Pegasus Documentation

Release 1.2.0

Bo Li, Joshua Gould, Yiming Yang, Siranush Sarkizova, et al.

Jan 08, 2021

CONTENTS

Pegasus is a tool for analyzing transcriptomes of millions of single cells. It is a command line tool, a python package and a base for Cloud-based analysis workflows.

Read documentation

CHAPTER

ONE

RELEASE HIGHLIGHTS IN CURRENT STABLE

1.1 1.2.0 December 25, 2020

- tSNE support:
  - `tsne` function in API: Use FIt-SNE for tSNE embedding calculation. No longer support MulticoreTSNE.
  - Determine `learning_rate` argument in `tsne` more dynamically. ([Belkina19], [Kobak19])
  - By default, use PCA embedding for initialization in `tsne`. ([Kobak19])
  - Remove `net_tsne` and `fitsne` functions from API.
  - Remove `--net-tsne` and `--fitsne` options from **pegasus cluster** command.
- Add multimodal support on RNA and CITE-Seq data back: `--citeseq`, `--citeseq-umap`, and `--citeseq-umap-exclude` in **pegasus cluster** command.
- Doublet detection:
  - Add automated doublet cutoff inference to `infer_doublets` function in API. ([Li20-2])
  - Expose doublet detection to command-line tool: `--infer-doublets`, `--expected-doublet-rate`, and `--dbl-cluster-attr` in **pegasus cluster** command.
  - Add doublet detection tutorial.
- Allow multiple marker files used in cell type annotation: `annotate` function in API; `--markers` option in **pegasus annotate_cluster** command.
- Rename *pc_regress_out* function in API to `regress_out`.
- Update the regress out tutorial.
- Bug fix.

1.1.1 Installation

Pegasus works with Python 3 only (Python 3.5 or above).

Linux

The installation has been tested on Ubuntu (18.04 or later).

Prerequisites

First, install the following dependencies by:

```
sudo apt install build-essential libxml2-dev zlib1g-dev
```

Next, you can install Pegasus system-wide by PyPI (see Linux Install via PyPI), or within a Miniconda environment (see Linux Install via Miniconda).

To use the Force-directed-layout (FLE) embedding feature, you'll need Java. You can either install Oracle JDK, or install OpenJDK which is included in Ubuntu official repository:

```
sudo apt install default-jdk
```

Install via PyPI

First, install Python 3 and *pip* tool for Python 3:

```
sudo apt install python3 python3-pip
```

Now install Pegasus via *pip*:

```
pip3 install pegasuspy
```

There are optional packages that you can install:

- **mkl**: This package improves math routines for science and engineering applications:

  ```
  pip3 install mkl
  ```

- **fitsne**: This package is to calculate t-SNE plots using a faster algorithm FIt-SNE:

  ```
  sudo apt install libfftw3-dev
  pip3 install fitsne
  ```

- **leiden**: This package provides Leiden clustering algorithm, besides the default Louvain algorithm in Pegasus:

  ```
  pip3 install leidenalg
  ```

Install via Miniconda

You can also choose to install pegasus in a dedicated Conda environment without affecting your OS settings.

1. Use the following commands to install a Miniconda on your system:

   ```
   sudo apt install wget
   wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh .
   export CONDA_PATH=/home/foo
   bash Miniconda3-latest-Linux-x86_64.sh -p $CONDA_PATH/miniconda3
   ```

(continues on next page)

(continued from previous page)

```
mv Miniconda3-latest-Linux-x86_64.sh $CONDA_PATH/miniconda3
source ~/.bashrc
```

Feel free to change `/home/foo` to your own directory on handling Miniconda.

In addition, remember to type `yes` when asked if you wish the installer to initialize Miniconda3 by running conda init.

2. Create a conda environment for pegasus. This tutorial uses `pegasus` as the environment name, but you are free to choose your own:

```
conda create -n pegasus -y python=3.8
```

Also notice that Python `3.8` is used in this tutorial. To choose a different version of Python, simply change the version number in the command above.

3. Enter `pegasus` environment by activating:

```
source activate pegasus
```

4. (Optional) Install the following dependency if you want *mkl* to for optimized math routines:

```
conda install -y -c anaconda numpy
```

5. Install pegasus:

```
pip install pegasuspy
```

6. (Optional) You can install the following optional features:

   - **fitsne**: Generate t-SNE plot by a faster algorithm FIt-SNE:

   ```
   sudo apt install libfftw3-dev
   pip install fitsne
   ```

   - **leiden**: Leiden clustering algorithm:

   ```
   pip install leidenalg
   ```

---

macOS

Prerequisites

First, install Homebrew by following the instruction on its website: https://brew.sh/. Then install the following dependencies:

```
brew install libxml2 libomp
```

And install macOS command line tools:

```
xcode-select --install
```

Next, you can install Pegasus system-wide by PyPI (see macOS Installation via PyPI), or within a Miniconda environment (see macOS Installation via Miniconda).

To use the Force-directed-layout (FLE) embedding feature, you'll need Java. You can either install Oracle JDK, or install OpenJDK via Homebrew:

```
brew cask install java
```

Install via PyPI

1. You need to install Python first:

   ```
   brew install python3
   ```

2. Starting from macOS Mojave (i.e. 10.14), *python-igraph*, one of the dependencies of Pegasus, needs to set the following environment variable before installation:

   ```
   export MACOSX_DEPLOYMENT_TARGET=10.14
   pip3 install python-igraph
   ```

You should change `10.14` to your macOS version number. For example, `10.15` is the number for Catalina.

3. Now install Pegasus:

   ```
   pip3 install pegasuspy
   ```

There are optional packages that you can install:

- **mkl**: This package improves math routines for science and engineering applications:

  ```
  pip3 install mkl
  ```

- **fitsne**: This package is to calculate t-SNE plots using a faster algorithm FIt-SNE. First, you need to install its dependency *fftw*:

  ```
  brew install fftw
  ```

Then install *fitsne* by:

```
pip3 install fitsne
```

- **leiden**: This package provides Leiden clustering algorithm, besides the default Louvain algorithm in Pegasus:

  ```
  pip3 install leidenalg
  ```

Install via Miniconda

1. Use the following commands to install a Miniconda on your system:

   ```
   brew install curl
   curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
   export CONDA_PATH=/Users/foo
   bash Miniconda3-latest-MacOSX-x86_64.sh -p $CONDA_PATH/miniconda3
   mv Miniconda3-latest-MacOSX-x86_64.sh $CONDA_PATH/miniconda3
   ```

Feel free to change `/Users/foo` to your own directory on handling Miniconda.

2. Create a conda environment for pegasus. This tutorial uses `pegasus` as the environment name, but you are free to choose your own:

```
conda create -n pegasus -y python=3.8
```

Also notice that Python `3.8` is used in this tutorial. To choose a different version of Python, simply change the version number in the command above.

3. Enter `pegasus` environment by activating:

```
conda activate pegasus
```

4. (Optional) Install the following dependency if you want *mkl* to for optimized math routines:

```
conda install -y -c anaconda numpy
```

5. **For macOS 10.14 or later:** for these macOS versions, you need to set the following environment variable before installing Pegasus:

```
export MACOSX_DEPLOYMENT_TARGET=10.15
```

where `10.15` is the version number for macOS Catalina. You should change it to your own OS version. For example, `10.14` is for macOS Mojave.

5. Install pegasus:

```
pip install pegasuspy
```

6. (Optional) You can install the following optional features:

   - **fitsne**: Generate t-SNE plot by a faster algorithm FIt-SNE:

```
conda install -y -c conda-forge fftw
pip install fitsne
```

   - **leiden**: Leiden clustering algorithm:

```
pip install leidenalg
```

Development Version

To install Pegasus development version directly from its GitHub respository, please do the following steps:

1. Install prerequisite libraries as mentioned in above sections.
2. Install Git. See here for how to install Git.
3. Use git to fetch repository source code, and install from it:

```
git clone https://github.com/klarman-cell-observatory/pegasus.git
cd pegasus
pip install -e .
```

where `-e` option of `pip` means to install in editing mode, so that your Pegasus installation will be automatically updated upon modifications in source code.

1.1.2 Use Pegasus as a command line tool

Pegasus can be used as a command line tool. Type:

```
pegasus -h
```

to see the help information:

```
Usage:
        pegasus <command> [<args>...]
        pegasus -h | --help
        pegasus -v | --version
```

`pegasus` has 9 sub-commands in 6 groups.

- Preprocessing:

  aggregate_matrix Aggregate sample count matrices into a single count matrix. It also enables users to import metadata into the count matrix.

- Demultiplexing:

  demuxEM Demultiplex cells/nuclei based on DNA barcodes for cell-hashing and nuclei-hashing data.

- Analyzing:

  cluster Perform first-pass analysis using the count matrix generated from 'aggregate_matrix'. This subcommand could perform low quality cell filtration, batch correction, variable gene selection, dimension reduction, diffusion map calculation, graph-based clustering, visualization. The final results will be written into zarr-formatted file, or h5ad file, which Seurat could load.

  de_analysis Detect markers for each cluster by performing differential expression analysis per cluster (within cluster vs. outside cluster). DE tests include Welch's t-test, Fisher's exact test, Mann-Whitney U test. It can also calculate AUROC values for each gene.

  find_markers Find markers for each cluster by training classifiers using LightGBM.

  annotate_cluster This subcommand is used to automatically annotate cell types for each cluster based on existing markers. Currently, it works for human/mouse immune/brain cells, etc.

- Plotting:

  plot Make static plots, which includes plotting tSNE, UMAP, and FLE embeddings by cluster labels and different groups.

- Web-based visualization:

  scp_output Generate output files for single cell portal.

- MISC:

  check_indexes Check CITE-Seq/hashing indexes to avoid index collision.

Quick guide

Suppose you have `example.csv` ready with the following contents:

```
Sample,Source,Platform,Donor,Reference,Location
sample_1,bone_marrow,NextSeq,1,GRCh38,/my_dir/sample_1/raw_feature_bc_matrices.h5
sample_2,bone_marrow,NextSeq,2,GRCh38,/my_dir/sample_2/raw_feature_bc_matrices.h5
sample_3,pbmc,NextSeq,1,GRCh38,/my_dir/sample_3/raw_gene_bc_matrices_h5.h5
sample_4,pbmc,NextSeq,2,GRCh38,/my_dir/sample_4/raw_gene_bc_matrices_h5.h5
```

You want to analyze all four samples but correct batch effects for bone marrow and pbmc samples separately. You can run the following commands:

```
pegasus aggregate_matrix --attributes Source,Platform,Donor example.csv example.aggr
pegasus cluster -p 20 --correct-batch-effect --batch-group-by Source --louvain --umap␣
↪example.aggr.zarr.zip example
pegasus de_analysis -p 20 --labels louvain_labels example.zarr.zip example.de.xlsx
pegasus annotate_cluster example.zarr.zip example.anno.txt
pegasus plot compo --groupby louvain_labels --condition Donor example.zarr.zip␣
↪example.composition.pdf
pegasus plot scatter --basis umap --attributes louvain_labels,Donor example.zarr.zip␣
↪example.umap.pdf
```

The above analysis will give you UMAP embedding and Louvain cluster labels in `example.zarr.zip`, along with differential expression analysis result stored in `example.de.xlsx`, and putative cluster-specific cell type annotation stored in `example.anno.txt`. You can investigate donor-specific effects by looking at `example.composition.pdf`. `example.umap.pdf` plotted UMAP colored by louvain_labels and Donor info side-by-side.

`pegasus aggregate_matrix`

The first step for single cell analysis is to generate one count matrix from cellranger's channel-specific count matrices. `pegasus aggregate_matrix` allows aggregating arbitrary matrices with the help of a *CSV* file.

Type:

```
pegasus aggregate_matrix -h
```

to see the usage information:

```
Usage:
        pegasus aggregate_matrix <csv_file> <output_name> [--restriction <restriction>
↪... --attributes <attributes> --default-reference <reference> --select-only-
↪singlets --min-genes <number>]
        pegasus aggregate_matrix -h
```

- Arguments:

  csv_file Input csv-formatted file containing information of each sc/snRNA-seq sample. This file must contain at least 2 columns - Sample, sample name and Location, location of the sample count matrix in either 10x v2/v3, DGE, mtx, csv, tsv or loom format. Additionally, an optional Reference column can be used to select samples generated from a same reference (e.g. mm10). If the count matrix is in either DGE, mtx, csv, tsv, or loom format, the value in this column will be used as the reference since the count matrix file does not contain reference name information. In addition, the Reference column can be used to aggregate count matrices generated from different

genome versions or gene annotations together under a unified reference. For example, if we have one matrix generated from mm9 and the other one generated from mm10, we can write mm9_10 for these two matrices in their Reference column. Pegasus will change their references to 'mm9_10' and use the union of gene symbols from the two matrices as the gene symbols of the aggregated matrix. For HDF5 files (e.g. 10x v2/v3), the reference name contained in the file does not need to match the value in this column. In fact, we use this column to rename references in HDF5 files. For example, if we have two HDF files, one generated from mm9 and the other generated from mm10. We can set these two files' Reference column value to 'mm9_10', which will rename their reference names into mm9_10 and the aggregated matrix will contain all genes from either mm9 or mm10. This renaming feature does not work if one HDF5 file contain multiple references (e.g. mm10 and GRCh38). See below for an example csv:

```
Sample,Source,Platform,Donor,Reference,Location
sample_1,bone_marrow,NextSeq,1,GRCh38,/my_dir/sample_1/raw_feature_bc_
↪matrices.h5
sample_2,bone_marrow,NextSeq,2,GRCh38,/my_dir/sample_2/raw_feature_bc_
↪matrices.h5
sample_3,pbmc,NextSeq,1,GRCh38,/my_dir/sample_3/raw_gene_bc_matrices_
↪h5.h5
sample_4,pbmc,NextSeq,2,GRCh38,/my_dir/sample_4/raw_gene_bc_matrices_
↪h5.h5
```

output_name The output file name.

- Options:

  -\-restriction <restriction>... Select channels that satisfy all restrictions. Each restriction takes the format of name:value,...,value or name:~value,..,value, where ~ refers to not. You can specifiy multiple restrictions by setting this option multiple times.

