMNS = ['read_id', 'index', 'contig', 'genome_i', 'conversion', 'quality']`**

`dynast.preprocessing.bam.`**`ALIGNMENT_COLUMNS = ['read_id', 'index', 'barcode', 'umi', 'GX', 'A', 'C', 'G', 'T', 'velocity', 'transcriptome', 'score']`** `dynast.preprocessing.bam.`**`read_alignments`**(*alignments_path*, **args*, ***kwargs*)

Read alignments CSV as a pandas DataFrame.

Any additional arguments and keyword arguments are passed to *pandas.read_csv*.

Parameters **`alignments_path`** (*`str`*) – path to alignments CSV

Returns conversions dataframe

Return type pandas.DataFrame

`dynast.preprocessing.bam.`**`read_conversions`**(*conversions_path*, **args*, ***kwargs*)

Read conversions CSV as a pandas DataFrame.

Any additional arguments and keyword arguments are passed to *pandas.read_csv*.

Parameters **`conversions_path`** (*`str`*) – path to conversions CSV

Returns conversions dataframe

Return type pandas.DataFrame

`dynast.preprocessing.bam.`**`select_alignments`**(*df_alignments*)

Select alignments among duplicates. This function performs preliminary deduplication and returns a list of tuples (read_id, alignment index) to use for coverage calculation and SNP detection.

Parameters **`df_alignments`** (*`pandas.DataFrame`*) – alignments dataframe

Returns set of (read_id, alignment index) that were selected

Return type set

`dynast.preprocessing.bam.`**`parse_read_contig`**(*counter*, *lock*, *bam_path*, *contig*, *gene_infos=None*, *transcript_infos=None*, *strand='forward'*, *umi_tag=None*, *barcode_tag=None*, *gene_tag='GX'*, *barcodes=None*, *temp_dir=None*, *update_every=2000*, *nasc=False*, *velocity=True*, *strict_exon_overlap=False*)

Parse all reads mapped to a contig, outputing conversion information as temporary CSVs. This function is designed to be called as a separate process.

Parameters

- **`counter`** (*`multiprocessing.Value`*) – counter that keeps track of how many reads have been processed
- **`lock`** (*`multiprocessing.Lock`*) – semaphore for the *counter* so that multiple processes do not modify it at the same time
- **`bam_path`** (*`str`*) – path to alignment BAM file
- **`contig`** (*`str`*) – only reads that map to this contig will be processed
- **`gene_infos`** (*`dictionary`*) – dictionary containing gene information, as returned by *preprocessing.gtf.parse_gtf*, required if *velocity=True*, defaults to *None*
- **`transcript_infos`** (*`dictionary`*) – dictionary containing transcript information, as returned by *preprocessing.gtf.parse_gtf*, required if *velocity=True*, defaults to *None*
- **`strand`** (*`str, optional`*) – strandedness of the sequencing protocol, defaults to *forward*, may be one of the following: *forward*, *reverse*, *None* (unstranded)
- **`umi_tag`** (*`str, optional`*) – BAM tag that encodes UMI, if not provided, *NA* is output in the *umi* column, defaults to *None*

- **barcode_tag** (*str, optional*) – BAM tag that encodes cell barcode, if not provided, *NA* is output in the *barcode* column, defaults to *None*
- **gene_tag** (*str, optional*) – BAM tag that encodes gene assignment, defaults to *GX*
- **barcodes** (*list, optional*) – list of barcodes to be considered. All barcodes are considered if not provided, defaults to *None*
- **temp_dir** (*str, optional*) – path to temporary directory, defaults to *None*
- **update_every** (*int, optional*) – update the counter every this many reads, defaults to *5000*
- **nasc** (*bool, optional*) – flag to change behavior to match NASC-seq pipeline, defaults to *False*
- **velocity** (*bool, optional*) – whether or not to assign a velocity type to each read, defaults to *True*
- **strict_exon_overlap** (*bool, optional*) – Whether to use a stricter algorithm to assin reads as spliced, defaults to *False*

Returns (path to conversions, path to conversions index, path to alignments)

Return type (str, str, str)

`dynast.preprocessing.bam.`**`get_tags_from_bam`**(*bam_path*, *n_reads=100000*, *n_threads=8*)

Utility function to retrieve all read tags present in a BAM.

Parameters

- **bam_path** (*str*) – path to BAM
- **n_reads** (*int, optional*) – number of reads to consider, defaults to *100000*
- **n_threads** (*int, optional*) – number of threads, defaults to *8*

Returns set of all tags found

Return type set

`dynast.preprocessing.bam.`**`check_bam_tags_exist`**(*bam_path*, *tags*, *n_reads=100000*, *n_threads=8*)

Utility function to check if BAM tags exists in a BAM within the first *n_reads* reads.

Parameters

- **bam_path** (*str*) – path to BAM
- **tags** (*list*) – tags to check for
- **n_reads** (*int, optional*) – number of reads to consider, defaults to *100000*
- **n_threads** (*int, optional*) – number of threads, defaults to *8*

Returns (whether all tags were found, list of not found tags)

Return type (bool, list)

`dynast.preprocessing.bam.`**`check_bam_is_paired`**(*bam_path*, *n_reads=100000*, *n_threads=8*)

Utility function to check if BAM has paired reads.

Parameters

- **bam_path** (*str*) – path to BAM
- **n_reads** (*int, optional*) – number of reads to consider, defaults to *100000*
- **n_threads** (*int, optional*) – number of threads, defaults to *8*

Returns whether paired reads were detected

Return type bool

`dynast.preprocessing.bam.`**`check_bam_contains_secondary`**(*bam_path*, *n_reads=100000*, *n_threads=8*)

`dynast.preprocessing.bam.`**`check_bam_contains_unmapped`**(*bam_path*)

`dynast.preprocessing.bam.`**`check_bam_contains_duplicate`**(*bam_path*, *n_reads=100000*, *n_threads=8*)

`dynast.preprocessing.bam.`**`sort_and_index_bam`**(*bam_path*, *out_path*, *n_threads=8*, *temp_dir=None*)

Sort and index BAM.

If the BAM is already sorted, the sorting step is skipped.

Parameters

- **bam_path** (`str`) – path to alignment BAM file to sort
- **out_path** (`str`) – path to output sorted BAM
- **n_threads** (`int, optional`) – number of threads, defaults to *8*
- **temp_dir** (`str, optional`) – path to temporary directory, defaults to *None*

Returns path to sorted and indexed BAM

Return type str

`dynast.preprocessing.bam.`**`split_bam`**(*bam_path*, *n*, *n_threads=8*, *temp_dir=None*)

Split BAM into n parts.

Parameters

- **bam_path** (`str`) – path to alignment BAM file
- **n** (`int`) – number of splits
- **n_threads** (`int, optional`) – number of threads, defaults to *8*
- **temp_dir** (`str, optional`) – path to temporary directory, defaults to *None*

Returns List of tuples containing (split BAM path, number of reads)

Return type list

`dynast.preprocessing.bam.`**`parse_all_reads`**(*bam_path*, *conversions_path*, *alignments_path*, *index_path*, *gene_infos*, *transcript_infos*, *strand='forward'*, *umi_tag=None*, *barcode_tag=None*, *gene_tag='GX'*, *barcodes=None*, *n_threads=8*, *temp_dir=None*, *nasc=False*, *control=False*, *velocity=True*, *strict_exon_overlap=False*, *return_splits=False*)

Parse all reads in a BAM and extract conversion, content and alignment information as CSVs.

Parameters

- **bam_path** (`str`) – path to alignment BAM file
- **conversions_path** (`str`) – path to output information about reads that have conversions
- **alignments_path** (`str`) – path to alignments information about reads
- **index_path** (`str`) – path to conversions index
- **no_index_path** (`str`) – path to no conversions index

- **gene_infos** (*dictionary*) – dictionary containing gene information, as returned by *ngs.gtf.genes_and_transcripts_from_gtf*
- **transcript_infos** (*dictionary*) – dictionary containing transcript information, as returned by *ngs.gtf.genes_and_transcripts_from_gtf*
- **strand** (*str, optional*) – strandedness of the sequencing protocol, defaults to *forward*, may be one of the following: *forward*, *reverse*, *unstranded*
- **umi_tag** (*str, optional*) – BAM tag that encodes UMI, if not provided, *NA* is output in the *umi* column, defaults to *None*
- **barcode_tag** (*str, optional*) – BAM tag that encodes cell barcode, if not provided, *NA* is output in the *barcode* column, defaults to *None*
- **gene_tag** (*str, optional*) – BAM tag that encodes gene assignment, defaults to *GX*
- **barcodes** (*list, optional*) – list of barcodes to be considered. All barcodes are considered if not provided, defaults to *None*
- **n_threads** (*int, optional*) – number of threads, defaults to *8*
- **temp_dir** (*str, optional*) – path to temporary directory, defaults to *None*
- **nasc** (*bool, optional*) – flag to change behavior to match NASC-seq pipeline, defaults to *False*
- **velocity** (*bool, optional*) – whether or not to assign a velocity type to each read, defaults to *True*
- **strict_exon_overlap** (*bool, optional*) – Whether to use a stricter algorithm to assin reads as spliced, defaults to *False*
- **return_splits** (*bool, optional*) – return BAM splits for later reuse, defaults to *True*

Returns (path to conversions, path to alignments, path to conversions index) If *return_splits* is True, then there is an additional return value, which is a list of tuples containing split BAM paths and number of reads in each BAM.

Return type (str, str, str) or (str, str, str, list)

dynast.preprocessing.consensus

Module Contents

Functions

call_consensus_from_reads(reads, header, quality=27, tags=None)	Call a single consensus alignment given a list of aligned reads.
call_consensus_from_reads_process(reads, header, tags, strand=None, quality=27)	
consensus_worker(args_q, results_q, *args, **kwargs)	
call_consensus(bam_path, out_path, gene_infos, strand='forward', umi_tag=None, barcode_tag=None, gene_tag='GX', barcodes=None, quality=27, add_RS_RI=False, temp_dir=None, n_threads=8)	

Attributes

BASES
BASE_IDX

dynast.preprocessing.consensus.**BASES = ['A', 'C', 'G', 'T']**

dynast.preprocessing.consensus.**BASE_IDX**

dynast.preprocessing.consensus.**call_consensus_from_reads**(*reads*, *header*, *quality=27*, *tags=None*)

Call a single consensus alignment given a list of aligned reads.

Reads must map to the same contig. Results are undefined otherwise. Additionally, consensus bases are called only for positions that match to the reference (i.e. no insertions allowed).

This function only sets the minimal amount of attributes such that the alignment is valid. These include: * read name – SHA256 hash of the provided read names * read sequence and qualities * reference name and ID * reference start * mapping quality (MAPQ) * cigarstring * MD tag * NM tag * Not unmapped, paired, duplicate, qc fail, secondary, nor supplementary

The caller is expected to further populate the alignment with additional tags, flags, and name.

Parameters

- **reads** (*list*) – List of reads to call a consensus sequence from
- **header** (*pysam.AlignmentHeader*) – header to use when creating the new pysam alignment
- **quality** (*int, optional*) – quality threshold, defaults to 27
- **tags** (*dict, optional*) – additional tags to set, defaults to *None*

Returns (New pysam alignment of the consensus sequence)

Return type pysam.AlignedSegment

dynast.preprocessing.consensus.**call_consensus_from_reads_process**(*reads*, *header*, *tags*, *strand=None*, *quality=27*)

dynast.preprocessing.consensus.**consensus_worker**(*args_q*, *results_q*, **args*, ***kwargs*)

dynast.preprocessing.consensus.**call_consensus**(*bam_path*, *out_path*, *gene_infos*, *strand='forward'*, *umi_tag=None*, *barcode_tag=None*, *gene_tag='GX'*, *barcodes=None*, *quality=27*, *add_RS_RI=False*, *temp_dir=None*, *n_threads=8*)

dynast.preprocessing.conversion

Module Contents

Functions

read_counts(counts_path, *args, **kwargs)	Read counts CSV as a pandas dataframe.
complement_counts(df_counts, gene_infos)	Complement the counts in the counts dataframe according to gene strand.
drop_multimappers(df_counts, conversions=None)	Drop multimappings that have the same read ID where
deduplicate_counts(df_counts, conversions=None, use_conversions=True)	Deduplicate counts based on barcode, UMI, and gene.
drop_multimappers_part(counter, lock, split_path, out_path)	
deduplicate_counts_part(counter, lock, split_path, out_path, conversions=None, use_conversions=True)	
split_counts_by_velocity(df_counts)	Split the given counts dataframe by the *velocity* column.
count_no_conversions(alignments_path, counter, lock, index, barcodes=None, temp_dir=None, update_every=10000)	Count reads that have no conversion.
count_conversions_part(conversions_path, alignments_path, counter, lock, index, barcodes=None, snps=None, quality=27, temp_dir=None, update_every=10000)	Count the number of conversions of each read per barcode and gene, along with
count_conversions(conversions_path, alignments_path, index_path, counts_path, gene_infos, barcodes=None, snps=None, quality=27, conversions=None, dedup_use_conversions=True, n_threads=8, temp_dir=None)	Count the number of conversions of each read per barcode and gene, along with

Attributes

CONVERSIONS_PARSER	
ALIGNMENTS_PARSER	
CONVERSION_IDX	
BASE_IDX	
CONVERSION_COMPLEMENT	
CONVERSION_COLUMNS	
BASE_COLUMNS	
COLUMNS	
CSV_COLUMNS	

dynast.preprocessing.conversion.**CONVERSIONS_PARSER**

dynast.preprocessing.conversion.**ALIGNMENTS_PARSER**

dynast.preprocessing.conversion.**CONVERSION_IDX**

dynast.preprocessing.conversion.**BASE_IDX**

dynast.preprocessing.conversion.**CONVERSION_COMPLEMENT**

dynast.preprocessing.conversion.**CONVERSION_COLUMNS**

dynast.preprocessing.conversion.**BASE_COLUMNS**

dynast.preprocessing.conversion.**COLUMNS**

dynast.preprocessing.conversion.**CSV_COLUMNS**

dynast.preprocessing.conversion.**read_counts**(*counts_path*, **args*, ***kwargs*)

Read counts CSV as a pandas dataframe.

Any additional arguments and keyword arguments are passed to *pandas.read_csv*.

Parameters **counts_path** (*str*) – path to CSV

Returns counts dataframe

Return type pandas.DataFrame

dynast.preprocessing.conversion.**complement_counts**(*df_counts*, *gene_infos*)

Complement the counts in the counts dataframe according to gene strand.

Parameters

- **df_counts** (*pandas.DataFrame*) – counts dataframe
- **gene_infos** (*dictionary*) – dictionary containing gene information, as returned by *preprocessing.gtf.parse_gtf*

Returns counts dataframe with counts complemented for reads mapping to genes on the reverse strand

Return type pandas.DataFrame

dynast.preprocessing.conversion.**drop_multimappers**(*df_counts*, *conversions=None*)

Drop multimappings that have the same read ID where * some map to the transcriptome while some do not – drop non-transcriptome alignments * none map to the transcriptome AND aligned to multiple genes – drop all * none map to the transcriptome AND assigned multiple velocity types – set to ambiguous

TODO: This function can probably be removed because BAM parsing only considers primary alignments now.

Parameters

- **df_counts** (*pandas.DataFrame*) – counts dataframe
- **conversions** (*list, optional*) – conversions to prioritize, defaults to *None*

Returns counts dataframe with multimappers appropriately filtered

Return type pandas.DataFrame

`dynast.preprocessing.conversion.`**`deduplicate_counts`**(*df_counts*, *conversions=None*, *use_conversions=True*)

Deduplicate counts based on barcode, UMI, and gene.

The order of priority is the following. 1. If *use_conversions=True*, reads that have at least one such conversion 2. Reads that align to the transcriptome (exon only) 3. Reads that have highest alignment score 4. If *conversions* is provided, reads that have a larger sum of such conversions

If *conversions* is not provided, reads that have larger sum of all conversions

Parameters

- **df_counts** (*pandas.DataFrame*) – counts dataframe
- **conversions** (*list, optional*) – conversions to prioritize, defaults to *None*
- **use_conversions** (*bool, optional*) – prioritize reads that have conversions first, defaults to *True*

Returns deduplicated counts dataframe

Return type pandas.DataFrame

`dynast.preprocessing.conversion.`**`drop_multimappers_part`**(*counter*, *lock*, *split_path*, *out_path*)

`dynast.preprocessing.conversion.`**`deduplicate_counts_part`**(*counter*, *lock*, *split_path*, *out_path*, *conversions=None*, *use_conversions=True*)

`dynast.preprocessing.conversion.`**`split_counts_by_velocity`**(*df_counts*)

Split the given counts dataframe by the *velocity* column.

Parameters **df_counts** (*pandas.DataFrame*) – counts dataframe

Returns dictionary containing *velocity* column values as keys and the subset dataframe as values

Return type dictionary

`dynast.preprocessing.conversion.`**`count_no_conversions`**(*alignments_path*, *counter*, *lock*, *index*, *barcodes=None*, *temp_dir=None*, *update_every=10000*)

Count reads that have no conversion.

Parameters

- **alignments_path** (*str*) – alignments CSV path
- **counter** (*multiprocessing.Value*) – counter that keeps track of how many reads have been processed
- **lock** (*multiprocessing.Lock*) – semaphore for the *counter* so that multiple processes do not modify it at the same time
- **index** (*list*) – index for conversions CSV
- **barcodes** (*list, optional*) – list of barcodes to be considered. All barcodes are considered if not provided, defaults to *None*
- **temp_dir** (*str, optional*) – path to temporary directory, defaults to *None*
- **update_every** (*int, optional*) – update the counter every this many reads, defaults to *5000*

Returns path to temporary counts CSV

Return type str

`dynast.preprocessing.conversion.`**`count_conversions_part`**(*conversions_path*, *alignments_path*, *counter*, *lock*, *index*, *barcodes=None*, *snps=None*, *quality=27*, *temp_dir=None*, *update_every=10000*)

Count the number of conversions of each read per barcode and gene, along with the total nucleotide content of the region each read mapped to, also per barcode and gene. This function is used exclusively for multiprocessing.

Parameters

- **conversions_path** (*str*) – path to conversions CSV
- **alignments_path** (*str*) – path to alignments information about reads
- **counter** (*multiprocessing.Value*) – counter that keeps track of how many reads have been processed
- **lock** (*multiprocessing.Lock*) – semaphore for the *counter* so that multiple processes do not modify it at the same time
- **index** (*list*) – list of (file position, number of lines) tuples to process
- **barcodes** (*list, optional*) – list of barcodes to be considered. All barcodes are considered if not provided, defaults to *None*
- **snps** (*dictionary, optional*) – dictionary of contig as keys and list of genomic positions as values that indicate SNP locations, defaults to *None*
- **quality** (*int, optional*) – only count conversions with PHRED quality greater than this value, defaults to *27*
- **temp_dir** (*str, optional*) – path to temporary directory, defaults to *None*
- **update_every** (*int, optional*) – update the counter every this many reads, defaults to *10000*

Returns path to temporary counts CSV

Return type tuple

`dynast.preprocessing.conversion.`**`count_conversions`**(*conversions_path*, *alignments_path*, *index_path*, *counts_path*, *gene_infos*, *barcodes=None*, *snps=None*, *quality=27*, *conversions=None*, *dedup_use_conversions=True*, *n_threads=8*, *temp_dir=None*)

Count the number of conversions of each read per barcode and gene, along with the total nucleotide content of the region each read mapped to, also per barcode. When a duplicate UMI for a barcode is observed, the read with the greatest number of conversions is selected.