  -\-attributes <attributes> Specify a comma-separated list of outputted attributes. These attributes should be column names in the csv file.

  -\-default-reference <reference> If sample count matrix is in either DGE, mtx, csv, tsv or loom format and there is no Reference column in the csv_file, use <reference> as the reference.

  -\-select-only-singlets If we have demultiplexed data, turning on this option will make pegasus only include barcodes that are predicted as singlets.

  -\-remap-singlets <remap_string> Remap singlet names using <remap_string>, where <remap_string> takes the format "new_name_i:old_name_1,old_name_2;new_name_ii:old_name_3;...". For example, if we hashed 5 libraries from 3 samples sample1_lib1, sample1_lib2, sample2_lib1, sample2_lib2 and sample3, we can remap them to 3 samples using this string: "sample1:sample1_lib1,sample1_lib2;sample2:sample2_lib1,sample2_lib2". In this way, the new singlet names will be in metadata field with key 'assignment', while the old names will be kept in metadata field with key 'assignment.orig'.

  -\-subset-singlets <subset_string> If select singlets, only select singlets in the <subset_string>, which takes the format "name1,name2,...". Note that if –remap-singlets is specified, subsetting happens after remapping. For example, we can only select singlets from sampe 1 and 3 using "sample1,sample3".

  -\-min-genes <number> Only keep barcodes with at least <ngene> expressed genes.

  -\-max-genes <number> Only keep cells with less than <number> of genes.

  -\-min-umis <number> Only keep cells with at least <number> of UMIs.

-\-max-umis <number> Only keep cells with less than <number> of UMIs.

-\-mito-prefix <prefix> Prefix for mitochondrial genes. If multiple prefixes are provided, separate them by comma (e.g. "MT-,mt-").

-\-percent-mito <percent> Only keep cells with mitochondrial percent less than <percent>%. Only when both mito_prefix and percent_mito set, the mitochondrial filter will be triggered.

-\-no-append-sample-name Turn this option on if you do not want to append sample name in front of each sample's barcode (concatenated using '-').

-h, -\-help Print out help information.

- Outputs:

  output_name.zarr.zip A zipped Zarr file containing aggregated data.

- Examples:

```
pegasus aggregate_matrix --restriction Source:BM,CB --restriction Individual:1-8 -
↪-attributes Source,Platform Manton_count_matrix.csv aggr_data
```

---

pegasus demuxEM

Demultiplex cell-hashing/nucleus-hashing data.

Type:

```
pegasus demuxEM -h
```

to see the usage information:

```
Usage:
        pegasus demuxEM [options] <input_raw_gene_bc_matrices_h5> <input_hto_csv_file>
↪ <output_name>
        pegasus demuxEM -h | --help
        pegasus demuxEM -v | --version
```

- Arguments:

  input_raw_gene_bc_matrices_h5 Input raw RNA expression matrix in 10x hdf5 format. It is important to feed raw (unfiltered) count matrix, as demuxEM uses it to estimate the background information.

  input_hto_csv_file Input HTO (antibody tag) count matrix in CSV format.

  output_name Output name. All outputs will use it as the prefix.

- Options:

  -p <number>, -\-threads <number> Number of threads. [default: 1]

  -\-genome <genome> Reference genome name. If not provided, we will infer it from the expression matrix file.

  -\-alpha-on-samples <alpha> The Dirichlet prior concentration parameter (alpha) on samples. An alpha value < 1.0 will make the prior sparse. [default: 0.0]

  -\-min-num-genes <number> We only demultiplex cells/nuclei with at least <number> of expressed genes. [default: 100]

-\-min-num-umis <number> We only demultiplex cells/nuclei with at least <number> of UMIs. [default: 100]

-\-min-signal-hashtag <count> Any cell/nucleus with less than <count> hashtags from the signal will be marked as unknown. [default: 10.0]

-\-random-state <seed> The random seed used in the KMeans algorithm to separate empty ADT droplets from others. [default: 0]

-\-generate-diagnostic-plots Generate a series of diagnostic plots, including the background/signal between HTO counts, estimated background probabilities, HTO distributions of cells and non-cells etc.

-\-generate-gender-plot <genes> Generate violin plots using gender-specific genes (e.g. Xist). <gene> is a comma-separated list of gene names.

-v, -\-version Show DemuxEM version.

-h, -\-help Print out help information.

- Outputs:

  output_name_demux.zarr.zip RNA expression matrix with demultiplexed sample identities in Zarr format.

  output_name.out.demuxEM.zarr.zip DemuxEM-calculated results in Zarr format, containing two datasets, one for HTO and one for RNA.

  output_name.ambient_hashtag.hist.pdf Optional output. A histogram plot depicting hashtag distributions of empty droplets and non-empty droplets.

  output_name.background_probabilities.bar.pdf Optional output. A bar plot visualizing the estimated hashtag background probability distribution.

  output_name.real_content.hist.pdf Optional output. A histogram plot depicting hashtag distributions of not-real-cells and real-cells as defined by total number of expressed genes in the RNA assay.

  output_name.rna_demux.hist.pdf Optional output. A histogram plot depicting RNA UMI distribution for singlets, doublets and unknown cells.

  output_name.gene_name.violin.pdf Optional outputs. Violin plots depicting gender-specific gene expression across samples. We can have multiple plots if a gene list is provided in '–generate-gender-plot' option.

- Examples:

```
pegasus demuxEM -p 8 --generate-diagnostic-plots sample_raw_gene_bc_matrices.h5␣
↪sample_hto.csv sample_output
```

pegasus cluster

Once we collected the count matrix in 10x (`example_10x.h5`) or Zarr (`example.zarr.zip`) format, we can perform single cell analysis using `pegasus cluster`.

Type:

```
pegasus cluster -h
```

to see the usage information:

```
Usage:
        pegasus cluster [options] <input_file> <output_name>
        pegasus cluster -h
```

- Arguments:

  input_file Input file in either 'zarr', 'h5ad', 'loom', '10x', 'mtx', 'csv', 'tsv' or 'fcs' format. If first-pass analysis has been performed, but you want to run some additional analysis, you could also pass a zarr-formatted file.

  output_name Output file name. All outputs will use it as the prefix.

- Options:

  -p <number>, -\-threads <number> Number of threads. [default: 1]

  -\-processed Input file is processed. Assume quality control, data normalization and log transformation, highly variable gene selection, batch correction/PCA and kNN graph building is done.

  -\-channel <channel_attr> Use <channel_attr> to create a 'Channel' column metadata field. All cells within a channel are assumed to come from a same batch.

  -\-black-list <black_list> Cell barcode attributes in black list will be popped out. Format is "attr1,attr2,...,attrn".

  -\-select-singlets Only select DemuxEM-predicted singlets for analysis.

  -\-remap-singlets <remap_string> Remap singlet names using <remap_string>, where <remap_string> takes the format "new_name_i:old_name_1,old_name_2;new_name_ii:old_name_3;...". For example, if we hashed 5 libraries from 3 samples sample1_lib1, sample1_lib2, sample2_lib1, sample2_lib2 and sample3, we can remap them to 3 samples using this string: "sample1:sample1_lib1,sample1_lib2;sample2:sample2_lib1,sample2_lib2". In this way, the new singlet names will be in metadata field with key 'assignment', while the old names will be kept in metadata field with key 'assignment.orig'.

  -\-subset-singlets <subset_string> If select singlets, only select singlets in the <subset_string>, which takes the format "name1,name2,...". Note that if –remap-singlets is specified, subsetting happens after remapping. For example, we can only select singlets from sampe 1 and 3 using "sample1,sample3".

  -\-genome <genome_name> If sample count matrix is in either DGE, mtx, csv, tsv or loom format, use <genome_name> as the genome reference name.

  -\-focus <keys> Focus analysis on Unimodal data with <keys>. <keys> is a comma-separated list of keys. If None, the self._selected will be the focused one.

  -\-append <key> Append Unimodal data <key> to any <keys> in `--focus`.

  -\-output-loom Output loom-formatted file.

  -\-output-h5ad Output h5ad-formatted file.

-\-min-genes <number> Only keep cells with at least <number> of genes. [default: 500]

-\-max-genes <number> Only keep cells with less than <number> of genes. [default: 6000]

-\-min-umis <number> Only keep cells with at least <number> of UMIs.

-\-max-umis <number> Only keep cells with less than <number> of UMIs.

-\-mito-prefix <prefix> Prefix for mitochondrial genes. Can provide multiple prefixes for multiple organisms (e.g. "MT-" means to use "MT-", "GRCh38:MT-,mm10:mt-,MT-" means to use "MT-" for GRCh38, "mt-" for mm10 and "MT-" for all other organisms). [default: GRCh38:MT-,mm10:mt-,MT-]

-\-percent-mito <ratio> Only keep cells with mitochondrial percent less than <percent>%. [default: 20.0]

-\-gene-percent-cells <ratio> Only use genes that are expressed in at least <percent>% of cells to select variable genes. [default: 0.05]

-\-output-filtration-results Output filtration results as a spreadsheet.

-\-plot-filtration-results Plot filtration results as PDF files.

-\-plot-filtration-figsize <figsize> Figure size for filtration plots. <figsize> is a comma-separated list of two numbers, the width and height of the figure (e.g. 6,4).

-\-min-genes-before-filtration <number> If raw data matrix is input, empty barcodes will dominate pre-filtration statistics. To avoid this, for raw data matrix, only consider barcodes with at lease <number> genes for pre-filtration condition. [default: 100]

-\-counts-per-cell-after <number> Total counts per cell after normalization. [default: 1e5]

-\-select-hvf-flavor <flavor> Highly variable feature selection method. <flavor> can be 'pegasus' or 'Seurat'. [default: pegasus]

-\-select-hvf-ngenes <nfeatures> Select top <nfeatures> highly variable features. If <flavor> is 'Seurat' and <ngenes> is 'None', select HVGs with z-score cutoff at 0.5. [default: 2000]

-\-no-select-hvf Do not select highly variable features.

-\-plot-hvf Plot highly variable feature selection.

-\-correct-batch-effect Correct for batch effects.

-\-correction-method <method> Batch correction method, can be either 'L/S' for location/scale adjustment algorithm (Li and Wong. The analysis of Gene Expression Data 2003) or 'harmony' for Harmony (Korsunsky et al. Nature Methods 2019) or 'scanorama' for Scanorama (Hie et al. Nature Biotechnology 2019). [default: harmony]

-\-batch-group-by <expression> Batch correction assumes the differences in gene expression between channels are due to batch effects. However, in many cases, we know that channels can be partitioned into several groups and each group is biologically different from others. In this case, we will only perform batch correction for channels within each group. This option defines the groups. If <expression> is None, we assume all channels are from one group. Otherwise, groups are defined according to <expression>. <expression> takes the form of either 'attr', or 'attr1+attr2+...+attrn', or 'attr=value11,...,value1n_1;value21,...,value2n_2;...;valuem1,...,valuemn_m'. In the first form, 'attr' should be an existing sample attribute, and groups are defined by 'attr'. In the second form, 'attr1',...,'attrn' are n existing sample attributes and groups are defined by the Cartesian product of these n attributes. In the last form, there will be m + 1 groups. A cell belongs to group i (i > 0) if and only if its sample attribute 'attr' has a value among valuei1,...,valuein_i. A cell belongs to group 0 if it does not belong to any other groups.

-\-harmony-nclusters <nclusters> Number of clusters used for Harmony batch correction.

-\-random-state <seed> Random number generator seed. [default: 0]

-\-temp-folder <temp_folder> Joblib temporary folder for memmapping numpy arrays.

-\-calc-signature-scores <sig_list> Calculate signature scores for gene sets in <sig_list>. <sig_list> is a comma-separated list of strings. Each string should either be a <GMT_file> or one of 'cell_cycle_human', 'cell_cycle_mouse', 'gender_human', 'gender_mouse', 'mitochondrial_genes_human', 'mitochondrial_genes_mouse', 'ribosomal_genes_human' and 'ribosomal_genes_mouse'.

-\-pca-n <number> Number of principal components. [default: 50]

-\-pca-robust Use 'arpack' instead of 'randomized' as svd_solver for large sparse matrices. It will take longer time to compute PCs, but the results are more numerically stable.

-\-knn-K <number> Number of nearest neighbors for building kNN graph. [default: 100]

-\-knn-full-speed For the sake of reproducibility, we only run one thread for building kNN indices. Turn on this option will allow multiple threads to be used for index building. However, it will also reduce reproducibility due to the racing between multiple threads.

-\-kBET Calculate kBET.

-\-kBET-batch <batch> kBET batch keyword.

-\-kBET-alpha <alpha> kBET rejection alpha. [default: 0.05]

-\-kBET-K <K> kBET K. [default: 25]

-\-diffmap Calculate diffusion maps.

-\-diffmap-ndc <number> Number of diffusion components. [default: 100]

-\-diffmap-solver <solver> Solver for eigen decomposition, either 'randomized' or 'eigsh'. [default: eigsh]

-\-diffmap-maxt <max_t> Maximum time stamp to search for the knee point. [default: 5000]

-\-diffmap-to-3d If map diffusion map into 3D space using PCA.

-\-calculate-pseudotime <roots> Calculate diffusion-based pseudotimes based on <roots>. <roots> should be a comma-separated list of cell barcodes.

-\-louvain Run louvain clustering algorithm.

-\-louvain-resolution <resolution> Resolution parameter for the louvain clustering algorithm. [default: 1.3]

-\-louvain-class-label <label> Louvain cluster label name in result. [default: louvain_labels]

-\-leiden Run leiden clustering algorithm.

-\-leiden-resolution <resolution> Resolution parameter for the leiden clustering algorithm. [default: 1.3]

-\-leiden-niter <niter> Number of iterations of running the Leiden algorithm. If <niter> is negative, run Leiden iteratively until no improvement. [default: -1]

-\-leiden-class-label <label> Leiden cluster label name in result. [default: leiden_labels]

-\-spectral-louvain Run spectral-louvain clustering algorithm.

-\-spectral-louvain-basis <basis> Basis used for KMeans clustering. Can be 'pca' or 'diffmap'. If 'diffmap' is not calculated, use 'pca' instead. [default: diffmap]

-\-spectral-louvain-nclusters <number> Number of first level clusters for Kmeans. [default: 30]

-\-spectral-louvain-nclusters2 <number> Number of second level clusters for Kmeans. [default: 50]

-\-spectral-louvain-ninit <number> Number of Kmeans tries for first level clustering. Default is the same as scikit-learn Kmeans function. [default: 10]

-\-spectral-louvain-resolution <resolution>. Resolution parameter for louvain. [default: 1.3]

-\-spectral-louvain-class-label <label> Spectral-louvain label name in result. [default: spectral_louvain_labels]

-\-spectral-leiden Run spectral-leiden clustering algorithm.

-\-spectral-leiden-basis <basis> Basis used for KMeans clustering. Can be 'pca' or 'diffmap'. If 'diffmap' is not calculated, use 'pca' instead. [default: diffmap]

-\-spectral-leiden-nclusters <number> Number of first level clusters for Kmeans. [default: 30]

-\-spectral-leiden-nclusters2 <number> Number of second level clusters for Kmeans. [default: 50]

-\-spectral-leiden-ninit <number> Number of Kmeans tries for first level clustering. Default is the same as scikit-learn Kmeans function. [default: 10]

-\-spectral-leiden-resolution <resolution> Resolution parameter for leiden. [default: 1.3]

-\-spectral-leiden-class-label <label> Spectral-leiden label name in result. [default: spectral_leiden_labels]

-\-tsne Run FIt-SNE package to compute t-SNE embeddings for visualization.

-\-tsne-perplexity <perplexity> t-SNE's perplexity parameter. [default: 30]

-\-tsne-initialization <choice> <choice> can be either 'random' or 'pca'. 'random' refers to random initialization. 'pca' refers to PCA initialization as described in (CITE Kobak et al. 2019) [default: pca]

-\-umap Run umap for visualization.

-\-umap-K <K> K neighbors for umap. [default: 15]

-\-umap-min-dist <number> Umap parameter. [default: 0.5]

-\-umap-spread <spread> Umap parameter. [default: 1.0]

-\-fle Run force-directed layout embedding.

-\-fle-K <K> K neighbors for building graph for FLE. [default: 50]

-\-fle-target-change-per-node <change> Target change per node to stop forceAtlas2. [default: 2.0]

-\-fle-target-steps <steps> Maximum number of iterations before stopping the forceAtlas2 algoritm. [default: 5000]

-\-fle-memory <memory> Memory size in GB for the Java FA2 component. [default: 8]

-\-net-down-sample-fraction <frac> Down sampling fraction for net-related visualization. [default: 0.1]

-\-net-down-sample-K <K> Use <K> neighbors to estimate local density for each data point for down sampling. [default: 25]

-\-net-down-sample-alpha <alpha> Weighted down sample, proportional to radius^alpha. [default: 1.0]

-\-net-regressor-L2-penalty <value> L2 penalty parameter for the deep net regressor. [default: 0.1]

-\-net-umap Run net umap for visualization.

-\-net-umap-polish-learning-rate <rate> After running the deep regressor to predict new coordinate, what is the learning rate to use to polish the coordinates for UMAP. [default: 1.0]

-\-net-umap-polish-nepochs <nepochs> Number of iterations for polishing UMAP run. [default: 40]

-\-net-umap-out-basis <basis> Output basis for net-UMAP. [default: net_umap]

-\-net-fle Run net FLE.

-\-net-fle-polish-target-steps <steps> After running the deep regressor to predict new coordinate, what is the number of force atlas 2 iterations. [default: 1500]

-\-net-fle-out-basis <basis> Output basis for net-FLE. [default: net_fle]

-\-infer-doublets Infer doublets using the method described here. Obs attribute 'doublet_score' stores Scrublet-like doublet scores and attribute 'demux_type' stores 'doublet/singlet' assignments.

-\-expected-doublet-rate <rate> The expected doublet rate per sample. By default, calculate the expected rate based on number of cells from the 10x multiplet rate table.

-\-dbl-cluster-attr <attr> <attr> refers to a cluster attribute containing cluster labels (e.g. 'louvain_labels'). Doublet clusters will be marked based on <attr> with the following criteria: passing the Fisher's exact test and having >= 50% of cells identified as doublets. By default, the first computed cluster attribute in the list of leiden, louvain, spectral_ledein and spectral_louvain is used.

-\-citeseq Input data contain both RNA and CITE-Seq modalities. This will set –focus to be the RNA modality and –append to be the CITE-Seq modality. In addition, 'ADT-' will be added in front of each antibody name to avoid name conflict with genes in the RNA modality.

-\-citeseq-umap For high quality cells kept in the RNA modality, generate a UMAP based on their antibody expression.

-\-citeseq-umap-exclude <list> <list> is a comma-separated list of antibodies to be excluded from the UMAP calculation (e.g. Mouse-IgG1,Mouse-IgG2a).

-h, -\-help Print out help information.

- Outputs:

output_name.zarr.zip Output file in Zarr format. To load this file in python, use `import pegasus; data = pegasus.read_input('output_name.zarr.zip')`. The log-normalized expression matrix is stored in `data.X` as a CSR-format sparse matrix. The `obs` field contains cell related attributes, including clustering results. For example, `data.obs_names` records cell barcodes; `data.obs['Channel']` records the channel each cell comes from; `data.obs['n_genes']`, `data.obs['n_counts']`, and `data.obs['percent_mito']` record the number of expressed genes, total UMI count, and mitochondrial rate for each cell respectively; `data.obs['louvain_labels']` and `data.obs['approx_louvain_labels']` record each cell's cluster labels using different clustring algorithms; `data.obs['pseudo_time']` records the inferred pseudotime for each cell. The `var` field contains gene related attributes. For example, `data.var_names` records gene symbols, `data.var['gene_ids']` records Ensembl gene IDs, and `data.var['selected']` records selected variable genes. The `obsm` field records embedding coordiates. For example, `data.obsm['X_pca']` records PCA coordinates, `data.obsm['X_tsne']` records tSNE coordinates, `data.obsm['X_umap']`

records UMAP coordinates, `data.obsm['X_diffmap']` records diffusion map coordinates, `data.obsm['X_diffmap_pca']` records the first 3 PCs by projecting the diffusion components using PCA, and `data.obsm['X_fle']` records the force-directed layout coordinates from the diffusion components. The `uns` field stores other related information, such as reference genome (`data.uns['genome']`). This file can be loaded into R and converted into a Seurat object.

output_name.<group>.h5ad Optional output. Only exists if '–output-h5ad' is set. Results in h5ad format per focused <group>. This file can be loaded into R and converted into a Seurat object.

output_name.<group>.loom Optional output. Only exists if '–output-loom' is set. Results in loom format per focused <group>.

output_name.<group>.filt.xlsx Optional output. Only exists if '–output-filtration-results' is set. Filtration statistics per focused <group>. This file has two sheets — Cell filtration stats and Gene filtration stats. The first sheet records cell filtering results and it has 10 columns: Channel, channel name; kept, number of cells kept; median_n_genes, median number of expressed genes in kept cells; median_n_umis, median number of UMIs in kept cells; median_percent_mito, median mitochondrial rate as UMIs between mitochondrial genes and all genes in kept cells; filt, number of cells filtered out; total, total number of cells before filtration, if the input contain all barcodes, this number is the cells left after '–min-genes-on-raw' filtration; median_n_genes_before, median expressed genes per cell before filtration; median_n_umis_before, median UMIs per cell before filtration; median_percent_mito_before, median mitochondrial rate per cell before filtration. The channels are sorted in ascending order with respect to the number of kept cells per channel. The second sheet records genes that failed to pass the filtering. This sheet has 3 columns: gene, gene name; n_cells, number of cells this gene is expressed; percent_cells, the fraction of cells this gene is expressed. Genes are ranked in ascending order according to number of cells the gene is expressed. Note that only genes not expressed in any cell are removed from the data. Other filtered genes are marked as non-robust and not used for TPM-like normalization.

output_name.<group>.filt.gene.pdf Optional output. Only exists if '–plot-filtration-results' is set. This file contains violin plots contrasting gene count distributions before and after filtration per channel per focused <group>.

output_name.<group>.filt.UMI.pdf Optional output. Only exists if '–plot-filtration-results' is set. This file contains violin plots contrasting UMI count distributions before and after filtration per channel per focused <group>.

output_name.<group>.filt.mito.pdf Optional output. Only exists if '–plot-filtration-results' is set. This file contains violin plots contrasting mitochondrial rate distributions before and after filtration per channel per focused <group>.

output_name.<group>.hvf.pdf Optional output. Only exists if '–plot-hvf' is set. This file contains a scatter plot describing the highly variable gene selection procedure per focused <group>.

output_name.<group>.<channel>.dbl.png Optional output. Only exists if '–infer-doublets' is set. Each figure consists of 4 panels showing diagnostic plots for doublet inference. If there is only one channel in <group>, file name becomes output_name.<group>.dbl.png.

- Examples:

```
pegasus cluster -p 20 --correct-batch-effect --louvain --fitsne example_10x.h5␣
↪example_out
pegasus cluster -p 20 --leiden --umap --net-fle example.zarr.zip example_out
```

pegasus de_analysis

Once we have the clusters, we can detect markers using `pegasus de_analysis`. We will calculate Mann-Whitney U test and AUROC values by default.

Type:

```
pegasus de_analysis -h
```

to see the usage information:

```
Usage:
        pegasus de_analysis [options] (--labels <attr>) <input_data_file> <output_
↪spreadsheet>
        pegasus de_analysis -h
```

- Arguments:

  input_data_file Single cell data with clustering calculated. DE results would be written back.

  output_spreadsheet Output spreadsheet with DE results.

- Options:

  -\-labels <attr> <attr> used as cluster labels. [default: louvain_labels]

  -p <threads> Use <threads> threads. [default: 1]

  -\-de-key <key> Store DE results into AnnData varm with key = <key>. [default: de_res]

  -\-t Calculate Welch's t-test.

  -\-fisher Calculate Fisher's exact test.

  -\-temp-folder <temp_folder> Joblib temporary folder for memmapping numpy arrays.

  -\-alpha <alpha> Control false discovery rate at <alpha>. [default: 0.05]

  -\-ndigits <ndigits> Round non p-values and q-values to <ndigits> after decimal point in the excel. [default: 3]

  -\-quiet Do not show detailed intermediate outputs.

  -h, -\-help Print out help information.

- Outputs:

  input_data_file DE results would be written back to the 'varm' field with name set by '–de-key <key>'.

  output_spreadsheet An excel spreadsheet containing DE results. Each cluster has two tabs in the spreadsheet. One is for up-regulated genes and the other is for down-regulated genes. If DE was performed on conditions within each cluster. Each cluster will have number of conditions tabs and each condition tab contains two spreadsheet: up for up-regulated genes and down for down-regulated genes.

- Examples:

```
pegasus de_analysis -p 26 --labels louvain_labels --t --fisher example.zarr.zip␣
↪example_de.xlsx
```

pegasus find_markers

Once we have the DE results, we can optionally find cluster-specific markers with gradient boosting using `pegasus find_markers`.

Type:

```
pegasus find_markers -h
```

to see the usage information:

```
Usage:
        pegasus find_markers [options] <input_data_file> <output_spreadsheet>
        pegasus find_markers -h
```

- Arguments:

  input_h5ad_file Single cell data after running the de_analysis.

  output_spreadsheet Output spreadsheet with LightGBM detected markers.

- Options:

  -p <threads> Use <threads> threads. [default: 1]

  -\-labels <attr> <attr> used as cluster labels. [default: louvain_labels]

  -\-de-key <key> Key for storing DE results in 'varm' field. [default: de_res]

  -\-remove-ribo Remove ribosomal genes with either RPL or RPS as prefixes.

  -\-min-gain <gain> Only report genes with a feature importance score (in gain) of at least <gain>. [default: 1.0]

  -\-random-state <seed> Random state for initializing LightGBM and KMeans. [default: 0]

  -h, -\-help Print out help information.

- Outputs:

  output_spreadsheet An excel spreadsheet containing detected markers. Each cluster has one tab in the spreadsheet and each tab has six columns, listing markers that are strongly up-regulated, weakly up-regulated, down-regulated and their associated LightGBM gains.

- Examples:

```
pegasus find_markers --labels louvain_labels --remove-ribo --min-gain 10.0 -p 10␣
↪example.zarr.zip example.markers.xlsx
```

---

pegasus annotate_cluster

Once we have the DE results, we could optionally identify putative cell types for each cluster using `pegasus annotate_cluster`. This command has two forms: the first form generates putative annotations, and the second form write annotations into the Zarr object.

Type:

```
pegasus annotate_cluster -h
```

to see the usage information:

```
Usage:
        pegasus annotate_cluster [--marker-file <file> --de-test <test> --de-alpha
↪<alpha> --de-key <key> --minimum-report-score <score> --do-not-use-non-de-genes]
↪<input_data_file> <output_file>
        pegasus annotate_cluster --annotation <annotation_string> <input_data_file>
        pegasus annotate_cluster -h
```

- Arguments:

  input_data_file Single cell data with DE analysis done by `pegasus de_analysis`.

  output_file Output annotation file.

- Options:

  -\-markers <str> <str> is a comma-separated list. Each element in the list either refers to a JSON file containing legacy markers, or 'human_immune'/'mouse_immune'/'human_brain'/'mouse_brain'/'human_lung' for predefined markers. [default: human_immune]

  -\-de-test <test> DE test to use to infer cell types. [default: mwu]

  -\-de-alpha <alpha> False discovery rate to control family-wise error rate. [default: 0.05]

  -\-de-key <key> Keyword where the DE results store in 'varm' field. [default: de_res]

  -\-minimum-report-score <score> Minimum cell type score to report a potential cell type. [default: 0.5]

  -\-do-not-use-non-de-genes Do not count non DE genes as down-regulated.

  -\-annotation <annotation_string> Write cell type annotations in <annotation_string> into <input_data_file>. <annotation_string> has this format: `'anno_name:clust_name:anno_1;anno_2;...;anno_n'`, where `anno_name` is the annotation attribute in the Zarr object, `clust_name` is the attribute with cluster ids, and `anno_i` is the annotation for cluster i.

  -h, -\-help Print out help information.