Parameters

- **conversions_path** (*str*) – path to conversions CSV
- **alignments_path** (*str*) – path to alignments information about reads
- **index_path** (*str*) – path to conversions index
- **counts_path** – path to write counts CSV
- **counts_path** – str
- **gene_infos** (*dictionary*) – dictionary containing gene information, as returned by *ngs.gtf.genes_and_transcripts_from_gtf*, defaults to *None*

- **barcodes** (*list, optional*) – list of barcodes to be considered. All barcodes are considered if not provided, defaults to *None*
- **snps** (*dictionary, optional*) – dictionary of contig as keys and list of genomic positions as values that indicate SNP locations, defaults to *None*
- **conversions** (*list, optional*) – conversions to prioritize when deduplicating only applicable for UMI technologies, defaults to *None*
- **dedup_use_conversions** (*bool, optional*) – prioritize reads that have at least one conversion when deduplicating, defaults to *True*
- **quality** (*int, optional*) – only count conversions with PHRED quality greater than this value, defaults to *27*
- **n_threads** (*int, optional*) – number of threads, defaults to *8*
- **temp_dir** (*str, optional*) – path to temporary directory, defaults to *None*

Returns path to counts CSV

Return type str

`dynast.preprocessing.coverage`

Module Contents

Functions

read_coverage(coverage_path)	Read coverage CSV as a dictionary.
calculate_coverage_contig(counter, lock, bam_path, contig, indices, alignments=None, umi_tag=None, barcode_tag=None, gene_tag='GX', barcodes=None, temp_dir=None, update_every=50000, velocity=True)	Calculate converage for a specific contig. This function is designed to
calculate_coverage(bam_path, conversions, coverage_path, alignments=None, umi_tag=None, barcode_tag=None, gene_tag='GX', barcodes=None, temp_dir=None, velocity=True)	Calculate coverage of each genomic position per barcode.

Attributes

COVERAGE_PARSER

`dynast.preprocessing.coverage.`**COVERAGE_PARSER**

`dynast.preprocessing.coverage.`**read_coverage**(*coverage_path*)

Read coverage CSV as a dictionary.

Parameters **coverage_path** (*str*) – path to coverage CSV

Returns coverage as a nested dictionary

Return type dict

dynast.preprocessing.coverage.**calculate_coverage_contig**(*counter*, *lock*, *bam_path*, *contig*, *indices*, *alignments=None*, *umi_tag=None*, *barcode_tag=None*, *gene_tag='GX'*, *barcodes=None*, *temp_dir=None*, *update_every=50000*, *velocity=True*)

Calculate converage for a specific contig. This function is designed to be called as a separate process.

Parameters

- **counter** (*multiprocessing.Value*) – counter that keeps track of how many reads have been processed
- **lock** (*multiprocessing.Lock*) – semaphore for the *counter* so that multiple processes do not modify it at the same time
- **bam_path** (*str*) – path to alignment BAM file
- **contig** (*str*) – only reads that map to this contig will be processed
- **indices** (*list*) – genomic positions to consider
- **alignments** (*set, optional*) – set of (read_id, alignment_index) tuples to process. All alignments are processed if this option is not provided.
- **umi_tag** (*str, optional*) – BAM tag that encodes UMI, if not provided, *NA* is output in the *umi* column, defaults to *None*
- **barcode_tag** (*str, optional*) – BAM tag that encodes cell barcode, if not provided, *NA* is output in the *barcode* column, defaults to *None*
- **gene_tag** (*str, optional*) – BAM tag that encodes gene assignment, defaults to *GX*
- **barcodes** (*list, optional*) – list of barcodes to be considered. All barcodes are considered if not provided, defaults to *None*
- **temp_dir** (*str, optional*) – path to temporary directory, defaults to *None*
- **update_every** (*int, optional*) – update the counter every this many reads, defaults to *30000*
- **velocity** (*bool, optional*) – whether or not velocities were assigned

Returns coverag

Return type dict

dynast.preprocessing.coverage.**calculate_coverage**(*bam_path*, *conversions*, *coverage_path*, *alignments=None*, *umi_tag=None*, *barcode_tag=None*, *gene_tag='GX'*, *barcodes=None*, *temp_dir=None*, *velocity=True*)

Calculate coverage of each genomic position per barcode.

Parameters

- **bam_path** (*str*) – path to alignment BAM file
- **conversions** (*dictionary*) – dictionary of contigs as keys and sets of genomic positions as values that indicates positions where conversions were observed
- **coverage_path** (*str*) – path to write coverage CSV
- **alignments** (*set, optional*) – set of (read_id, alignment_index) tuples to process. All alignments are processed if this option is not provided.

- **umi_tag** (*str, optional*) – BAM tag that encodes UMI, if not provided, *NA* is output in the *umi* column, defaults to *None*
- **barcode_tag** (*str, optional*) – BAM tag that encodes cell barcode, if not provided, *NA* is output in the *barcode* column, defaults to *None*
- **gene_tag** (*str, optional*) – BAM tag that encodes gene assignment, defaults to *GX*
- **barcodes** (*list, optional*) – list of barcodes to be considered. All barcodes are considered if not provided, defaults to *None*
- **temp_dir** (*str, optional*) – path to temporary directory, defaults to *None*
- **velocity** (*bool, optional*) – whether or not velocities were assigned

Returns coverage CSV path

Return type str

dynast.preprocessing.snp

Module Contents

Functions

read_snps(snps_path)	Read SNPs CSV as a dictionary
read_snp_csv(snp_csv)	Read a user-provided SNPs CSV
extract_conversions_part(conversions_path, counter, lock, index, alignments=None, conversions=None, quality=27, update_every=5000)	Extract number of conversions for every genomic position.
extract_conversions(conversions_path, index_path, alignments=None, conversions=None, quality=27, n_threads=8)	Wrapper around *extract_conversions_part* that works in parallel
detect_snps(conversions_path, index_path, coverage, snps_path, alignments=None, conversions=None, quality=27, threshold=0.5, min_coverage=1, n_threads=8)	Detect SNPs.

Attributes

SNP_COLUMNS	

dynast.preprocessing.snp.**SNP_COLUMNS = ['contig', 'genome_i', 'conversion']**

dynast.preprocessing.snp.**read_snps**(*snps_path*)

Read SNPs CSV as a dictionary

Parameters **snps_path** (*str*) – path to SNPs CSV

Returns dictionary of contigs as keys and sets of genomic positions with SNPs as values

Return type dictionary

`dynast.preprocessing.snp.`**`read_snp_csv`**(*snp_csv*)

Read a user-provided SNPs CSV

Parameters **`snp_csv`** (*`str`*) – path to SNPs CSV

Returns dictionary of contigs as keys and sets of genomic positions with SNPs as values

Return type dictionary

`dynast.preprocessing.snp.`**`extract_conversions_part`**(*conversions_path*, *counter*, *lock*, *index*, *alignments=None*, *conversions=None*, *quality=27*, *update_every=5000*)

Extract number of conversions for every genomic position.

Parameters

- **`conversions_path`** (*`str`*) – path to conversions CSV
- **`counter`** (*`multiprocessing.Value`*) – counter that keeps track of how many reads have been processed
- **`lock`** (*`multiprocessing.Lock`*) – semaphore for the *counter* so that multiple processes do not modify it at the same time
- **`index`** (*`list`*) – list of (file position, number of lines) tuples to process
- **`alignments`** (*`set, optional`*) – set of (read_id, alignment_index) tuples to process. All alignments are processed if this option is not provided.
- **`conversions`** (*`set, optional`*) – set of conversions to consider
- **`quality`** (*`int, optional`*) – only count conversions with PHRED quality greater than this value, defaults to *27*
- **`update_every`** (*`int, optional`*) – update the counter every this many reads, defaults to *5000*

Returns nested dictionary that contains number of conversions for each contig and position

Return type dictionary

`dynast.preprocessing.snp.`**`extract_conversions`**(*conversions_path*, *index_path*, *alignments=None*, *conversions=None*, *quality=27*, *n_threads=8*)

Wrapper around *extract_conversions_part* that works in parallel

Parameters

- **`conversions_path`** (*`str`*) – path to conversions CSV
- **`index_path`** (*`str`*) – path to conversions index
- **`alignments`** (*`set, optional`*) – set of (read_id, alignment_index) tuples to process. All alignments are processed if this option is not provided.
- **`conversions`** (*`set, optional`*) – set of conversions to consider
- **`quality`** (*`int, optional`*) – only count conversions with PHRED quality greater than this value, defaults to *27*
- **`n_threads`** (*`int, optional`*) – number of threads, defaults to *8*

Returns nested dictionary that contains number of conversions for each contig and position

Return type dictionary

`dynast.preprocessing.snp.`**`detect_snps`**(*conversions_path*, *index_path*, *coverage*, *snps_path*, *alignments=None*, *conversions=None*, *quality=27*, *threshold=0.5*, *min_coverage=1*, *n_threads=8*)

Detect SNPs.

Parameters

- **`conversions_path`** (*`str`*) – path to conversions CSV
- **`index_path`** (*`str`*) – path to conversions index
- **`coverage`** (*`dict`*) – dictionary containing genomic coverage
- **`snps_path`** (*`str`*) – path to output SNPs
- **`alignments`** (*`set, optional`*) – set of (read_id, alignment_index) tuples to process. All alignments are processed if this option is not provided.
- **`conversions`** (*`set, optional`*) – set of conversions to consider
- **`quality`** (*`int, optional`*) – only count conversions with PHRED quality greater than this value, defaults to *27*
- **`threshold`** (*`float, optional`*) – positions with conversions / coverage > threshold will be considered as SNPs, defaults to *0.5*
- **`min_coverage`** (*`int, optional`*) – only positions with at least this many mapping read_snps are considered, defaults to *1*
- **`n_threads`** (*`int, optional`*) – number of threads, defaults to *8*