- Outputs:

  output_file This is a text file. For each cluster, all its putative cell types are listed in descending order of the cell type score. For each putative cell type, all markers support this cell type are listed. If one putative cell type has cell subtypes, all subtypes will be listed under this cell type.

- Examples:

```
pegasus annotate_cluster example.zarr.zip example.anno.txt
pegasus annotate_cluster --markers human_immune,human_lung lung.zarr.zip lung.
↪anno.txt
pegasus annotate_cluster --annotation "anno:louvain_labels:T cells;B cells;NK␣
↪cells;Monocytes" example.zarr.zip
```

---

pegasus plot

We can make a variety of figures using `pegasus plot`.

Type:

```
pegasus plot -h
```

to see the usage information:

```
Usage:
        pegasus plot [options] [--restriction <restriction>...] [--palette <palette>..
↪.] <plot_type> <input_file> <output_file>
        pegasus plot -h
```

- Arguments:

  plot_type Plot type, either 'scatter' for scatter plots or 'compo' for composition plots.

  input_file Single cell data in Zarr or H5ad format.

  output_file Output image file.

- Options:

  -\-dpi <dpi> DPI value for the figure. [default: 500]

  -\-basis <basis> Basis for 2D plotting, chosen from 'tsne', 'fitsne', 'umap', 'pca', 'fle', 'diffmap_pca', 'net_tsne', 'net_umap' or 'net_fle'. [default: umap]

  -\-attributes <attrs> <attrs> is a comma-separated list of attributes to color the basis. This option is only used in 'scatter'.

  -\-restriction <restriction>... Set restriction if you only want to plot a subset of data. Multiple <restriction> strings are allowed. Each <restriction> takes the format of 'attr:value,value', or 'attr:~value,value..' which means excluding values. This option is used in 'composition' and 'scatter'.

  -\-alpha <alpha> Point transparent parameter. Can be a single value or a list of values separated by comma used for each attribute in <attrs>.

  -\-legend-loc <str> Legend location, can be either "right margin" or "on data". If a list is provided, set 'legend_loc' for each attribute in 'attrs' separately. [default: "right margin"]

  -\-palette <str> Used for setting colors for every categories in categorical attributes. Multiple <palette> strings are allowed. Each string takes the format of 'attr:color1,color2,...,colorn'. 'attr' is the categorical attribute and 'color1' - 'colorn' are the colors for each category in 'attr' (e.g. 'cluster_labels:black,blue,red,...,yellow'). If there is only one categorical attribute in 'attrs', `palletes` can be set as a single string and the 'attr' keyword can be omitted (e.g. "blue,yellow,red").

  -\-show-background Show points that are not selected as gray.

  -\-nrows <nrows> Number of rows in the figure. If not set, pegasus will figure it out automatically.

  -\-ncols <ncols> Number of columns in the figure. If not set, pegasus will figure it out automatically.

  -\-panel-size <sizes> Panel size in inches, w x h, separated by comma. Note that margins are not counted in the sizes. For composition, default is (6, 4). For scatter plots, default is (4, 4).

  -\-left <left> Figure's left margin in fraction with respect to panel width.

  -\-bottom <bottom> Figure's bottom margin in fraction with respect to panel height.

-\-wspace <wspace> Horizontal space between panels in fraction with respect to panel width.

-\-hspace <hspace> Vertical space between panels in fraction with respect to panel height.

-\-groupby <attr> Use <attr> to categorize the cells for the composition plot, e.g. cell type.

-\-condition <attr> Use <attr> to calculate frequency within each category defined by '–groupby' for the composition plot, e.g. donor.

-\-style <style> Composition plot styles. Can be either 'frequency' or 'normalized'. [default: normalized]

-h, -\-help Print out help information.

Examples:

```
pegasus plot scatter --basis tsne --attributes louvain_labels,Donor example.h5ad␣
↪scatter.pdf
pegasus plot compo --groupby louvain_labels --condition Donor example.zarr.zip compo.
↪pdf
```

---

pegasus scp_output

If we want to visualize analysis results on single cell portal (SCP), we can generate required files for SCP using this subcommand.

Type:

```
pegasus scp_output -h
```

to see the usage information:

```
Usage:
        pegasus scp_output <input_data_file> <output_name>
        pegasus scp_output -h
```

- Arguments:

  input_data_file Analyzed single cell data in zarr format.

  output_name Name prefix for all outputted files.

- Options:

  -\-dense Output dense expression matrix instead.

  -\-round-to <ndigit> Round expression to <ndigit> after the decimal point. [default: 2]

  -h, -\-help Print out help information.

- Outputs:

  output_name.scp.metadata.txt, output_name.scp.barcodes.tsv, output_name.scp.genes.tsv, output_name.scp.matrix.m
  Files that single cell portal needs.

- Examples:

  ```
  pegasus scp_output example.zarr.zip example
  ```

---

pegasus check_indexes

If we run CITE-Seq or any kind of hashing, we need to make sure that the library indexes of CITE-Seq/hashing do not collide with 10x's RNA indexes. This command can help us to determine which 10x index sets we should use.

Type:

```
pegasus check_indexes -h
```

to see the usage information:

```
Usage:
        pegasus check_indexes [--num-mismatch <mismatch> --num-report <report>]
↪<index_file>
        pegasus check_indexes -h
```

- Arguments:

  index_file Index file containing CITE-Seq/hashing index sequences. One sequence per line.

- Options:

  -\-num-mismatch <mismatch> Number of mismatch allowed for each index sequence. [default: 1]

  -\-num-report <report> Number of valid 10x indexes to report. Default is to report all valid indexes. [default: 9999]

  -h, -\-help Print out help information.

- Outputs:

  Up to <report> number of valid 10x indexes will be printed out to standard output.

- Examples:

  ```
  pegasus check_indexes --num-report 8 index_file.txt
  ```

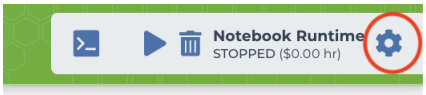
1.1.3 Use Pegasus on Terra Notebook

You need to first have a Terra account.

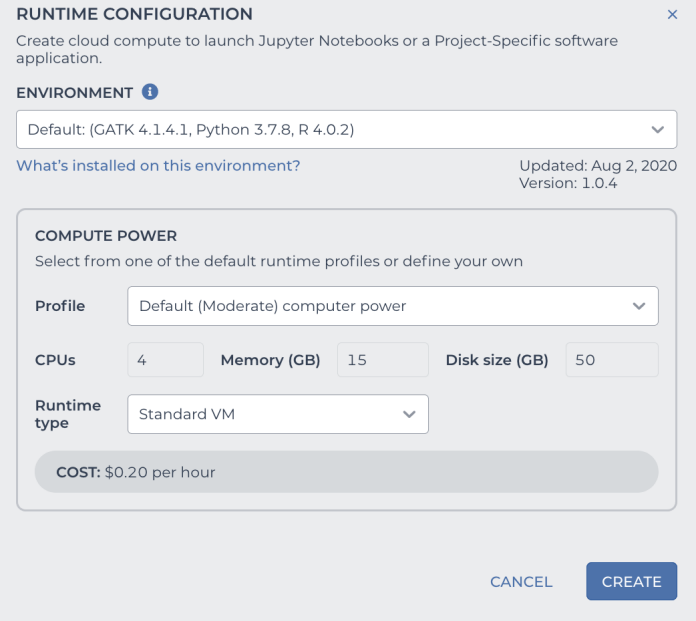
1. Start Notebook Runtime on Terra

The first time when you use Terra notebook, you need to create a runtime environment.

On top-right panel of your workspace, click the following button within red circle:



Then you'll need to set the configuration of your runtime environment in the pop-out dialog (see image below):

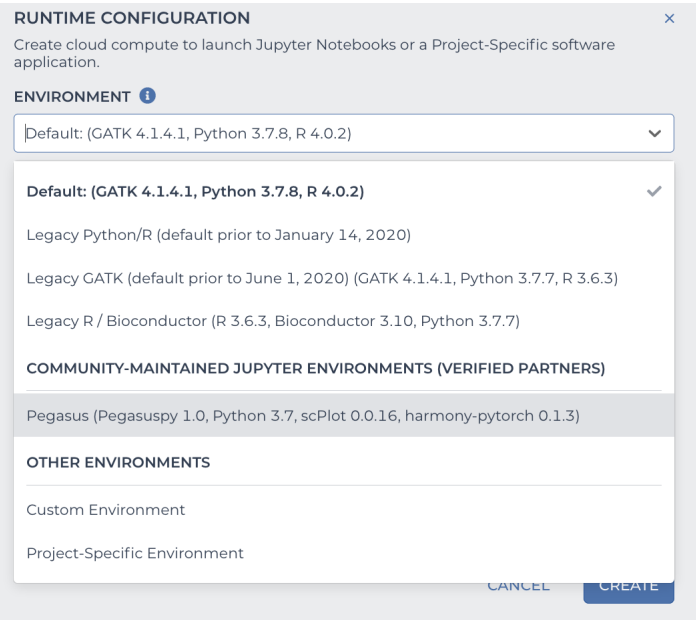


There are 2 ways of creating a runtime environment for Pegasus.

1.1. Create from Terra official environment

Terra maintains a list of runtimes for users to quickly create an environment.

In *ENVIRONMENT* field, select `Pegasus` from the drop-down menu:



After that set other fields in the pop-out dialog:

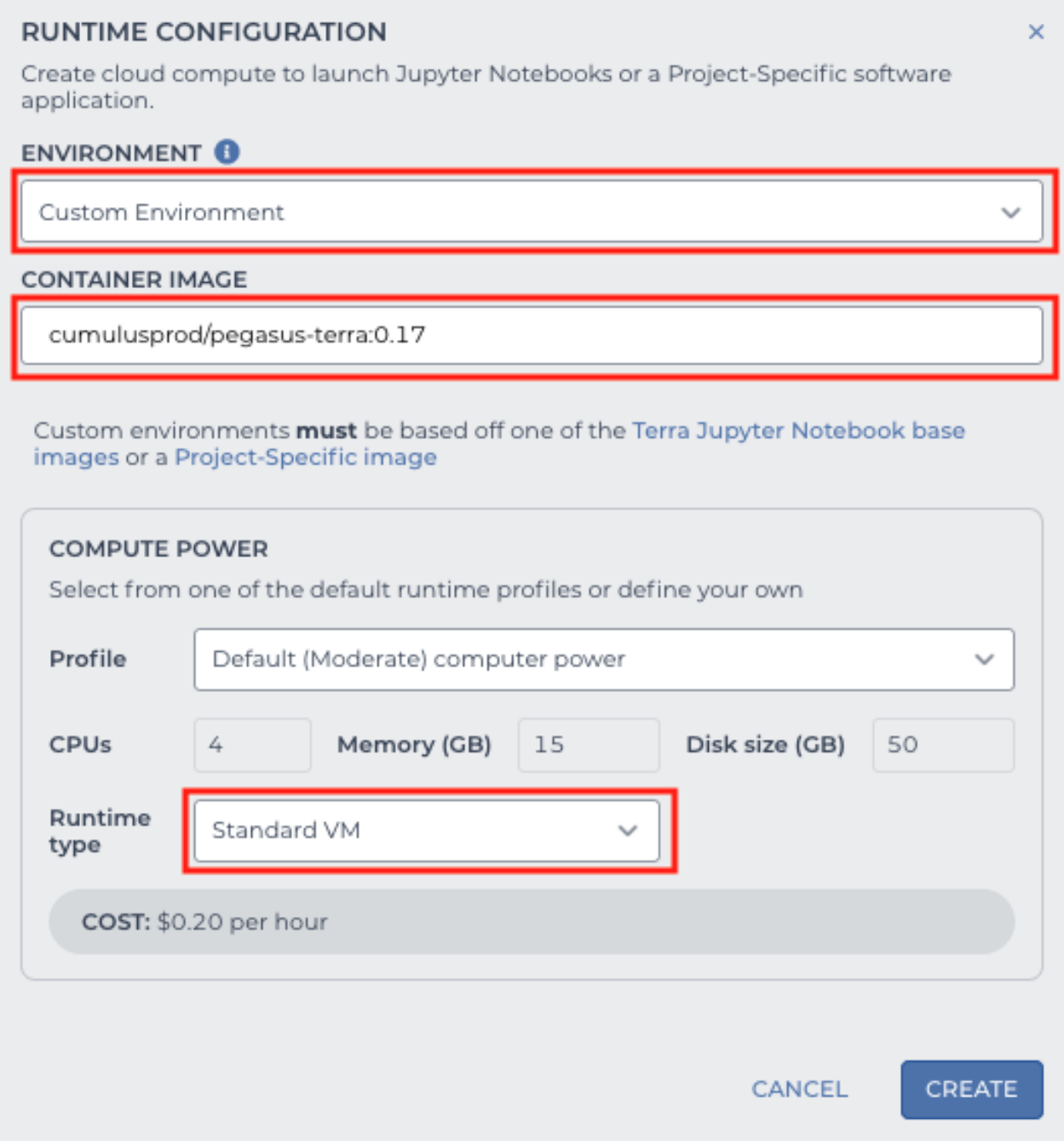
- In *Runtime type* field, choose `Standard VM` (see the third red rectangle above), as this is the cheapest type and is enough to use Pegasus.
- In *COMPUTE POWER* field, you can set the computing resources you want to use.

Now click `CREATE` button to start the creation. After waiting for 1-2 minutes, your runtime environment will be ready to use, and it's started automatically.

In this way, you can create your environment with the most updated version of Pegasus.

1.2. Create from custom environment

Another is to create your environment directly from *Pegasus* docker image which is maintained by Cumulus team. If you want to use an older version of Pegasus, this is way to create your environment.



- In *ENVIRONMENT* field, choose `Custom Environment` (see the first red rectangle above).

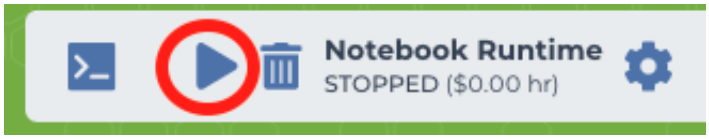
- In *CONTAINER IMAGE* field, type `cumulusprod/pegasus-terra:<version>` (see the second red rectangle above), where `<version>` should be chosen from this list. All the tags are for different versions of Pegasus.
- In *Runtime type* field, choose `Standard VM` (see the third red rectangle above), as this is the cheapest type and is enough to use Pegasus.
- In *COMPUTE POWER* field, you can set the computing resources you want to use.

Note: Sometimes DockerHub server may go down. In this situation, we also provide backup docker images on RedHat docker registry. Simply type `quay.io/cumulus/pegasus-terra:<version>` in **CONTAINER IMAGE** field.

Now click `CREATE` button to start the creation. After waiting for 1-2 minutes, your runtime environment will be ready to use, and it's started automatically.

1.3. Start an environment already created

After creation, this runtime environment is associated with your workspace. You can start the same environment anytime in your workspace by clicking the following button within red circle on top-right panel:



2. Create Your Terra Notebook

In **NOTEBOOKS** tab of your workspace, you can either create a blank notebook, or upload your local notebook. After creation, click **EDIT** button to enter the edit mode, and Terra will automatically start your notebook runtime.

When the start-up is done, you can type the following code in notebook to check if *Pegasus* can be loaded and if it's the correct version you want to use:

```
import pegasus as pg
pg.__version__
```

3. Load Data into Runtime

To use your data on cloud (i.e. in the Google Bucket of your workspace), you should copy it into your notebook runtime by Google Cloud SDK:

```
!gsutil -m cp gs://link-to-count-matrix .
```

where `gs://link-to-count-matrix` is the Google Bucket URL to your count matrix data file.

After that, you can use Pegasus function to load it into memory.

Please refer to tutorials for how to use Pegasus on Terra notebook.

4. Stop Notebook Runtime

When you are done with interactive analysis, don't forget to stop your notebook runtime by clicking the following button of the top-right panel of your workspace within red circle:

Otherwise, Google Cloud will keep on charging you.

1.1.4 Tutorials

- Analysis Tutorial: A case study on single-cell RNA sequencing data analysis using Pegasus.
- Plotting Tutotrial: Show how to use Pegasus to generate different kinds of plots for analysis.
- Batch Correction Tutorial: Introduction on batch correction / data integration methods available in Pegasus.
- Doublet Detection Tutorial: Introduction on doublet detection method in Pegasus.
- Regress Out Tutorial: Introduction on regressing out via a case study on cell-cycle gene effects.

1.1.5 API

Pegasus can also be used as a python package. Import pegasus by:

```
import pegasus as pg
```

Read and Write

read_input(input_file[, file_type, mode, ...])	Load data into memory.
write_output(data, output_file[, file_type, ...])	Write data back to disk.
aggregate_matrices(csv_file[, restrictions, ...])	Aggregate channel-specific count matrices into one big count matrix.

pegasus.read_input

pegasus.**read_input**(*input_file*, *file_type=None*, *mode='r'*, *genome=None*, *modality=None*, *black_list=None*, *select_data=None*, *select_genome=None*, *select_modality=None*)

Load data into memory.

This function is used to load input data into memory. Inputs can be in 'zarr', 'h5ad', 'loom', '10x', 'mtx', 'csv', 'tsv', 'fcs' (for flow/mass cytometry data) or 'nanostring' (Nanostring GeoMx spatial data) formats.

Parameters

- **input_file** (*str*) – Input file name.
- **file_type** (*str*, optional (default: None)) – File type, choosing from 'zarr', 'h5ad', 'loom', '10x', 'mtx', 'csv', 'tsv', 'fcs' (for flow/mass cytometry data) or 'nanostring'. If None, inferred from input_file.

- **mode** (*str*, optional (default: 'r')) – File open mode, options are 'r' or 'a'. If mode == 'a', file_type must be zarr and ngene/select_singlets cannot be set.
- **genome** (*str*, optional (default: None)) – For formats like loom, mtx, dge, csv and tsv, genome is used to provide genome name. In this case if genome is None, except mtx format, "unknown" is used as the genome name instead.
- **modality** (*str*, optional (default: None)) – Default modality, choosing from 'rna', 'atac', 'tcr', 'bcr', 'crispr', 'hashing', 'citeseq', 'cyto' (flow cytometry / mass cytometry) or 'nanostring'. If None, use 'rna' as default.
- **black_list** (*Set[str]*, optional (default: None)) – Attributes in black list will be poped out.
- **select_data** (*Set[str]*, optional (default: None)) – Only select data with keys in select_data. Select_data, select_genome and select_modality are mutually exclusive.
- **select_genome** (*Set[str]*, optional (default: None)) – Only select data with genomes in select_genome. Select_data, select_genome and select_modality are mutually exclusive.
- **select_modality** (*Set[str]*, optional (default: None)) – Only select data with modalities in select_modality. Select_data, select_genome and select_modality are mutually exclusive.

Returns

Return type A MultimodalData object.

Examples

```
>>> data = io.read_input('example_10x.h5')
>>> data = io.read_input('example.h5ad')
>>> data = io.read_input('example_ADT.csv', genome = 'hashing_HTO', modality =
↪'hashing')
```

pegasus.write_output

pegasus.**write_output**(*data*, *output_file*, *file_type=None*, *is_sparse=True*, *precision=2*)

Write data back to disk.

This function is used to write data back to disk.

Parameters

- **data** (*MutimodalData*) – data to write back.
- **output_file** (*str*) – output file name. Note that for mtx files, output_file specifies a directory. For scp format, file_type must be specified.
- **file_type** (*str*, optional (default: None)) – File type can be 'zarr' (as folder), 'zarr.zip' (as a ZIP file), 'h5ad', 'loom', 'mtx' or 'scp'. If file_type is None, it will be inferred based on output_file.
- **is_sparse** (*bool*, optional (default: True)) – Only used for writing out SCP-compatible files, if write expression as a sparse matrix.
- **precision** (*int*, optional (default: 2)) – Precision after decimal point for values in mtx and scp expression matrix.

Returns

Return type *None*

Examples

```
>>> io.write_output(data, 'test.zarr')
```

pegasus.aggregate_matrices

pegasus.**aggregate_matrices**(*csv_file*, *restrictions=[]*, *attributes=[]*, *default_ref=None*, *append_sample_name=True*, *select_singlets=False*, *remap_string=None*, *subset_string=None*, *min_genes=None*, *max_genes=None*, *min_umis=None*, *max_umis=None*, *mito_prefix=None*, *percent_mito=None*)

Aggregate channel-specific count matrices into one big count matrix.

This function takes as input a csv_file, which contains at least 2 columns — Sample, sample name; Location, file that contains the count matrices (e.g. filtered_gene_bc_matrices_h5.h5), and merges matrices from the same genome together. If multi-modality exists, a third Modality column might be required. An aggregated Multimodal Data will be returned.

Parameters

- **csv_file** (*str*) – The CSV file containing information about each channel. Alternatively, a dictionary or pd.Dataframe can be passed.
- **restrictions** (*list[str]* or *str*, optional (default: [])) – A list of restrictions used to select channels, each restriction takes the format of name:value,...,value or name:~value,..,value, where ~ refers to not. If only one restriction is provided, it can be provided as a string instead of a list.
- **attributes** (*list[str]* or *str*, optional (default: [])) – A list of attributes need to be incorporated into the output count matrix. If only one attribute is provided, this attribute can be provided as a string instead of a list.
- **default_ref** (*str*, optional (default: None)) – Default reference name to use. If there is no Reference column in the csv_file, a Reference column will be added with default_ref as its value. This argument can also be used for replacing genome names. For example, if default_ref is 'hg19:GRCh38,GRCh38', we will change any genome with name 'hg19' to 'GRCh38' and if no genome is provided, 'GRCh38' is the default.
- **append_sample_name** (*bool*, optional (default: True)) – By default, append sample_name to each channel. Turn this option off if each channel has distinct barcodes.
- **select_singlets** (*bool*, optional (default: False)) – If we have demultiplexed data, turning on this option will make pegasus only include barcodes that are predicted as singlets.
- **remap_string** (`str`, optional, default `None`) – Remap singlet names using <remap_string>, where <remap_string> takes the format "new_name_i:old_name_1,old_name_2;new_name_ii:old_name_3;...". For example, if we hashed 5 libraries from 3 samples sample1_lib1, sample1_lib2, sample2_lib1, sample2_lib2 and sample3, we can remap them to 3 samples using this string: "sample1:sample1_lib1,sample1_lib2;sample2:sample2_lib1,sample2_lib2". In this way, the new singlet names will be in metadata field with key 'assignment', while the old names will be kept in metadata field with key 'assignment.orig'.

- **subset_string** (str, optional, default None) – If select singlets, only select singlets in the <subset_string>, which takes the format "name1,name2,...". Note that if –remap-singlets is specified, subsetting happens after remapping. For example, we can only select singlets from sampe 1 and 3 using "sample1,sample3".
- **min_genes** (int, optional, default: None) – Only keep cells with at least min_genes genes.
- **max_genes** (int, optional, default: None) – Only keep cells with less than max_genes genes.
- **min_umis** (int, optional, default: None) – Only keep cells with at least min_umis UMIs.
- **max_umis** (int, optional, default: None) – Only keep cells with less than max_umis UMIs.
- **mito_prefix** (str, optional, default: None) – Prefix for mitochondrial genes.
- **percent_mito** (float, optional, default: None) – Only keep cells with percent mitochondrial genes less than percent_mito % of total counts. Only when both mito_prefix and percent_mito set, the mitochondrial filter will be triggered.

Returns The aggregated count matrix as an MultimodalData object.

Return type *MultimodalData* object.

Examples

```
>>> data = aggregate_matrix('example.csv', restrictions=['Source:pbmc', 'Donor:1
↪'], attributes=['Source', 'Platform', 'Donor'])
```

Analysis Tools

Preprocess

qc_metrics(data[, select_singlets, ...])	Generate Quality Control (QC) metrics regarding cell barcodes on the dataset.
get_filter_stats(data[, min_genes_before_filt])	Calculate filtration stats on cell barcodes.
filter_data(data[, focus_list])	Filter data based on qc_metrics calculated in pg.qc_metrics.
identify_robust_genes(data[, percent_cells])	Identify robust genes as candidates for HVG selection and remove genes that are not expressed in any cells.
log_norm(data[, norm_count, backup_matrix])	Normalization, and then apply natural logarithm to the data.
highly_variable_features(data[, ...])	Highly variable features (HVF) selection.
select_features(data[, features, ...])	Subset the features and store the resulting matrix in dense format in data.uns with *'_tmp_fmat_'* prefix, with the option of standardization and truncating based on max_value.
pca(data[, n_components, features, ...])	Perform Principle Component Analysis (PCA) to the data.
regress_out(data, attrs[, rep])	Regress out effects due to specific observational attributes.

Batch Correction

set_group_attribute(data, attribute_string)	Set group attributes used in batch correction.
correct_batch(data[, features])	Batch correction on data using Location-Scale (L/S) Adjustment method.
run_harmony(data[, rep, n_jobs, n_clusters, ...])	Batch correction on PCs using Harmony.
run_scanorama(data[, n_components, ...])	Batch correction using Scanorama.

Nearest Neighbors

neighbors(data[, K, rep, n_jobs, ...])	Compute k nearest neighbors and affinity matrix, which will be used for diffmap and graph-based community detection algorithms.
get_neighbors(data[, K, rep, n_jobs, ...])	Find K nearest neighbors for each data point and return the indices and distances arrays.
calc_kBET(data, attr[, rep, K, alpha, ...])	Calculate the kBET metric of the data regarding a specific sample attribute and embedding.
calc_kSIM(data, attr[, rep, K, min_rate, ...])	Calculate the kSIM metric of the data regarding a specific sample attribute and embedding.

pegasus.neighbors

pegasus.**neighbors**(*data*, *K=100*, *rep='pca'*, *n_jobs=- 1*, *random_state=0*, *full_speed=False*)

Compute k nearest neighbors and affinity matrix, which will be used for diffmap and graph-based community detection algorithms.

The kNN calculation uses hnswlib introduced by [Malkov16].

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **K** (int, optional, default: 100) – Number of neighbors, including the data point itself.
- **rep** (str, optional, default: "pca") – Embedding representation used to calculate kNN. If None, use data.X; otherwise, keyword 'X_' + rep must exist in data.obsm.
- **n_jobs** (int, optional, default: -1) – Number of threads to use. If -1, use all available threads.
- **random_state** (int, optional, default: 0) – Random seed set for reproducing results.
- **full_speed** (bool, optional, default: False) –
  - If True, use multiple threads in constructing hnsw index. However, the kNN results are not reproducible.
  - Otherwise, use only one thread to make sure results are reproducible.

Return type None

Returns

- None

- Update `data.uns` –
  - `data.uns[rep + "_knn_indices"]`: kNN index matrix. Row i is the index list of kNN of cell i (excluding itself), sorted from nearest to farthest.
  - `data.uns[rep + "_knn_distances"]`: kNN distance matrix. Row i is the distance list of kNN of cell i (excluding itselt), sorted from smallest to largest.
  - `data.uns["W_" + rep]`: kNN graph of the data in terms of affinity matrix.

Examples

```
>>> pg.neighbors(data)
```

pegasus.get_neighbors

pegasus.**get_neighbors**(*data*, *K=100*, *rep='pca'*, *n_jobs=- 1*, *random_state=0*, *full_speed=False*)

Find K nearest neighbors for each data point and return the indices and distances arrays.

Parameters

- **data** (*pegasusio.MultimodalData*) – An AnnData object.
- **K** (*int*, optional (default: 100)) – Number of neighbors, including the data point itself.
- **rep** (*str*, optional (default: 'pca')) – Representation used to calculate kNN. If *None* use data.X
- **n_jobs** (*int*, optional (default: -1)) – Number of threads to use. -1 refers to all available threads
- **random_state** (*int*, optional (default: 0)) – Random seed for random number generator.
- **full_speed** (*bool*, optional (default: False)) – If full_speed, use multiple threads in constructing hnsw index. However, the kNN results are not reproducible. If not full_speed, use only one thread to make sure results are reproducible.

Returns

Return type kNN indices and distances arrays.