Package Contents

Functions

`aggregate_counts`(df_counts, aggregates_path, conversions=frozenset([('TC',)]))	Aggregate conversion counts for each pair of bases.
`calculate_mutation_rates`(df_counts, rates_path, group_by=None)	Calculate mutation rate for each pair of bases.
`merge_aggregates`(*dfs)	Merge multiple aggregate dataframes into one.
`read_aggregates`(aggregates_path)	Read aggregates CSV as a pandas dataframe.
`read_rates`(rates_path)	Read mutation rates CSV as a pandas dataframe.
`check_bam_contains_duplicate`(bam_path, n_reads=100000, n_threads=8)	
`check_bam_contains_secondary`(bam_path, n_reads=100000, n_threads=8)	
`check_bam_contains_unmapped`(bam_path)	
`get_tags_from_bam`(bam_path, n_reads=100000, n_threads=8)	Utility function to retrieve all read tags present in a BAM.
`parse_all_reads`(bam_path, conversions_path, alignments_path, index_path, gene_infos, transcript_infos, strand='forward', umi_tag=None, barcode_tag=None, gene_tag='GX', barcodes=None, n_threads=8, temp_dir=None, nasc=False, control=False, velocity=True, strict_exon_overlap=False, return_splits=False)	Parse all reads in a BAM and extract conversion, content and alignment
`read_alignments`(alignments_path, *args, **kwargs)	Read alignments CSV as a pandas DataFrame.
`read_conversions`(conversions_path, *args, **kwargs)	Read conversions CSV as a pandas DataFrame.
`select_alignments`(df_alignments)	Select alignments among duplicates. This function performs preliminary
`sort_and_index_bam`(bam_path, out_path, n_threads=8, temp_dir=None)	Sort and index BAM.
`call_consensus`(bam_path, out_path, gene_infos, strand='forward', umi_tag=None, barcode_tag=None, gene_tag='GX', barcodes=None, quality=27, add_RS_RI=False, temp_dir=None, n_threads=8)	
`complement_counts`(df_counts, gene_infos)	Complement the counts in the counts dataframe according to gene strand.
`count_conversions`(conversions_path, alignments_path, index_path, counts_path, gene_infos, barcodes=None, snps=None, quality=27, conversions=None, dedup_use_conversions=True, n_threads=8, temp_dir=None)	Count the number of conversions of each read per barcode and gene, along with
`deduplicate_counts`(df_counts, conversions=None, use_conversions=True)	Deduplicate counts based on barcode, UMI, and gene.
`read_counts`(counts_path, *args, **kwargs)	Read counts CSV as a pandas dataframe.
`split_counts_by_velocity`(df_counts)	Split the given counts dataframe by the *velocity* column.
`calculate_coverage`(bam_path, conversions, coverage_path, alignments=None, umi_tag=None, barcode_tag=None, gene_tag='GX', barcodes=None, temp_dir=None, velocity=True)	Calculate coverage of each genomic position per barcode.
`read_coverage`(coverage_path)	Read coverage CSV as a dictionary.
`detect_snps`(conversions_path, index_path, coverage, snps_path, alignments=None, conversions=None, quality=27, threshold=0.5, min_coverage=1, n_threads=8)	Detect SNPs.
`read_snp_csv`(snp_csv)	Read a user-provided SNPs CSV
`read_snps`(snps_path)	Read SNPs CSV as a dictionary

Attributes

CONVERSION_COMPLEMENT

`dynast.preprocessing.`**`aggregate_counts`**(*df_counts*, *aggregates_path*, *conversions=frozenset([('TC',)])*)

Aggregate conversion counts for each pair of bases.

Parameters

- **`df_counts`** (*`pandas.DataFrame`*) – counts dataframe, with complemented reverse strand bases
- **`aggregates_path`** (*`str`*) – path to write aggregate CSV
- **`conversions`** (*`list, optional`*) – conversion(s) in question, defaults to *frozenset([('TC',)])*

Returns path to aggregate CSV that was written

Return type str

`dynast.preprocessing.`**`calculate_mutation_rates`**(*df_counts*, *rates_path*, *group_by=None*)

Calculate mutation rate for each pair of bases.

Parameters

- **`df_counts`** (*`pandas.DataFrame`*) – counts dataframe, with complemented reverse strand bases
- **`rates_path`** (*`str`*) – path to write rates CSV
- **`group_by`** (*`list`*) – column(s) to group calculations by, defaults to *None*, which combines all rows

Returns path to rates CSV

Return type str

`dynast.preprocessing.`**`merge_aggregates`**(**dfs*)

Merge multiple aggregate dataframes into one.

Parameters **`*dfs`** – dataframes to merge

Returns merged dataframe

Return type pandas.DataFrame

`dynast.preprocessing.`**`read_aggregates`**(*aggregates_path*)

Read aggregates CSV as a pandas dataframe.

Parameters **`aggregates_path`** (*`str`*) – path to aggregates CSV

Returns aggregates dataframe

Return type pandas.DataFrame

`dynast.preprocessing.`**`read_rates`**(*rates_path*)

Read mutation rates CSV as a pandas dataframe.

Parameters **`rates_path`** (*`str`*) – path to rates CSV

Returns rates dataframe

Return type pandas.DataFrame

`dynast.preprocessing.`**`check_bam_contains_duplicate`**`(`*bam_path*, *n_reads=100000*, *n_threads=8*`)`

`dynast.preprocessing.`**`check_bam_contains_secondary`**`(`*bam_path*, *n_reads=100000*, *n_threads=8*`)`

`dynast.preprocessing.`**`check_bam_contains_unmapped`**`(`*bam_path*`)`

`dynast.preprocessing.`**`get_tags_from_bam`**`(`*bam_path*, *n_reads=100000*, *n_threads=8*`)`

Utility function to retrieve all read tags present in a BAM.

Parameters

- **bam_path** (`str`) – path to BAM
- **n_reads** (`int, optional`) – number of reads to consider, defaults to *100000*
- **n_threads** (`int, optional`) – number of threads, defaults to *8*

Returns set of all tags found

Return type set

`dynast.preprocessing.`**`parse_all_reads`**`(`*bam_path*, *conversions_path*, *alignments_path*, *index_path*, *gene_infos*, *transcript_infos*, *strand='forward'*, *umi_tag=None*, *barcode_tag=None*, *gene_tag='GX'*, *barcodes=None*, *n_threads=8*, *temp_dir=None*, *nasc=False*, *control=False*, *velocity=True*, *strict_exon_overlap=False*, *return_splits=False*`)`

Parse all reads in a BAM and extract conversion, content and alignment information as CSVs.

Parameters

- **bam_path** (`str`) – path to alignment BAM file
- **conversions_path** (`str`) – path to output information about reads that have conversions
- **alignments_path** (`str`) – path to alignments information about reads
- **index_path** (`str`) – path to conversions index
- **no_index_path** (`str`) – path to no conversions index
- **gene_infos** (`dictionary`) – dictionary containing gene information, as returned by *ngs.gtf.genes_and_transcripts_from_gtf*
- **transcript_infos** (`dictionary`) – dictionary containing transcript information, as returned by *ngs.gtf.genes_and_transcripts_from_gtf*
- **strand** (`str, optional`) – strandedness of the sequencing protocol, defaults to *forward*, may be one of the following: *forward*, *reverse*, *unstranded*
- **umi_tag** (`str, optional`) – BAM tag that encodes UMI, if not provided, *NA* is output in the *umi* column, defaults to *None*
- **barcode_tag** (`str, optional`) – BAM tag that encodes cell barcode, if not provided, *NA* is output in the *barcode* column, defaults to *None*
- **gene_tag** (`str, optional`) – BAM tag that encodes gene assignment, defaults to *GX*
- **barcodes** (`list, optional`) – list of barcodes to be considered. All barcodes are considered if not provided, defaults to *None*
- **n_threads** (`int, optional`) – number of threads, defaults to *8*
- **temp_dir** (`str, optional`) – path to temporary directory, defaults to *None*

- `nasc` (*bool, optional*) – flag to change behavior to match NASC-seq pipeline, defaults to *False*
- `velocity` (*bool, optional*) – whether or not to assign a velocity type to each read, defaults to *True*
- `strict_exon_overlap` (*bool, optional*) – Whether to use a stricter algorithm to assin reads as spliced, defaults to *False*
- `return_splits` (*bool, optional*) – return BAM splits for later reuse, defaults to *True*

Returns (path to conversions, path to alignments, path to conversions index) If *return_splits* is True, then there is an additional return value, which is a list of tuples containing split BAM paths and number of reads in each BAM.

Return type (str, str, str) or (str, str, str, list)

`dynast.preprocessing.`**`read_alignments`**(*alignments_path*, **args*, ***kwargs*)

Read alignments CSV as a pandas DataFrame.

Any additional arguments and keyword arguments are passed to *pandas.read_csv*.

Parameters `alignments_path` (*str*) – path to alignments CSV

Returns conversions dataframe

Return type pandas.DataFrame

`dynast.preprocessing.`**`read_conversions`**(*conversions_path*, **args*, ***kwargs*)

Read conversions CSV as a pandas DataFrame.

Any additional arguments and keyword arguments are passed to *pandas.read_csv*.

Parameters `conversions_path` (*str*) – path to conversions CSV

Returns conversions dataframe

Return type pandas.DataFrame

`dynast.preprocessing.`**`select_alignments`**(*df_alignments*)

Select alignments among duplicates. This function performs preliminary deduplication and returns a list of tuples (read_id, alignment index) to use for coverage calculation and SNP detection.

Parameters `df_alignments` (*pandas.DataFrame*) – alignments dataframe

Returns set of (read_id, alignment index) that were selected

Return type set

`dynast.preprocessing.`**`sort_and_index_bam`**(*bam_path*, *out_path*, *n_threads=8*, *temp_dir=None*)

Sort and index BAM.

If the BAM is already sorted, the sorting step is skipped.

Parameters

- `bam_path` (*str*) – path to alignment BAM file to sort
- `out_path` (*str*) – path to output sorted BAM
- `n_threads` (*int, optional*) – number of threads, defaults to *8*
- `temp_dir` (*str, optional*) – path to temporary directory, defaults to *None*

Returns path to sorted and indexed BAM

Return type str

`dynast.preprocessing.`**`call_consensus`**(*bam_path*, *out_path*, *gene_infos*, *strand='forward'*, *umi_tag=None*, *barcode_tag=None*, *gene_tag='GX'*, *barcodes=None*, *quality=27*, *add_RS_RI=False*, *temp_dir=None*, *n_threads=8*)

`dynast.preprocessing.`**`complement_counts`**(*df_counts*, *gene_infos*)

Complement the counts in the counts dataframe according to gene strand.

Parameters

- **df_counts** (*pandas.DataFrame*) – counts dataframe
- **gene_infos** (*dictionary*) – dictionary containing gene information, as returned by *preprocessing.gtf.parse_gtf*

Returns counts dataframe with counts complemented for reads mapping to genes on the reverse strand

Return type pandas.DataFrame

`dynast.preprocessing.`**`CONVERSION_COMPLEMENT`**

`dynast.preprocessing.`**`count_conversions`**(*conversions_path*, *alignments_path*, *index_path*, *counts_path*, *gene_infos*, *barcodes=None*, *snps=None*, *quality=27*, *conversions=None*, *dedup_use_conversions=True*, *n_threads=8*, *temp_dir=None*)

Count the number of conversions of each read per barcode and gene, along with the total nucleotide content of the region each read mapped to, also per barcode. When a duplicate UMI for a barcode is observed, the read with the greatest number of conversions is selected.

Parameters

- **conversions_path** (*str*) – path to conversions CSV
- **alignments_path** (*str*) – path to alignments information about reads
- **index_path** (*str*) – path to conversions index
- **counts_path** – path to write counts CSV
- **counts_path** – str
- **gene_infos** (*dictionary*) – dictionary containing gene information, as returned by *ngs.gtf.genes_and_transcripts_from_gtf*, defaults to *None*
- **barcodes** (*list, optional*) – list of barcodes to be considered. All barcodes are considered if not provided, defaults to *None*
- **snps** (*dictionary, optional*) – dictionary of contig as keys and list of genomic positions as values that indicate SNP locations, defaults to *None*
- **conversions** (*list, optional*) – conversions to prioritize when deduplicating only applicable for UMI technologies, defaults to *None*
- **dedup_use_conversions** (*bool, optional*) – prioritize reads that have at least one conversion when deduplicating, defaults to *True*
- **quality** (*int, optional*) – only count conversions with PHRED quality greater than this value, defaults to *27*
- **n_threads** (*int, optional*) – number of threads, defaults to *8*
- **temp_dir** (*str, optional*) – path to temporary directory, defaults to *None*

Returns path to counts CSV

Return type str

`dynast.preprocessing.`**`deduplicate_counts`**(*df_counts*, *conversions=None*, *use_conversions=True*)

Deduplicate counts based on barcode, UMI, and gene.

The order of priority is the following. 1. If *use_conversions=True*, reads that have at least one such conversion 2. Reads that align to the transcriptome (exon only) 3. Reads that have highest alignment score 4. If *conversions* is provided, reads that have a larger sum of such conversions

If *conversions* is not provided, reads that have larger sum of all conversions

Parameters

- **df_counts** (*pandas.DataFrame*) – counts dataframe
- **conversions** (*list, optional*) – conversions to prioritize, defaults to *None*
- **use_conversions** (*bool, optional*) – prioritize reads that have conversions first, defaults to *True*

Returns deduplicated counts dataframe

Return type pandas.DataFrame

`dynast.preprocessing.`**`read_counts`**(*counts_path*, **args*, ***kwargs*)

Read counts CSV as a pandas dataframe.

Any additional arguments and keyword arguments are passed to *pandas.read_csv*.

Parameters **counts_path** (*str*) – path to CSV

Returns counts dataframe

Return type pandas.DataFrame

`dynast.preprocessing.`**`split_counts_by_velocity`**(*df_counts*)

Split the given counts dataframe by the *velocity* column.

Parameters **df_counts** (*pandas.DataFrame*) – counts dataframe

Returns dictionary containing *velocity* column values as keys and the subset dataframe as values

Return type dictionary

`dynast.preprocessing.`**`calculate_coverage`**(*bam_path*, *conversions*, *coverage_path*, *alignments=None*, *umi_tag=None*, *barcode_tag=None*, *gene_tag='GX'*, *barcodes=None*, *temp_dir=None*, *velocity=True*)

Calculate coverage of each genomic position per barcode.

Parameters

- **bam_path** (*str*) – path to alignment BAM file
- **conversions** (*dictionary*) – dictionary of contigs as keys and sets of genomic positions as values that indicates positions where conversions were observed
- **coverage_path** (*str*) – path to write coverage CSV
- **alignments** (*set, optional*) – set of (read_id, alignment_index) tuples to process. All alignments are processed if this option is not provided.
- **umi_tag** (*str, optional*) – BAM tag that encodes UMI, if not provided, *NA* is output in the *umi* column, defaults to *None*

- `barcode_tag` (*str, optional*) – BAM tag that encodes cell barcode, if not provided, *NA* is output in the *barcode* column, defaults to *None*
- `gene_tag` (*str, optional*) – BAM tag that encodes gene assignment, defaults to *GX*
- `barcodes` (*list, optional*) – list of barcodes to be considered. All barcodes are considered if not provided, defaults to *None*
- `temp_dir` (*str, optional*) – path to temporary directory, defaults to *None*
- `velocity` (*bool, optional*) – whether or not velocities were assigned

Returns coverage CSV path

Return type str

`dynast.preprocessing.`**`read_coverage`**(*coverage_path*)

Read coverage CSV as a dictionary.

Parameters `coverage_path` (*str*) – path to coverage CSV

Returns coverage as a nested dictionary

Return type dict

`dynast.preprocessing.`**`detect_snps`**(*conversions_path, index_path, coverage, snps_path, alignments=None, conversions=None, quality=27, threshold=0.5, min_coverage=1, n_threads=8*)

Detect SNPs.

Parameters

- `conversions_path` (*str*) – path to conversions CSV
- `index_path` (*str*) – path to conversions index
- `coverage` (*dict*) – dictionary containing genomic coverage
- `snps_path` (*str*) – path to output SNPs
- `alignments` (*set, optional*) – set of (read_id, alignment_index) tuples to process. All alignments are processed if this option is not provided.
- `conversions` (*set, optional*) – set of conversions to consider
- `quality` (*int, optional*) – only count conversions with PHRED quality greater than this value, defaults to *27*
- `threshold` (*float, optional*) – positions with conversions / coverage > threshold will be considered as SNPs, defaults to *0.5*
- `min_coverage` (*int, optional*) – only positions with at least this many mapping read_snps are considered, defaults to *1*
- `n_threads` (*int, optional*) – number of threads, defaults to *8*

`dynast.preprocessing.`**`read_snp_csv`**(*snp_csv*)

Read a user-provided SNPs CSV

Parameters `snp_csv` (*str*) – path to SNPs CSV

Returns dictionary of contigs as keys and sets of genomic positions with SNPs as values

Return type dictionary

`dynast.preprocessing.`**`read_snps`**(*snps_path*)

Read SNPs CSV as a dictionary

Parameters **`snps_path`** (*`str`*) – path to SNPs CSV

Returns dictionary of contigs as keys and sets of genomic positions with SNPs as values

Return type dictionary

4.2 Submodules

4.2.1 `dynast.align`

Module Contents

Functions

`STAR_solo`(fastqs, index_dir, out_dir, technology, whitelist_path=None, strand='forward', n_threads=8, temp_dir=None, nasc=False, overrides=None)	Align FASTQs with STARsolo.
`align`(fastqs, index_dir, out_dir, technology, whitelist_path=None, strand='forward', n_threads=8, temp_dir=None, nasc=False, overrides=None)	

`dynast.align.`**`STAR_solo`**(*fastqs*, *index_dir*, *out_dir*, *technology*, *whitelist_path=None*, *strand='forward'*, *n_threads=8*, *temp_dir=None*, *nasc=False*, *overrides=None*)

Align FASTQs with STARsolo.

Parameters

- **`fastqs`** (*`list`*) – list of path to FASTQs. Order matters – STAR assumes the UMI and barcode are in read 2
- **`index_dir`** (*`str`*) – path to directory containing STAR index
- **`out_dir`** (*`str`*) – path to directory to place STAR output
- **`technology`** (*`collections.namedtuple`*) – a *Technology* object defined in *technology.py*
- **`whitelist_path`** (*`str, optional`*) – path to textfile containing barcode whitelist, defaults to *None*
- **`strand`** (*`str, optional`*) – strandedness of the sequencing protocol, defaults to *forward*, may be one of the following: *forward*, *reverse*, *unstranded*
- **`n_threads`** (*`int, optional`*) – number of threads to use, defaults to *8*
- **`temp_dir`** (*`str, optional`*) – STAR temporary directory, defaults to *None*, which uses the system temporary directory
- **`nasc`** (*`bool, optional`*) – whether or not to use STAR configuration used in NASC-seq pipeline, defaults to *False*
- **`overrides`** (*`dictionary, optional`*) – STAR command-line argument overrides, defaults to *None*

Returns dictionary containing output files

Return type dict

`dynast.align.align`(*fastqs*, *index_dir*, *out_dir*, *technology*, *whitelist_path=None*, *strand='forward'*, *n_threads=8*, *temp_dir=None*, *nasc=False*, *overrides=None*)

4.2.2 `dynast.config`

Module Contents

`dynast.config.PACKAGE_PATH`

`dynast.config.PLATFORM`

`dynast.config.BINS_DIR`

`dynast.config.MODELS_DIR`

`dynast.config.MODEL_PATH`

`dynast.config.MODEL_NAME = pi`

`dynast.config.RECOMMENDED_MEMORY`

`dynast.config.STAR_ARGUMENTS`

`dynast.config.STAR_SOLO_ARGUMENTS`

`dynast.config.NASC_ARGUMENTS`

`dynast.config.BAM_PEEK_READS = 500000`

`dynast.config.BAM_REQUIRED_TAGS = ['MD']`

`dynast.config.BAM_READGROUP_TAG = RG`

`dynast.config.BAM_BARCODE_TAG = CB`

`dynast.config.BAM_UMI_TAG = UB`

`dynast.config.BAM_GENE_TAG = GX`

`dynast.config.BAM_CONSENSUS_READ_COUNT_TAG = RN`

`dynast.config.COUNTS_SPLIT_THRESHOLD = 50000`

`dynast.config.VELOCITY_BLACKLIST = ['unassigned', 'ambiguous']`

4.2.3 `dynast.consensus`

Module Contents

Functions

consensus(bam_path, gtf_path, out_dir, strand='forward', umi_tag=None, barcode_tag=None, gene_tag='GX', barcodes=None, quality=27, add_RS_RI=False, n_threads=8, temp_dir=None)	