Examples

```
>>> indices, distances = tools.get_neighbors(data)
```

pegasus.calc_kBET

pegasus.**calc_kBET**(*data*, *attr*, *rep='pca'*, *K=25*, *alpha=0.05*, *n_jobs=- 1*, *random_state=0*, *temp_folder=None*)

Calculate the kBET metric of the data regarding a specific sample attribute and embedding.

The kBET metric is defined in [Büttner18], which measures if cells from different samples mix well in their local neighborhood.

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **attr** (str) – The sample attribute to consider. Must exist in data.obs.
- **rep** (str, optional, default: "pca") – The embedding representation to be used. The key 'X_' + rep must exist in data.obsm. By default, use PCA coordinates.
- **K** (int, optional, default: 25) – Number of nearest neighbors, using L2 metric.
- **alpha** (float, optional, default: 0.05) – Acceptance rate threshold. A cell is accepted if its kBET p-value is greater than or equal to alpha.
- **n_jobs** (int, optional, default: -1) – Number of threads used. If -1, use all available threads.
- **random_state** (int, optional, default: 0) – Random seed set for reproducing results.
- **temp_folder** (str, optional, default: None) – Temporary folder for joblib execution.

Return type Tuple[float, float, float]

Returns

- **stat_mean** (float) – Mean kBET chi-square statistic over all cells.
- **pvalue_mean** (float) – Mean kBET p-value over all cells.
- **accept_rate** (float) – kBET Acceptance rate of the sample.

Examples

```
>>> pg.calc_kBET(data, attr = 'Channel')
```

```
>>> pg.calc_kBET(data, attr = 'Channel', rep = 'umap')
```

pegasus.calc_kSIM

pegasus.**calc_kSIM**(*data*, *attr*, *rep='pca'*, *K=25*, *min_rate=0.9*, *n_jobs=- 1*, *random_state=0*)

Calculate the kSIM metric of the data regarding a specific sample attribute and embedding.

The kSIM metric is defined in [Li20], which measures if a sample attribute is not diffused too much in each cell's local neighborhood.

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **attr** (str) – The sample attribute to consider. Must exist in data.obs.
- **rep** (str, optional, default: "pca") – The embedding representation to consider. The key 'X_' + rep must exist in data.obsm.
- **K** (int, optional, default: 25) – The number of nearest neighbors to be considered.
- **min_rate** (float, optional, default: 0.9) – Acceptance rate threshold. A cell is accepted if its kSIM rate is larger than or equal to min_rate.
- **n_jobs** (int, optional, default: -1) – Number of threads used. If -1, use all available threads.

- **random_state** (int, optional, default: 0) – Random seed set for reproducing results.

Return type Tuple[float, float]

Returns

- **kSIM_mean** (float) – Mean kSIM rate over all the cells.
- **kSIM_accept_rate** (float) – kSIM Acceptance rate of the sample.

Examples

```
>>> pg.calc_kSIM(data, attr = 'cell_type')
```

```
>>> pg.calc_kSIM(data, attr = 'cell_type', rep = 'umap')
```

Diffusion Map

diffmap(data[, n_components, rep, solver, ...])	Calculate Diffusion Map.
reduce_diffmap_to_3d(data[, random_state])	Reduce high-dimensional Diffusion Map matrix to 3-dimentional.
calc_pseudotime(data, roots)	Calculate Pseudotime based on Diffusion Map.
infer_path(data, cluster, clust_id, path_name)	Inference on path of a cluster.

pegasus.diffmap

pegasus.**diffmap**(*data*, *n_components=100*, *rep='pca'*, *solver='eigsh'*, *random_state=0*, *max_t=5000*)

Calculate Diffusion Map.

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **n_components** (int, optional, default: 100) – Number of diffusion components to calculate.
- **rep** (str, optional, default: "pca") – Embedding Representation of data used for calculating the Diffusion Map. By default, use PCA coordinates.
- **solver** (str, optional, default: "eigsh") –

  Solver for eigen decomposition:

  - "eigsh": default setting. Use *scipy* eigsh as the solver to find eigenvalus and eigenvectors using the Implicitly Restarted Lanczos Method.
  - "randomized": Use *scikit-learn* randomized_svd as the solver to calculate a truncated randomized SVD.
- **random_state** (int, optional, default: 0) – Random seed set for reproducing results.
- **max_t** (float, optional, default: 5000) – pegasus tries to determine the best t to sum up to between [1, max_t].

Return type None

Returns

- None
- Update data.obsm –
  - data.obsm["X_diffmap"]: Diffusion Map matrix of the data.
- Update data.uns –
  - data.uns["diffmap_evals"]: Eigenvalues corresponding to Diffusion Map matrix.

Examples

```
>>> pg.diffmap(data)
```

pegasus.reduce_diffmap_to_3d

pegasus.**reduce_diffmap_to_3d**(*data*, *random_state=0*)

Reduce high-dimensional Diffusion Map matrix to 3-dimentional.

Parameters **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.

random_state: int, optional, default: 0 Random seed set for reproducing results.

Return type None

Returns

- None
- Update data.obsm –
  - data.obsm["X_diffmap_pca"]: 3D Diffusion Map matrix of data.

Examples

```
>>> pg.reduce_diffmap_to_3d(data)
```

pegasus.calc_pseudotime

pegasus.**calc_pseudotime**(*data*, *roots*)

Calculate Pseudotime based on Diffusion Map.

Parameters

- **data** (anndata.AnnData) – Annotated data matrix with rows for cells and columns for genes.
- **roots** (List[str]) – List of cell barcodes in the data.

Return type None

Returns

- None
- Update data.obs –
  - data.obs["pseudotime"]: Pseudotime result.

Examples

```
>>> pg.calc_pseudotime(adata, roots = list(adata.obs_names[0:100]))
```

pegasus.infer_path

pegasus.**infer_path**(*data*, *cluster*, *clust_id*, *path_name*, *k=10*)

Inference on path of a cluster.

Parameters

- **data** (anndata.AnnData) – Annotated data matrix with rows for cells and columns for genes.
- **cluster** (str) – Cluster name. Must exist in data.obs.
- **clust_id** – Cluster label. Must be a value of data.obs[cluster].
- **path_name** (str) – Key name of the resulting path information.
- **k** (int, optional, default: 10) – Number of nearest neighbors on Diffusion Map coordinates used in path reference.

Returns

- None
- Update data.obs –
  - data.obs[path_name]: The inferred path information on Diffusion Map about a specific cluster.

Examples

```
>>> pg.infer_path(adata, cluster = 'leiden_labels', clust_id = '1', path_name =
↪'leiden_1_path')
```

Cluster Algorithms

cluster(data[, algo, rep, resolution, ...])	Cluster the data using the chosen algorithm.
louvain(data[, rep, resolution, ...])	Cluster the cells using Louvain algorithm.
leiden(data[, rep, resolution, n_iter, ...])	Cluster the data using Leiden algorithm.
spectral_louvain(data[, rep, resolution, ...])	Cluster the data using Spectral Louvain algorithm.
spectral_leiden(data[, rep, resolution, ...])	Cluster the data using Spectral Leiden algorithm.

pegasus.cluster

pegasus.**cluster**(*data*, *algo='louvain'*, *rep='pca'*, *resolution=1.3*, *random_state=0*, *class_label=None*, *n_iter=- 1*, *rep_kmeans='diffmap'*, *n_clusters=30*, *n_clusters2=50*, *n_init=10*)

Cluster the data using the chosen algorithm.

Candidates are *louvain*, *leiden*, *spectral_louvain* and *spectral_leiden*. If data have < 1000 cells and there are clusters with sizes of 1, resolution is automatically reduced until no cluster of size 1 appears.

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **algo** (str, optional, default: "louvain") – Which clustering algorithm to use. Choices are louvain, leiden, spectral_louvain, spectral_leiden
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X_' + rep must exist in data.obsm. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters.
- **random_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class_label** (str, optional, default: None) – Key name for storing cluster labels in data.obs. If None, use 'algo_labels'.
- **n_iter** (int, optional, default: -1) – Number of iterations that Leiden algorithm runs. If -1, run the algorithm until reaching its optimal clustering.
- **rep_kmeans** (str, optional, default: "diffmap") – The embedding representation on which the KMeans runs. Keyword must exist in data.obsm. By default, use Diffusion Map coordinates. If diffmap is not calculated, use PCA coordinates instead.
- **n_clusters** (int, optional, default: 30) – The number of first level clusters.
- **n_clusters2** (int, optional, default: 50) – The number of second level clusters.
- **n_init** (int, optional, default: 10) – Number of kmeans tries for the first level clustering. Default is set to be the same as scikit-learn Kmeans function.

Return type None

Returns

- None
- Update data.obs –
  - data.obs[class_label]: Cluster labels of cells as categorical data.

Examples

```
>>> pg.cluster(data, algo = 'leiden')
```

pegasus.louvain

pegasus.**louvain**(*data*, *rep='pca'*, *resolution=1.3*, *random_state=0*, *class_label='louvain_labels'*)

Cluster the cells using Louvain algorithm. [Blondel08]

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X_' + rep must exist in data.obsm and nearest neighbors must be calculated so that affinity matrix 'W_' + rep exists in data.uns. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters with smaller sizes.
- **random_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class_label** (str, optional, default: "louvain_labels") – Key name for storing cluster labels in data.obs.

Return type None

Returns

- None
- Update data.obs –
  - data.obs[class_label]: Cluster labels of cells as categorical data.

Examples

```
>>> pg.louvain(data)
```

pegasus.leiden

pegasus.**leiden**(*data*, *rep='pca'*, *resolution=1.3*, *n_iter=-1*, *random_state=0*, *class_label='leiden_labels'*)

Cluster the data using Leiden algorithm. [Traag19]

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X_' + rep must exist in data.obsm and nearest neighbors must be calculated so that affinity matrix 'W_' + rep exists in data.uns. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters.
- **n_iter** (int, optional, default: -1) – Number of iterations that Leiden algorithm runs. If -1, run the algorithm until reaching its optimal clustering.
- **random_state** (int, optional, default: 0) – Random seed for reproducing results.

- **class_label** (str, optional, default: "leiden_labels") – Key name for storing cluster labels in data.obs.

Return type None

Returns

- None
- Update data.obs –
  - data.obs[class_label]: Cluster labels of cells as categorical data.

Examples

```
>>> pg.leiden(data)
```

pegasus.spectral_louvain

pegasus.**spectral_louvain**(*data*, *rep='pca'*, *resolution=1.3*, *rep_kmeans='diffmap'*, *n_clusters=30*, *n_clusters2=50*, *n_init=10*, *random_state=0*, *class_label='spectral_louvain_labels'*)

Cluster the data using Spectral Louvain algorithm. [Li20]

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X_' + rep must exist in data.obsm. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters with smaller sizes.
- **rep_kmeans** (str, optional, default: "diffmap") – The embedding representation on which the KMeans runs. Keyword must exist in data.obsm. By default, use Diffusion Map coordinates. If diffmap is not calculated, use PCA coordinates instead.
- **n_clusters** (int, optional, default: 30) – The number of first level clusters.
- **n_clusters2** (int, optional, default: 50) – The number of second level clusters.
- **n_init** (int, optional, default: 10) – Number of kmeans tries for the first level clustering. Default is set to be the same as scikit-learn Kmeans function.
- **random_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class_label** (str, optional, default: "spectral_louvain_labels") – Key name for storing cluster labels in data.obs.

Return type None

Returns

- None
- Update data.obs –
  - data.obs[class_label]: Cluster labels for cells as categorical data.

Examples

```
>>> pg.spectral_louvain(data)
```

pegasus.spectral_leiden

pegasus.**spectral_leiden**(*data*, *rep='pca'*, *resolution=1.3*, *rep_kmeans='diffmap'*, *n_clusters=30*, *n_clusters2=50*, *n_init=10*, *random_state=0*, *class_label='spectral_leiden_labels'*)

Cluster the data using Spectral Leiden algorithm. [Li20]

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X_' + rep must exist in data.obsm. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters.
- **rep_kmeans** (str, optional, default: "diffmap") – The embedding representation on which the KMeans runs. Keyword must exist in data.obsm. By default, use Diffusion Map coordinates. If diffmap is not calculated, use PCA coordinates instead.
- **n_clusters** (int, optional, default: 30) – The number of first level clusters.
- **n_clusters2** (int, optional, default: 50) – The number of second level clusters.
- **n_init** (int, optional, default: 10) – Number of kmeans tries for the first level clustering. Default is set to be the same as scikit-learn Kmeans function.
- **random_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class_label** (str, optional, default: "spectral_leiden_labels") – Key name for storing cluster labels in data.obs.

Return type None

Returns

- None
- Update data.obs –
  - data.obs[class_label]: Cluster labels for cells as categorical data.

Examples

```
>>> pg.spectral_leiden(data)
```

Visualization Algorithms

tsne(data[, rep, n_jobs, n_components, ...])	Calculate t-SNE embedding of cells using the FIt-SNE package.
umap(data[, rep, n_components, n_neighbors, ...])	Calculate UMAP embedding of cells.
fle(data[, file_name, n_jobs, rep, K, ...])	Construct the Force-directed (FLE) graph.
net_umap(data[, rep, n_jobs, n_components, ...])	Calculate Net-UMAP embedding of cells.
net_fle(data[, file_name, n_jobs, rep, K, ...])	Construct Net-Force-directed (FLE) graph.

pegasus.tsne

pegasus.**tsne**(*data*, *rep='pca'*, *n_jobs=- 1*, *n_components=2*, *perplexity=30*, *early_exaggeration=12*, *learning_rate='auto'*, *initialization='pca'*, *random_state=0*, *out_basis='tsne'*)

Calculate t-SNE embedding of cells using the FIt-SNE package.

This function uses fitsne package. See [Linderman19] for details on FIt-SNE algorithm.

Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (`str`, optional, default: `"pca"`) – Representation of data used for the calculation. By default, use PCA coordinates. If `None`, use the count matrix `data.X`.
- **n_jobs** (`int`, optional, default: `-1`) – Number of threads to use. If `-1`, use all available threads.
- **n_components** (`int`, optional, default: `2`) – Dimension of calculated FI-tSNE coordinates. By default, generate 2-dimensional data for 2D visualization.
- **perplexity** (`float`, optional, default: `30`) – The perplexity is related to the number of nearest neighbors used in other manifold learning algorithms. Larger datasets usually require a larger perplexity.
- **early_exaggeration** (`int`, optional, default: `12`) – Controls how tight natural clusters in the original space are in the embedded space, and how much space will be between them.
- **learning_rate** (`float`, optional, default: `auto`) – By default, the learning rate is determined automatically as max(data.shape[0] / early_exaggeration, 200). See [Belkina19] and [Kobak19] for details.
- **initialization** (`str`, optional, default: `pca`) – Initialization can be either `pca` or `random` or np.ndarray. By default, we use `pca` initialization according to [Kobak19].
- **random_state** (`int`, optional, default: `0`) – Random seed set for reproducing results.
- **out_basis** (`str`, optional, default: `"fitsne"`) – Key name for calculated FI-tSNE coordinates to store.

Return type `None`

Returns

- `None`
- Update `data.obsm` –
  - `data.obsm['X_' + out_basis]`: FI-tSNE coordinates of the data.

Examples

```
>>> pg.tsne(data)
```

pegasus.umap

pegasus.**umap**(*data*, *rep='pca'*, *n_components=2*, *n_neighbors=15*, *min_dist=0.5*, *spread=1.0*, *n_jobs=- 1*, *full_speed=False*, *random_state=0*, *out_basis='umap'*)

Calculate UMAP embedding of cells.

This function uses umap-learn package. See [McInnes18] for details on UMAP.

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – Representation of data used for the calculation. By default, use PCA coordinates. If None, use the count matrix data.X.
- **n_components** (int, optional, default: 2) – Dimension of calculated UMAP coordinates. By default, generate 2-dimensional data for 2D visualization.
- **n_neighbors** (int, optional, default: 15) – Number of nearest neighbors considered during the computation.
- **min_dist** (float, optional, default: 0.5) – The effective minimum distance between embedded data points.
- **spread** (float, optional, default: 1.0) – The effective scale of embedded data points.
- **n_jobs** (int, optional, default: -1) – Number of threads to use for computing kNN graphs. If -1, use all available threads.
- **full_speed** (bool, optional, default: False) –
  - If True, use multiple threads in constructing hnsw index. However, the kNN results are not reproducible.
  - Otherwise, use only one thread to make sure results are reproducible.
- **random_state** (int, optional, default: 0) – Random seed set for reproducing results.
- **out_basis** (str, optional, default: "umap") – Key name for calculated UMAP coordinates to store.

Return type None

Returns

- None
- Update data.obsm –
  - data.obsm['X_' + out_basis]: UMAP coordinates of the data.

Examples

```
>>> pg.umap(data)
```

pegasus.fle

pegasus.**fle**(*data*, *file_name=None*, *n_jobs=- 1*, *rep='diffmap'*, *K=50*, *full_speed=False*, *target_change_per_node=2.0*, *target_steps=5000*, *is3d=False*, *memory=8*, *random_state=0*, *out_basis='fle'*)

Construct the Force-directed (FLE) graph.

This implementation uses forceatlas2-python package, which is a Python wrapper of ForceAtlas2.

See [Jacomy14] for details on FLE.

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **file_name** (str, optional, default: None) – Temporary file to store the coordinates as the input to forceatlas2. If None, use tempfile.mkstemp to generate file name.
- **n_jobs** (int, optional, default: -1) – Number of threads to use. If -1, use all available threads.
- **rep** (str, optional, default: "diffmap") – Representation of data used for the calculation. By default, use Diffusion Map coordinates. If None, use the count matrix data.X.
- **K** (int, optional, default: 50) – Number of nearest neighbors to be considered during the computation.
- **full_speed** (bool, optional, default: False) –
  - If True, use multiple threads in constructing hnsw index. However, the kNN results are not reproducible.
  - Otherwise, use only one thread to make sure results are reproducible.
- **target_change_per_node** (float, optional, default: 2.0) – Target change per node to stop ForceAtlas2.
- **target_steps** (int, optional, default: 5000) – Maximum number of iterations before stopping the ForceAtlas2 algorithm.
- **is3d** (bool, optional, default: False) – If True, calculate 3D force-directed layout.
- **memory** (int, optional, default: 8) – Memory size in GB for the Java FA2 component. By default, use 8GB memory.
- **random_state** (int, optional, default: 0) – Random seed set for reproducing results.
- **out_basis** (str, optional, default: "fle") – Key name for calculated FLE coordinates to store.

Return type None

Returns

- None
- Update data.obsm –

– `data.obsm['X_' + out_basis]`: FLE coordinates of the data.

Examples

```
>>> pg.fle(data)
```

pegasus.net_umap

`pegasus.net_umap`(*data*, *rep='pca'*, *n_jobs=- 1*, *n_components=2*, *n_neighbors=15*, *min_dist=0.5*, *spread=1.0*, *random_state=0*, *select_frac=0.1*, *select_K=25*, *select_alpha=1.0*, *full_speed=False*, *net_alpha=0.1*, *polish_learning_rate=10.0*, *polish_n_epochs=30*, *out_basis='net_umap'*)

Calculate Net-UMAP embedding of cells.

Net-UMAP is an approximated UMAP embedding using Deep Learning model to improve the speed.

In specific, the deep model used is MLPRegressor, the *scikit-learn* implementation of Multi-layer Perceptron regressor.

See [Li20] for details.

Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (`str`, optional, default: `"pca"`) – Representation of data used for the calculation. By default, use PCA coordinates. If `None`, use the count matrix `data.X`.
- **n_components** (`int`, optional, default: `2`) – Dimension of calculated UMAP coordinates. By default, generate 2-dimensional data for 2D visualization.
- **n_neighbors** (`int`, optional, default: `15`) – Number of nearest neighbors considered during the computation.
- **min_dist** (`float`, optional, default: `0.5`) – The effective minimum distance between embedded data points.
- **spread** (`float`, optional, default: `1.0`) – The effective scale of embedded data points.
- **random_state** (`int`, optional, default: `0`) – Random seed set for reproducing results.
- **select_frac** (`float`, optional, default: `0.1`) – Down sampling fraction on the cells.
- **select_K** (`int`, optional, default: `25`) – Number of neighbors to be used to estimate local density for each data point for down sampling.
- **select_alpha** (`float`, optional, default: `1.0`) – Weight the down sample to be proportional to `radius ** select_alpha`.
- **full_speed** (`bool`, optional, default: `False`) –
  - If `True`, use multiple threads in constructing `hnsw` index. However, the kNN results are not reproducible.
  - Otherwise, use only one thread to make sure results are reproducible.
- **net_alpha** (`float`, optional, default: `0.1`) – L2 penalty (regularization term) parameter of the deep regressor.

- **polish_learning_frac** (float, optional, default: 10.0) – After running the deep regressor to predict new coordinates, use polish_learning_frac * n_obs as the learning rate to polish the coordinates.
- **polish_n_iter** (int, optional, default: 30) – Number of iterations for polishing UMAP run.
- **out_basis** (str, optional, default: "net_umap") – Key name for calculated UMAP coordinates to store.

Return type None

Returns

- None
- Update data.obsm –
  - data.obsm['X_' + out_basis]: Net UMAP coordinates of the data.
- Update data.obs –
  - data.obs['ds_selected']: Boolean array to indicate which cells are selected during the down sampling phase.

Examples

```
>>> pg.net_umap(data)
```

pegasus.net_fle

pegasus.**net_fle**(*data*, *file_name=None*, *n_jobs=- 1*, *rep='diffmap'*, *K=50*, *full_speed=False*, *target_change_per_node=2.0*, *target_steps=5000*, *is3d=False*, *memory=8*, *random_state=0*, *select_frac=0.1*, *select_K=25*, *select_alpha=1.0*, *net_alpha=0.1*, *polish_target_steps=1500*, *out_basis='net_fle'*)

Construct Net-Force-directed (FLE) graph.

Net-FLE is an approximated FLE graph using Deep Learning model to improve the speed.

In specific, the deep model used is MLPRegressor, the *scikit-learn* implementation of Multi-layer Perceptron regressor.

See [Li20] for details.

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **file_name** (str, optional, default: None) – Temporary file to store the coordinates as the input to forceatlas2. If None, use tempfile.mkstemp to generate file name.
- **n_jobs** (int, optional, default: -1) – Number of threads to use. If -1, use all available threads.
- **rep** (str, optional, default: "diffmap") – Representation of data used for the calculation. By default, use Diffusion Map coordinates. If None, use the count matrix data.X.
- **K** (int, optional, default: 50) – Number of nearest neighbors to be considered during the computation.

- **full_speed** (bool, optional, default: False) –
  - If True, use multiple threads in constructing hnsw index. However, the kNN results are not reproducible.
  - Otherwise, use only one thread to make sure results are reproducible.
- **target_change_per_node** (float, optional, default: 2.0) – Target change per node to stop ForceAtlas2.
- **target_steps** (int, optional, default: 5000) – Maximum number of iterations before stopping the ForceAtlas2 algorithm.
- **is3d** (bool, optional, default: False) – If True, calculate 3D force-directed layout.
- **memory** (int, optional, default: 8) – Memory size in GB for the Java FA2 component. By default, use 8GB memory.
- **random_state** (int, optional, default: 0) – Random seed set for reproducing results.
- **select_frac** (float, optional, default: 0.1) – Down sampling fraction on the cells.
- **select_K** (int, optional, default: 25) – Number of neighbors to be used to estimate local density for each data point for down sampling.
- **select_alpha** (float, optional, default: 1.0) – Weight the down sample to be proportional to radius ** select_alpha.
- **net_alpha** (float, optional, default: 0.1) – L2 penalty (regularization term) parameter of the deep regressor.
- **polish_target_steps** (int, optional, default: 1500) – After running the deep regressor to predict new coordinate, Number of ForceAtlas2 iterations.
- **out_basis** (str, optional, default: "net_fle") – Key name for calculated FLE coordinates to store.

Return type None

Returns

- None
- Update data.obsm –
  - data.obsm['X_' + out_basis]: Net FLE coordinates of the data.
- Update data.obs –
  - data.obs['ds_selected']: Boolean array to indicate which cells are selected during the down sampling phase.

Examples

```
>>> pg.net_fle(data)
```

Doublet Detection

infer_doublets(data[, channel_attr, ...])	Infer doublets using a Scrublet-like strategy.
mark_doublets(data[, demux_attr, dbl_clusts])	Convert doublet prediction into doublet annotations that Pegasus can recognize.

pegasus.infer_doublets

pegasus.**infer_doublets**(*data*, *channel_attr=None*, *clust_attr=None*, *min_cell=100*, *expected_doublet_rate=None*, *sim_doublet_ratio=2.0*, *n_prin_comps=30*, *robust=False*, *k=None*, *n_jobs=- 1*, *alpha=0.05*, *random_state=0*, *plot_hist='sample'*)

Infer doublets using a Scrublet-like strategy. [Li20-2]

This function must be called after clustering.

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **channel_attr** (str, optional, default: None) – Attribute indicating sample channels. If set, calculate scrublet-like doublet scores per channel.
- **clust_attr** (str, optional, default: None) – Attribute indicating cluster labels. If set, estimate proportion of doublets in each cluster and statistical significance.
- **min_cell** (int, optional, default: 100) – Minimum number of cells per sample to calculate doublet scores. For samples having less than 'min_cell' cells, doublet score calculation will be skipped.
- **expected_doublet_rate** (float, optional, default: None) – The expected doublet rate for the experiment. By default, calculate the expected rate based on number of cells from the 10x multiplet rate table
- **sim_doublet_ratio** (float, optional, default: 2.0) – The ratio between synthetic doublets and observed cells.
- **n_prin_comps** (int, optional, default: 30) – Number of principal components.
- **robust** (bool, optional, default: False.) – If true, use 'arpack' instead of 'randomized' for large matrices (i.e. max(X.shape) > 500 and n_components < 0.8 * min(X.shape))
- **k** (int, optional, default: None) – Number of observed cell neighbors. If None, k = round(0.5 * sqrt(number of observed cells)). Total neighbors k_adj = round(k * (1.0 + sim_doublet_ratio)).
- **n_jobs** (int, optional, default: -1) – Number of threads to use. If -1, use all available threads.
- **alpha** (float, optional, default: 0.05) – FDR significant level for cluster-level fisher exact test.
- **random_state** (int, optional, default: 0) – Random seed for reproducing results.
- **plot_hist** (str, optional, default: sample) – If not None, plot diagnostic histograms using plot_hist as the prefix. If *channel_attr* is None, plot_hist.dbl.png is generated; Otherwise, plot_hist.channel_name.dbl.png files are generated. Each figure consists of 4 panels showing histograms of doublet scores for observed cells (panel

1, density in log scale), simulated doublets (panel 2, density in log scale), KDE plot (panel 3) and signed curvature plot (panel 4) of log doublet scores for simulated doublets.

Return type None

Returns

- None
- Update data.obs –
  - data.obs['pred_dbl_type']: Predicted singlet/doublet types.
  - data.uns['pred_dbl_cluster']: Only generated if 'clust_attr' is not None. This is a dataframe with two columns, 'Cluster' and 'Qval'. Only clusters with significantly more doublets than expected will be recorded here.

Examples

```
>>> pg.infer_doublets(data, channel_attr = 'Channel', clust_attr = 'Annotation')
```

pegasus.mark_doublets

pegasus.**mark_doublets**(*data*, *demux_attr='demux_type'*, *dbl_clusts=None*)

Convert doublet prediction into doublet annotations that Pegasus can recognize. In addition, clusters in dbl_clusts will be marked as doublets.

Must run infer_doublets first.

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **demux_attr** (str, optional, default: demux_type) – Attribute indicating singlets/doublets that Pegasus can recognize. Currently this is 'demux_type', which is also used for hashing.
- **dbl_clusts** (str, optional, default: None) – Indicate which clusters should be marked as all doublets. It takes the format of 'clust:value1,value2,...', where 'clust' refers to the cluster attribute.

Return type None

Returns

- None
- Update data.obs –
  - data.obs[demux_attr]: Singlet/doublet annotation.