`dynast.consensus.`**`consensus`**(*bam_path*, *gtf_path*, *out_dir*, *strand='forward'*, *umi_tag=None*, *barcode_tag=None*, *gene_tag='GX'*, *barcodes=None*, *quality=27*, *add_RS_RI=False*, *n_threads=8*, *temp_dir=None*)

4.2.4 `dynast.constants`

Module Contents

`dynast.constants.`**`STATS_PREFIX = run_info`**

`dynast.constants.`**`STAR_SOLO_DIR = Solo.out`**

`dynast.constants.`**`STAR_GENE_DIR = Gene`**

`dynast.constants.`**`STAR_RAW_DIR = raw`**

`dynast.constants.`**`STAR_FILTERED_DIR = filtered`**

`dynast.constants.`**`STAR_VELOCYTO_DIR = Velocyto`**

`dynast.constants.`**`STAR_BAM_FILENAME = Aligned.sortedByCoord.out.bam`**

`dynast.constants.`**`STAR_BAI_FILENAME = Aligned.sortedByCoord.out.bai`**

`dynast.constants.`**`STAR_BARCODES_FILENAME = barcodes.tsv`**

`dynast.constants.`**`STAR_FEATURES_FILENAME = features.tsv`**

`dynast.constants.`**`STAR_MATRIX_FILENAME = matrix.mtx`**

`dynast.constants.`**`CONSENSUS_BAM_FILENAME = consensus.bam`**

`dynast.constants.`**`COUNT_DIR = count`**

`dynast.constants.`**`PARSE_DIR = 0_parse`**

`dynast.constants.`**`CONVS_FILENAME = convs.pkl.gz`**

`dynast.constants.`**`GENES_FILENAME = genes.pkl.gz`**

`dynast.constants.`**`CONVERSIONS_FILENAME = conversions.csv`**

`dynast.constants.`**`CONVERSIONS_INDEX_FILENAME = conversions.idx`**

`dynast.constants.`**`ALIGNMENTS_FILENAME = alignments.csv`**

`dynast.constants.`**`COVERAGE_FILENAME = coverage.csv`**

`dynast.constants.`**`COVERAGE_INDEX_FILENAME = coverage.idx`**

`dynast.constants.`**`SNPS_FILENAME = snps.csv`**

`dynast.constants.`**`COUNTS_PREFIX = counts`**

`dynast.constants.`**`ESTIMATE_DIR = estimate`**

`dynast.constants.`**`RATES_FILENAME = rates.csv`** `dynast.constants.`**`P_E_FILENAME = p_e.csv`**

`dynast.constants.`**`P_C_PREFIX = p_c`**

`dynast.constants.`**`AGGREGATE_FILENAME = aggregate.csv`**

`dynast.constants.`**`ADATA_FILENAME = adata.h5ad`**

4.2.5 `dynast.count`

Module Contents

Functions

`count`(bam_path, gtf_path, out_dir, strand='forward', umi_tag=None, barcode_tag=None, gene_tag='GX', barcodes=None, control=False, quality=27, conversions=frozenset([('TC',)]), snp_threshold=0.5, snp_min_coverage=1, snp_csv=None, n_threads=8, temp_dir=None, velocity=True, strict_exon_overlap=False, dedup_mode='auto', nasc=False, overwrite=False)	

`dynast.count.`**`count`**(*bam_path*, *gtf_path*, *out_dir*, *strand='forward'*, *umi_tag=None*, *barcode_tag=None*, *gene_tag='GX'*, *barcodes=None*, *control=False*, *quality=27*, *conversions=frozenset([('TC',)])*, *snp_threshold=0.5*, *snp_min_coverage=1*, *snp_csv=None*, *n_threads=8*, *temp_dir=None*, *velocity=True*, *strict_exon_overlap=False*, *dedup_mode='auto'*, *nasc=False*, *overwrite=False*)

4.2.6 `dynast.estimate`

Module Contents

Functions

`estimate`(count_dirs, out_dir, reads='complete', groups=None, ignore_groups_for_pi=True, genes=None, downsample=None, downsample_mode='uniform', cell_threshold=1000, cell_gene_threshold=16, control_p_e=None, control=False, n_threads=8, temp_dir=None, nasc=False, seed=None)	

`dynast.estimate.`**`estimate`**(*count_dirs*, *out_dir*, *reads='complete'*, *groups=None*, *ignore_groups_for_pi=True*, *genes=None*, *downsample=None*, *downsample_mode='uniform'*, *cell_threshold=1000*, *cell_gene_threshold=16*, *control_p_e=None*, *control=False*, *n_threads=8*, *temp_dir=None*, *nasc=False*, *seed=None*)

4.2.7 `dynast.logging`

Module Contents

`dynast.logging.`**`logger`**

4.2.8 `dynast.main`

Module Contents

Functions

`print_technologies`()	Displays a list of supported technologies along with whether a whitelist
`setup_ref_args`(parser, parent)	Helper function to set up a subparser for the *ref* command.
`setup_align_args`(parser, parent)	
`setup_consensus_args`(parser, parent)	Helper function to set up a subparser for the *consensus* command.
`setup_count_args`(parser, parent)	Helper function to set up a subparser for the *count* command.
`setup_estimate_args`(parser, parent)	Helper function to set up a subparser for the *estimate* command.
`parse_ref`(parser, args, temp_dir=None)	Parser for the *ref* command.
`parse_align`(parser, args, temp_dir=None)	
`parse_consensus`(parser, args, temp_dir=None)	Parser for the *consensus* command.
`parse_count`(parser, args, temp_dir=None)	Parser for the *count* command.
`parse_estimate`(parser, args, temp_dir=None)	
`main`()	

Attributes

`COMMAND_TO_FUNCTION`

`dynast.main.`**`print_technologies`**`()`

Displays a list of supported technologies along with whether a whitelist is provided for that technology.

`dynast.main.`**`setup_ref_args`**`(`*parser*, *parent*`)`

Helper function to set up a subparser for the *ref* command.

Parameters

- **`parser`** – argparse parser to add the *ref* command to

- `parent` – argparse parser parent of the newly added subcommand. used to inherit shared commands/flags

Returns the newly added parser

Return type argparse.ArgumentParser

`dynast.main.`**`setup_align_args`**`(`*parser*, *parent*`)`

`dynast.main.`**`setup_consensus_args`**`(`*parser*, *parent*`)`

Helper function to set up a subparser for the *consensus* command.

Parameters

- `parser` – argparse parser to add the *consensus* command to
- `parent` – argparse parser parent of the newly added subcommand. used to inherit shared commands/flags

Returns the newly added parser

Return type argparse.ArgumentParser

`dynast.main.`**`setup_count_args`**`(`*parser*, *parent*`)`

Helper function to set up a subparser for the *count* command.

Parameters

- `parser` – argparse parser to add the *count* command to
- `parent` – argparse parser parent of the newly added subcommand. used to inherit shared commands/flags

Returns the newly added parser

Return type argparse.ArgumentParser

`dynast.main.`**`setup_estimate_args`**`(`*parser*, *parent*`)`

Helper function to set up a subparser for the *estimate* command.

Parameters

- `parser` – argparse parser to add the *estimate* command to
- `parent` – argparse parser parent of the newly added subcommand. used to inherit shared commands/flags

Returns the newly added parser

Return type argparse.ArgumentParser

`dynast.main.`**`parse_ref`**`(`*parser*, *args*, *temp_dir=None*`)`

Parser for the *ref* command. :param args: Command-line arguments dictionary, as parsed by argparse :type args: dict

`dynast.main.`**`parse_align`**`(`*parser*, *args*, *temp_dir=None*`)`

`dynast.main.`**`parse_consensus`**`(`*parser*, *args*, *temp_dir=None*`)`

Parser for the *consensus* command. :param args: Command-line arguments dictionary, as parsed by argparse :type args: dict

`dynast.main.`**`parse_count`**`(`*parser*, *args*, *temp_dir=None*`)`

Parser for the *count* command. :param args: Command-line arguments dictionary, as parsed by argparse :type args: dict `dynast.main.`**`parse_estimate`**(*parser*, *args*, *temp_dir=None*)

`dynast.main.`**`COMMAND_TO_FUNCTION`**

`dynast.main.`**`main`**()

4.2.9 `dynast.ref`

Module Contents

Functions

`STAR_genomeGenerate`(fasta_path, gtf_path, index_dir, n_threads=8, memory=16 * 1024**3, temp_dir=None)	Generate a STAR index from a reference.
`ref`(fasta_path, gtf_path, index_dir, n_threads=8, memory=16 * 1024**3, temp_dir=None)	

`dynast.ref.`**`STAR_genomeGenerate`**(*fasta_path*, *gtf_path*, *index_dir*, *n_threads=8*, *memory=16 * 1024 ** 3*, *temp_dir=None*)

Generate a STAR index from a reference.

Parameters

- **`fasta_path`** (*`str`*) – path to genome fasta
- **`gtf_path`** (*`str`*) – path to GTF annotation
- **`index_dir`** (*`str`*) – path to output STAR index
- **`n_threads`** (*`int, optional`*) – number of threads, defaults to *8*
- **`memory`** (*`int, optional`*) – *suggested* memory to use (this is not guaranteed), in bytes, defaults to *16 * 1024**3*
- **`temp_dir`** (*`str, optional`*) – temporary directory, defaults to *None*

Returns dictionary of generated index

Return type dictionary

`dynast.ref.`**`ref`**(*fasta_path*, *gtf_path*, *index_dir*, *n_threads=8*, *memory=16 * 1024 ** 3*, *temp_dir=None*)

4.2.10 `dynast.stats`

Module Contents

Classes

`Step`	Class that represents a processing step.
`Stats`	Class used to collect run statistics.

`class dynast.stats.Step`(*skipped=False*, ***kwargs*)

Class that represents a processing step.

`start`(*self*)

Signal the step has started.

`end`(*self*)

Signal the step has ended.

`to_dict`(*self*)

Convert this step to a dictionary.

Returns dictionary of class variables

Return type dictionary

`class dynast.stats.Stats`

Class used to collect run statistics.

`start`(*self*)

Start collecting statistics.

Sets start time, the command line call.

`end`(*self*)

End collecting statistics.

`step`(*self*, *key*, *skipped=False*, ***kwargs*)

Register a processing step.

Any additional keyword arguments are passed to the constructor of *Step*.

Parameters

- **key** (`str`) – processing key
- **skipped** (`bool, optional`) – whether or not this step is skipped, defaults to *False*

`save`(*self*, *path*)

Save statistics as JSON to path.

Parameters **path** (`str`) – path to JSON

Returns path to saved JSON

Return type str

`to_dict`(*self*)

Convert statistics to dictionary, so that it is easily parsed by the report-rendering functions.

4.2.11 `dynast.technology`

Module Contents

`dynast.technology.Technology`

`dynast.technology.BARCODE_UMI_TECHNOLOGIES`

`dynast.technology.PLATE_TECHNOLOGIES`

`dynast.technology.TECHNOLOGIES`

`dynast.technology.TECHNOLOGIES_MAP`

4.2.12 `dynast.utils`

Module Contents

Classes

suppress_stdout_stderr	A context manager for doing a "deep suppression" of stdout and stderr in

Functions

get_STAR_binary_path()	Get the path to the platform-dependent STAR binary included with
get_STAR_version()	Get the provided STAR version.
combine_arguments(args, additional)	Combine two dictionaries representing command-line arguments.
arguments_to_list(args)	Convert a dictionary of command-line arguments to a list.
get_file_descriptor_limit()	Get the current value for the maximum number of open file descriptors
get_max_file_descriptor_limit()	Get the maximum allowed value for the maximum number of open file
increase_file_descriptor_limit(limit)	Context manager that can be used to temporarily increase the maximum
get_available_memory()	Get total amount of available memory (total memory - used memory) in bytes.
make_pool_with_counter(n_threads)	Create a new Process pool with a shared progress counter.
display_progress_with_counter(counter, total, *async_results, desc=None)	Display progress bar for displaying multiprocessing progress.
as_completed_with_progress(futures)	Wrapper around *concurrent.futures.as_completed* that displays a progress bar.
split_index(index, n=8)	Split a conversions index, which is a list of tuples (file position,
downsample_counts(df_counts, proportion=None, count=None, seed=None, group_by=None)	Downsample the given counts dataframe according to the `proportion` or
counts_to_matrix(df_counts, barcodes, features, barcode_column='barcode', feature_column='GX')	Convert a counts dataframe to a sparse counts matrix.
split_counts(df_counts, barcodes, features, barcode_column='barcode', feature_column='GX', conversions=('TC',))	Split counts dataframe into two count matrices by a column.
split_matrix(matrix, pis, barcodes, features)	Split the given matrix based on provided fraction of new RNA.
results_to_adata(df_counts, conversions=frozenset([('TC',)]), gene_infos=None, pis=None)	Compile all results to a single anndata.
patch_mp_connection_bpo_17560()	Apply PR-10305 / bpo-17560 connection send/receive max size update

Attributes

run_executable	
open_as_text	
decompress_gzip	
flatten_dict_values	
mkstemp	
all_exists	
flatten_dictionary	
flatten_iter	
merge_dictionaries	
write_pickle	
read_pickle	

dynast.utils.**run_executable**

dynast.utils.**open_as_text**

dynast.utils.**decompress_gzip**

dynast.utils.**flatten_dict_values**

dynast.utils.**mkstemp**

dynast.utils.**all_exists**

dynast.utils.**flatten_dictionary**

dynast.utils.**flatten_iter**

dynast.utils.**merge_dictionaries**

dynast.utils.**write_pickle**

dynast.utils.**read_pickle**

exception dynast.utils.**UnsupportedOSException**

Bases: Exception

Common base class for all non-exit exceptions.

class dynast.utils.**suppress_stdout_stderr**

A context manager for doing a "deep suppression" of stdout and stderr in Python, i.e. will suppress all print, even if the print originates in a compiled C/Fortran sub-function.

This will not suppress raised exceptions, since exceptions are printed

to stderr just before a script exits, and after the context manager has exited (at least, I think that is why it lets exceptions through). https://github.com/facebook/prophet/issues/223

`__enter__`(*self*)

`__exit__`(*self*, **_*)

`dynast.utils.get_STAR_binary_path()`

Get the path to the platform-dependent STAR binary included with the installation.

Returns path to the binary

Return type str

`dynast.utils.get_STAR_version()`

Get the provided STAR version.

Returns version string

Return type str

`dynast.utils.combine_arguments`(*args*, *additional*)

Combine two dictionaries representing command-line arguments.

Any duplicate keys will be merged according to the following procedure: 1. If the value in both dictionaries are lists, the two lists are combined. 2. Otherwise, the value in the first dictionary is OVERWRITTEN.

Parameters

- `args` (*dictionary*) – original command-line arguments
- `additional` (*dictionary*) – additional command-line arguments

Returns combined command-line arguments

Return type dictionary

`dynast.utils.arguments_to_list`(*args*)

Convert a dictionary of command-line arguments to a list.

Parameters `args` (*dictionary*) – command-line arguments

Returns list of command-line arguments

Return type list

`dynast.utils.get_file_descriptor_limit()`

Get the current value for the maximum number of open file descriptors in a platform-dependent way.

Returns the current value of the maximum number of open file descriptors.

Return type int

`dynast.utils.get_max_file_descriptor_limit()`

Get the maximum allowed value for the maximum number of open file descriptors.

Note that for Windows, there is not an easy way to get this, as it requires reading from the registry. So, we just return the maximum for a vanilla Windows installation, which is 8192. https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/setmaxstdio?view=vs-2019

Similarly, on MacOS, we return a hardcoded 10240.

Returns maximum allowed value for the maximum number of open file descriptors

Return type int

dynast.utils.**increase_file_descriptor_limit**(*limit*)

Context manager that can be used to temporarily increase the maximum number of open file descriptors for the current process. The original value is restored when execution exits this function.

This is required when running STAR with many threads.

Parameters **limit** (*int*) – maximum number of open file descriptors will be increased to this value for the duration of the context

dynast.utils.**get_available_memory**()

Get total amount of available memory (total memory - used memory) in bytes.

Returns available memory in bytes

Return type int

dynast.utils.**make_pool_with_counter**(*n_threads*)

Create a new Process pool with a shared progress counter.

Parameters **n_threads** (*int*) – number of processes

Returns (Process pool, progress counter, lock)

Return type (multiprocessing.Pool, multiprocessing.Value, multiprocessing.Lock)

dynast.utils.**display_progress_with_counter**(*counter*, *total*, **async_results*, *desc=None*)

Display progress bar for displaying multiprocessing progress.

Parameters

- **counter** (*multiprocessing.Value*) – progress counter
- **total** (*int*) – maximum number of units of processing
- ***async_results** – multiprocessing results to monitor. These are used to determine when all processes are done.
- **desc** (*str, optional*) – progress bar description, defaults to *None*

dynast.utils.**as_completed_with_progress**(*futures*)

Wrapper around *concurrent.futures.as_completed* that displays a progress bar.

Parameters **futures** (*iterable*) – iterator of *concurrent.futures.Future* objects

dynast.utils.**split_index**(*index*, *n=8*)

Split a conversions index, which is a list of tuples (file position, number of lines, alignment position), one for each read, into *n* approximately equal parts. This function is used to split the conversions CSV for multiprocessing.

Parameters

- **index** (*list*) – index
- **n** (*int, optional*) – number of splits, defaults to *8*

Returns list of parts, where each part is a list of (file position, number of lines, alignment position) tuples

Return type list

dynast.utils.**downsample_counts**(*df_counts*, *proportion=None*, *count=None*, *seed=None*, *group_by=None*)

Downsample the given counts dataframe according to the `proportion` or `count` arguments. One of these two must be provided, but not both. The dataframe is assumed to be UMI-deduplicated.

Parameters

- **df_counts** (*pandas.DataFrame*) – counts dataframe
- **proportion** (*float, optional*) – proportion of reads (UMIs) to keep, defaults to None
- **count** (*int, optional*) – absolute number of reads (UMIs) to keep, defaults to None
- **seed** (*int, optional*) – random seed, defaults to None
- **group_by** (*list, optional*) – Columns in the counts dataframe to use to group entries. When this is provided, UMIs are no longer sampled at random, but instead grouped by this argument, and only groups that have more than `count` UMIs are downsampled.

Returns downsampled counts dataframe

Return type pandas.DataFrame

`dynast.utils.`**`counts_to_matrix`**(*df_counts*, *barcodes*, *features*, *barcode_column='barcode'*, *feature_column='GX'*)

Convert a counts dataframe to a sparse counts matrix.

Counts are assumed to be appropriately deduplicated.

Parameters

- **df_counts** (*pandas.DataFrame*) – counts dataframe
- **barcodes** (*list*) – list of barcodes that will map to the rows
- **features** (*list*) – list of features (i.e. genes) that will map to the columns
- **barcode_column** (*str*) – column in counts dataframe to use as barcodes, defaults to *barcode*
- **feature_column** (*str*) – column in counts dataframe to use as features, defaults to *GX*

Returns sparse counts matrix

Return type scipy.sparse.csrmatrix

`dynast.utils.`**`split_counts`**(*df_counts*, *barcodes*, *features*, *barcode_column='barcode'*, *feature_column='GX'*, *conversions=('TC',)*)

Split counts dataframe into two count matrices by a column.

Parameters

- **df_counts** (*pandas.DataFrame*) – counts dataframe
- **barcodes** (*list*) – list of barcodes that will map to the rows
- **features** (*list*) – list of features (i.e. genes) that will map to the columns
- **barcode_column** (*str, optional*) – column in counts dataframe to use as barcodes, defaults to *barcode*
- **feature_column** (*str, optional*) – column in counts dataframe to use as features, defaults to *GX*
- **conversions** (*tuple, optional*) – conversion(s) in question, defaults to *('TC',)*

Returns (count matrix of *conversion==0*, count matrix of *conversion>0*)

Return type (scipy.sparse.csrmatrix, scipy.sparse.csrmatrix) `dynast.utils.`**`split_matrix`**(*matrix*, *pis*, *barcodes*, *features*)

Split the given matrix based on provided fraction of new RNA.

Parameters

- **`matrix`** (*`numpy.ndarray or scipy.sparse.spmatrix`*) – matrix to split
- **`pis`** (*`dictionary`*) – dictionary containing pi estimates
- **`barcodes`** (*`list`*) – all barcodes
- **`features`** (*`list`*) – all features (i.e. genes)

Returns (matrix of pi masks, matrix of unlabeled RNA, matrix of labeled RNA)

Return type (scipy.sparse.spmatrix, scipy.sparse.spmatrix, scipy.sparse.spmatrix)

`dynast.utils.`**`results_to_adata`**(*df_counts*, *conversions=frozenset([('TC',)])*, *gene_infos=None*, *pis=None*)

Compile all results to a single anndata.

Parameters

- **`df_counts`** (*`pandas.DataFrame`*) – counts dataframe, with complemented reverse strand bases
- **`conversions`** (*`list, optional`*) – conversion(s) in question, defaults to *frozenset([('TC',)])*
- **`gene_infos`** (*`dict, optional`*) – dictionary containing gene information, defaults to *None*
- **`pis`** (*`dict, optional`*) – dictionary of estimated pis, defaults to *None*

Returns anndata containing all results

Return type anndata.AnnData

`dynast.utils.`**`patch_mp_connection_bpo_17560`**()

Apply PR-10305 / bpo-17560 connection send/receive max size update

See the original issue at https://bugs.python.org/issue17560 and https://github.com/python/cpython/pull/10305 for the pull request.

This only supports Python versions 3.3 - 3.7, this function does nothing for Python versions outside of that range.

Taken from https://stackoverflow.com/a/47776649

4.3 Package Contents

`dynast.`**`__version__`**` = `**`0.2.0`**

CHAPTER

FIVE

REFERENCES

CHAPTER

SIX

NASC-SEQ

The new transcriptome alkylation-dependent scRNA-seq (NASC-seq) was developed by [Hendriks2019]. It uses Smart-seq, which is a plate-based scRNA-seq method that provides great read coverage, compared to droplet-based methods [Picelli2013]. Smart-seq experiments generate single or pairs of FASTQs for each cell sequenced, which dynast processes simultaneously.

- Sequencing technology: Smart-Seq2
- Induced conversion: T>C

6.1 Alignment

Here, we assume the appropriate STAR index has already been built (see *Building the STAR index with ref*). Since we have multiple sets of FASTQs, we need to prepare a FASTQ manifest CSV, instead of providing these as an argument to `dynast align`. The manifest CSV contains three columns where the first column is a unique cell name/ID, the second column is the path to the first FASTQ, and the third is the path to the second FASTQ. For single-end reads, the third column can be a single `-` character. Here is an example with two cells:

```
cell_1,path/to/R1.fastq.gz,path/to/R2.fastq.gz
cell_2,path/to/R1.fastq.gz,-
```

Then, we use this manifest as the input to `dynast align`.

```
dynast align -i path/to/STAR/index -o path/to/align/output -x smartseq manifest.csv
```

This will run STAR alignment and output files to `path/to/align/output`.

6.2 Quantification

The alignment BAM is generated at `path/to/align/output/Aligned.sortedByCoord.out.bam`, which we provde as input to `dynast count`. We also need to provide the gene annotation GTF that was used to generate the STAR index to `-g`.

```
dynast count -g path/to/GTF.gtf --barcode-tag RG path/to/align/output/Aligned.
↪sortedByCoord.out.bam -o path/to/count/output --conversion TC
```

This will quantify all RNA species and write the count matrices to `path/to/count/output/adata.h5ad`.

CHAPTER

SEVEN

SCSLAM-SEQ

scSLAM-seq was developed by [Erhard2019] and is the single-cell adaptation of thiol(SH)-linked alkylation for metabolic sequencing of RNA (SLAM-seq) [Herzog2017]. Similar to *NASC-seq*, scSLAM-seq is based on the Smart-seq protocol [Picelli2013]. Smart-seq experiments generate single or pairs of FASTQs for each cell sequenced, which dynast processes simultaneously.

- Sequencing technology: Smart-Seq2
- Induced conversion: T>C

7.1 Alignment

Here, we assume the appropriate STAR index has already been built (see *Building the STAR index with ref*). Since we have multiple sets of FASTQs, we need to prepare a FASTQ manifest CSV, instead of providing these as an argument to `dynast align`. The manifest CSV contains three columns where the first column is a unique cell name/ID, the second column is the path to the first FASTQ, and the third is the path to the second FASTQ. For single-end reads, the third column can be a single `-` character. Here is an example with two cells:

```
cell_1,path/to/R1.fastq.gz,path/to/R2.fastq.gz
cell_2,path/to/R1.fastq.gz,-
```

Then, we use this manifest as the input to `dynast align`.

```
dynast align -i path/to/STAR/index -o path/to/align/output -x smartseq --strand␣
↪unstranded manifest.csv
```

Note that we provide `--strand unstranded` because the Smart-seq protocol used with scSLAM-seq produces unstranded reads. This will run STAR alignment and output files to `path/to/align/output`.

7.2 Quantification

The alignment BAM is generated at `path/to/align/output/Aligned.sortedByCoord.out.bam`, which we provde as input to `dynast count`. We also need to provide the gene annotation GTF that was used to generate the STAR index to `-g`.

```
dynast count -g path/to/GTF.gtf --barcode-tag RG path/to/align/output/Aligned.
↪sortedByCoord.out.bam -o path/to/count/output --conversion TC --strand unstranded
```

Note that we provide `--strand unstranded` again because the Smart-seq protocol used with scSLAM-seq produces unstranded reads. This will quantify all RNA species and write the count matrices to `path/to/count/output/adata.h5ad`.

CHAPTER

EIGHT

SCNT-SEQ

The single-cell metabolically labeled new RNA tagging sequencing (scNT-seq) was developed by [Qiu2020]. It uses Drop-seq, which is a droplet-based scRNA-seq method [Macosko2015].

- Sequencing technology: Drop-seq
- Induced conversion: T>C

8.1 Alignment

Here, we assume the appropriate STAR index has already been built (see *Building the STAR index with ref*). A single sample will consist of a pair of FASTQs, one containing the cell barcode and UMI sequences and the other containing the biological cDNA sequences. Let's say these two FASTQs are `barcode_umi.fastq.gz` and `cdna.fastq.gz`.

```
dynast align -i path/to/STAR/index -o path/to/align/output -x dropseq cdna.fastq.gz␣
↪barcode_umi.fastq.gz
```

This will run STAR alignment and output files to `path/to/align/output`.

8.2 Consensus

Optionally, we can call consensus sequences for each UMI using `dynast consensus`. This command requires the alignment BAM and the gene annotation GTF that was used to generate the STAR index.

```
dynast consensus -g path/to/GTF.gtf --barcode-tag CB --umi-tag UB path/to/align/output/
↪Aligned.sortedByCoord.out.bam -o path/to/consensus/output
```

This will create a new BAM file named `path/to/consensus/output/consensus.bam`, which you can then use in the next step in place of the original alignment BAM.

8.3 Quantification

Finally, to quantify the number of labeled/unlabeled RNA, we run `dynast count` with the appropriate alignment BAM and the gene annotation GTF that was used to generate the STAR index to `-g`.

```
dynast count -g path/to/GTF.gtf --barcode-tag CB --umi-tag UB path/to/alignment.bam -o␣
↪path/to/count/output --conversion TC
```

where `path/to/alignment.bam` should be `path/to/align/output/Aligned.sortedByCoord.out.bam` if you did not run `dynast consensus`, or `path/to/consensus/output/consensus.bam` if you did.

This will quantify all RNA species and write the count matrices to `path/to/count/output/adata.h5ad`.

CHAPTER

NINE

SCI-FATE

The single-cell combinatorial indexing and messenger RNA labeling (sci-fate) was developed by [Cao2020].

- Sequencing technology: sci-fate
- Induced conversion: T>C

9.1 Alignment

Here, we assume the appropriate STAR index has already been built (see *Building the STAR index with ref*). A single sample will consist of a pair of FASTQs, one containing the cell barcode and UMI sequences and the other containing the biological cDNA sequences. Let's say these two FASTQs are `barcode_umi.fastq.gz` and `cdna.fastq.gz`.

```
dynast align -i path/to/STAR/index -o path/to/align/output -x scifate cdna.fastq.gz␣
↪barcode_umi.fastq.gz
```

This will run STAR alignment and output files to `path/to/align/output`.

9.2 Consensus

Optionally, we can call consensus sequences for each UMI using `dynast consensus`. This command requires the alignment BAM and the gene annotation GTF that was used to generate the STAR index.

```
dynast consensus -g path/to/GTF.gtf --barcode-tag CB --umi-tag UB path/to/align/output/
↪Aligned.sortedByCoord.out.bam -o path/to/consensus/output
```

This will create a new BAM file named `path/to/consensus/output/consensus.bam`, which you can then use in the next step in place of the original alignment BAM.

9.3 Quantification

Finally, to quantify the number of labeled/unlabeled RNA, we run `dynast count` with the appropriate alignment BAM and the gene annotation GTF that was used to generate the STAR index to `-g`.

```
dynast count -g path/to/GTF.gtf --barcode-tag CB --umi-tag UB path/to/alignment.bam -o␣
↪path/to/count/output --conversion TC
```

where `path/to/alignment.bam` should be `path/to/align/output/Aligned.sortedByCoord.out.bam` if you did not run `dynast consensus`, or `path/to/consensus/output/consensus.bam` if you did.

This will quantify all RNA species and write the count matrices to `path/to/count/output/adata.h5ad`.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Dobin2013] https://doi.org/10.1093/bioinformatics/bts635

[Picelli2013] https://doi.org/10.1038/nmeth.2639

[Macosko2015] https://doi.org/10.1016/j.cell.2015.05.002

[Herzog2017] https://doi.org/10.1038/nmeth.4435

[Jürges2018] https://doi.org/10.1093/bioinformatics/bty256

[Erhard2019] https://doi.org/10.1038/s41586-019-1369-y

[Hendriks2019] https://doi.org/10.1038/s41467-019-11028-9

[Cao2020] https://doi.org/10.1038/s41587-020-0480-9

[Qiu2020] https://doi.org/10.1038/s41592-020-0935-4

PYTHON MODULE INDEX

INDEX