Examples

```
>>> pg.mark_doublets(data, dbl_clusts='Annotation:B/T doublets')
```

Gene Module Score

calc_signature_score(data, signatures[, ...])	Calculate signature / gene module score.

pegasus.calc_signature_score

pegasus.**calc_signature_score**(*data*, *signatures*, *n_bins=50*, *show_omitted_genes=False*, *random_state=0*)

Calculate signature / gene module score. [Li20-1]

This is an improved version of implementation in [Jerby-Arnon18].

Parameters

- **data** (`MultimodalData`, `UnimodalData`, or `anndata.AnnData` object.) – Single cell expression data.
- **signatures** (`Dict[str, List[str]]` or `str`) – This argument accepts either a dictionary or a string. If `signatures` is a dictionary, it can contain multiple signature score calculation requests. Each key in the dictionary represents a separate signature score calculation and its corresponding value contains a list of gene symbols. Each score will be stored in data.obs field with key as the keyword. If `signatures` is a string, it should refer to a Gene Matrix Transposed (GMT)-formatted file. Pegasus will load signatures from the GMT file. Pegasus also provide 5 default signature panels for each of human and mouse. They are `cell_cycle_human`, `gender_human`, `mitochondrial_genes_human`, `ribosomal_genes_human` and `apoptosis_human` for human; `cell_cycle_mouse`, `gender_mouse`, `mitochondrial_genes_mouse`, `ribosomal_genes_mouse` and `apoptosis_mouse` for mouse.
  - `cell_cycle_human` contains two cell-cycle signatures, `G1/S` and `G2/M`, obtained from Tirosh et al. 2016. We also updated gene symbols according to Seurat's `cc.genes.updated.2019` vector. We additionally calculate signature score `cycle_diff`, which is `G2/M - G1/S`. We provide predicted cell cycle phases in `data.obs['predicted_phase']` in case it is useful. `predicted_phase` is predicted based on `G1/S` and `G2/M` scores. First, we identify `G0` cells. We calculate vector `maxvalues` as the maximum score of `G1/S` and `G2/M` for each cell and then apply KMeans algorithm to obtain 2 clusters based on `maxvalues`. `G0` cells are from the cluster with smallest mean value. For each cell from the other cluster, if `G1/S` > `G2/M`, it is a `G1/S` cell, otherwise it is a `G2/M` cell.
  - `gender_human` contains two gender-specific signatures, `female_score` and `male_score`. Genes were selected based on DE analysis between genders based on 8 channels of bone marrow data from HCA Census of Immune Cells and the brain nuclei data from Gaublomme and Li et al, 2019, Nature Communications. After calculation, three signature scores will be calculated: `female_score`, `male_score` and `gender_score`. `female_score` and `male_score` are calculated based on female and male signatures respectively and a larger score represent a higher likelihood of that gender. `gender_score` is calculated as `male_score - female_score`. A large

positive score likely represents male and a large negative score likely represents female. Pegasus also provides predicted gender for each cell based on `gender_score`, which is stored in `data.obs['predicted_gender']`. To predict genders, we apply the KMeans algorithm to the `gender_score` and ask for 3 clusters. The clusters with a minimum and maximum clauster centers are predicted as `female` and `male` respectively and the cluster in the middle is predicted as `uncertain`. Note that this approach is conservative and it is likely that users can predict genders based on `gender_score` for cells in the `uncertain` cluster with a reasonable accuracy.

  - `mitochondrial_genes_human` contains two signatures, `mito_genes` and `mito_ribo`. `mito_genes` contains 13 mitocondrial genes from chrM and `mito_ribo` contains mitocondrial ribosomal genes that are not from chrM. Note that `mito_genes` correlates well with percent of mitocondrial UMIs and `mito_ribo` does not.
  - `ribosomal_genes_human` contains one signature, `ribo_genes`, which includes ribosomal genes from both large and small units.
  - `apoptosis_human` contains one signature, `apoptosis`, which includes apoptosis-related genes from the KEGG pathway.
  - `cell_cycle_mouse`, `gender_mouse`, `mitochondrial_genes_mouse`, `ribosomal_genes_mouse` and `apoptosis_mouse` are the corresponding signatures for mouse. Gene symbols are directly translated from human genes.

- **`n_bins`** (`int`, optional, default: 50) – Number of bins on expression levels for grouping genes
- **`show_omitted_genes`** (`bool`, optional, default False) – Signature genes that are not expressed in the data will be omitted. By default, pegasus does not report which genes are omitted. If this option is turned on, report omitted genes.
- **`random_state`** (`int`, optional, default: 0) – Random state used by KMeans if signature == `gender_human` or `gender_mouse`.

Return type `None`

Returns

- `None`.
- Update `data.obs` –
  - `data.obs["key"]`: signature / gene module score for signature "key"
- Update `data.var` –
  - `data.var["mean"]`: Mean expression of each gene across all cells. Only updated if "mean" does not exist in data.var.
  - `data.var["bins"]`: Bin category for each gene. Only updated if data.uns["sig_n_bins"] is updated.
- Update `data.obsm` –
  - `data.obsm["sig_background"]`: Expected signature score for each bin category. Only updated if data.uns["sig_n_bins"] is updated.
- Update `data.uns` –
  - `data.uns["sig_n_bins"]`: Number of bins to partition genes into. Only updated if "sig_n_bins" does not exist or the recorded number of bins does not match n_bins.

Examples

```
>>> pg.calc_signature_score(data, {"T_cell_sig": ["CD3D", "CD3E", "CD3G", "TRAC"]}
↪)
>>> pg.calc_signature_score(data, "cell_cycle_human")
```

Differential Expression Analysis

de_analysis(data, cluster[, condition, ...])	Perform Differential Expression (DE) Analysis on data.
markers(data[, head, de_key, alpha])	Extract DE results into a human readable structure.
write_results_to_excel(results, output_file)	Write DE analysis results into Excel workbook.

pegasus.de_analysis

pegasus.**de_analysis**(*data*, *cluster*, *condition=None*, *subset=None*, *de_key='de_res'*, *n_jobs=-1*, *t=False*, *fisher=False*, *temp_folder=None*, *verbose=True*)

Perform Differential Expression (DE) Analysis on data.

The analysis considers one cluster at one time, comparing gene expression levels on cells within the cluster with all the others using a number of statistical tools, and determining up-regulated genes and down-regulated genes of the cluster.

Mann-Whitney U test and AUROC are calculated by default. Welch's T test and Fisher's Exact test are optionally.

The scalability performance on calculating all the test statistics is improved by the inspiration from Presto.

Parameters

- **data** (`MultimodalData`, `UnimodalData`, or `anndata.AnnData`) – Data matrix with rows for cells and columns for genes.
- **cluster** (`str`) – Cluster labels used in DE analysis. Must exist in `data.obs`.
- **condition** (`str`, optional, default: `None`) – Sample attribute used as condition in DE analysis. If `None`, no condition is considered; otherwise, must exist in `data.obs`. If `condition` is used, the DE analysis will be performed on cells of each level of `data.obs[condition]` respectively, and collect the results after finishing.
- **subset** (`List[str]`, optional, default: `None`) – Perform DE analysis on only a subset of cluster IDs. Cluster ID subset is specified as a list of strings, such as `[clust_1, clust_3,clust_5]`, where all IDs must exist in `data.obs[cluster]`.
- **de_key** (`str`, optional, default: `"de_res"`) – Key name of DE analysis results stored.
- **n_jobs** (`int`, optional, default: `-1`) – Number of threads to use. If `-1`, use all available threads.
- **t** (`bool`, optional, default: `True`) – If `True`, calculate Welch's t test.
- **fisher** (`bool`, optional, default: `False`) – If `True`, calculate Fisher's exact test.
- **temp_folder** (`str`, optional, default: `None`) – Joblib temporary folder for memmapping numpy arrays.
- **verbose** (`bool`, optional, default: `True`) – If `True`, show detailed intermediate output.

Return type `None`

Returns

- None
- Update data.varm – data.varm[de_key]: DE analysis result.

Examples

```
>>> pg.de_analysis(data, cluster='spectral_leiden_labels')
>>> pg.de_analysis(data, cluster='louvain_labels', condition='anno')
```

pegasus.markers

pegasus.**markers**(*data*, *head=None*, *de_key='de_res'*, *alpha=0.05*)

Extract DE results into a human readable structure.

This function extracts information from data.varm[de_key], and return as a human readible dictionary of pandas DataFrame objects.

Parameters

- **data** (MultimodalData, UnimodalData, or anndata.AnnData) – Data matrix with rows for cells and columns for genes.
- **head** (int, optional, default: None) – List only top head genes for each cluster. If None, show any DE genes.
- **de_key** (str, optional, default, de_res) – Keyword of DE result stored in data.varm.
- **alpha** (float, optional, default: 0.05) – q-value threshold for getting significant DE genes. Only those with q-value of any test no less than alpha are significant, and thus considered as DE genes.

Returns results – A Python dictionary containing markers. If DE is performed between clusters, the structure is dict[cluster_id]['up' or 'down'][dataframe]. If DE is performed between conditions within each cluster, the structure is dict[cluster_id][condition_id]['up' or 'down'][dataframe].

Return type Dict[str, Dict[str, pd.DataFrame]]

Examples

```
>>> marker_dict = pg.markers(data)
```

pegasus.write_results_to_excel

pegasus.**write_results_to_excel**(*results*, *output_file*, *ndigits=3*)

Write DE analysis results into Excel workbook.

Parameters

- **results** (Dict[str, Dict[str, pd.DataFrame]], or Dict[str, Dict[str, Dict[str, pd.DataFrame]]]) – DE marker dictionary generated by pg.markers.
- **output_file** (str) – File name to which the marker dictionary is written.

- **ndigits** (int, optional, default: 3) – Round non p-values and q-values to ndigits after decimal point in the excel.

Return type None

Returns

- None
- Marker information is written to file with name output_file.
- *In the generated Excel workbook, –*
  - If condition is None in pg.de_analysis: Each tab stores DE result of up/down-regulated genes of cells within one cluster, and its name follows the pattern: **"cluster_id|up"** or **"cluster_id|dn"**.
  - If condition is not None in pg.de_analysis: Each tab stores DE result of up/down-regulated genes of cells within one cluster under one condition level. The tab's name follows the pattern: **"cluster_id|cond_level|up"** or **"cluster_id|cond_level|dn"**.
  - Notice that the tab name in Excel only allows at most 31 characters. Therefore, some of the resulting tab names may be truncated if their names are longer than this threshold.

Examples

```
>>> pg.write_results_to_excel(marker_dict, "result.de.xlsx")
```

Annotate clusters

infer_cell_types(data, markers[, de_test, ...])	Infer putative cell types for each cluster using legacy markers.
annotate(data, name, based_on, anno_dict)	Add annotation to AnnData obj.

pegasus.infer_cell_types

pegasus.**infer_cell_types**(*data*, *markers*, *de_test='mwu'*, *de_alpha=0.05*, *de_key='de_res'*, *threshold=0.5*, *ignore_nonde=False*, *output_file=None*)

Infer putative cell types for each cluster using legacy markers.

Parameters

- **data** (MultimodalData, UnimodalData, or anndata.AnnData.) – Data structure of count matrix and DE analysis results.
- **markers** (str or Dict) –
  - **If str, it is a string representing a comma-separated list; each element in the list**
    - either refers to a JSON file containing legacy markers, or predefined markers
    - 'human_immune' for human immune cells;
    - 'mouse_immune' for mouse immune cells;
    - 'human_brain' for human brain cells;
    - 'mouse_brain' for mouse brain cells;

* 'human_lung' for human lung cells.

– If Dict, it refers to a Python dictionary describing the markers.

- **de_test** (str, optional, default: "mwu") – pegasus determines cell types using DE test results. This argument indicates which DE test result to use, can be either 't', 'fisher' or 'mwu'. By default, it uses 'mwu'.
- **de_alpha** (float, optional, default: 0.05) – False discovery rate for controling family-wide error.
- **de_key** (str, optional, default: "de_res") – The keyword in data.varm that stores DE analysis results.
- **threshold** (float, optional, defaut: 0.5) – Only report putative cell types with a score larger than or equal to threshold.
- **ignore_nonde** (bool, optional, default: False) – Do not consider non DE genes as weak negative markers.
- **output_file** (str, optional, default: None) – File name of output cluster annotation. If None, do not write to any file.

Returns Python dictionary with cluster ID's being keys, and their corresponding cell type lists sortec by scores being values.

Return type Dict[str, List["CellType"]]

Examples

```
>>> cell_type_dict = pg.infer_cell_types(adata, markers = 'human_immune,human_
↪brain')
```

pegasus.annotate

pegasus.**annotate**(*data*, *name*, *based_on*, *anno_dict*)

Add annotation to AnnData obj.

Parameters

- **data** (MultimodalData, UnimodalData, or anndata.AnnData) – Gene-count matrix with DE analysis information.
- **name** (*str*) – Name of the new annotation in data.obs.
- **based_on** (*str*) – Name of the attribute the cluster ids coming from.
- **anno_dict** (*Dict[str, str]* or *List[str]*) – Dictionary mapping from cluster id to cell type. If it is a List, map cell types to cluster ids one to one in correspondence.

Returns

Return type None

Examples

```
>>> pg.annotate(data, 'anno', 'spectral_louvain_labels', {'1': 'T cell', '2': 'B␣
↪cell'})
>>> pg.annotate(data, 'anno', 'louvain_labels', ['T cell', 'B cell'])
```

Plotting

scatter(data, attrs[, basis, matkey, ...])	Generate scatter plots for different attributes
scatter_groups(data, attr, groupby[, basis, ...])	Generate scatter plots of attribute 'attr' for each category in attribute 'group'.
compo_plot(data, groupby, condition[, ...])	Generate a composition plot, which shows the percentage of cells from each condition for every cluster.
violin(data, attrs, groupby[, hue, matkey, ...])	Generate a stacked violin plot.
heatmap(data, attrs, groupby[, matkey, ...])	Generate a heatmap.
dotplot(data, genes, groupby[, ...])	Generate a dot plot.
dendrogram(data, groupby[, rep, genes, ...])	Generate a dendrogram on hierarchical clustering result.
hvfplot(data[, top_n, panel_size, ...])	Generate highly variable feature plot.
qcviolin(data, plot_type[, ...])	Plot quality control statistics (before filtration vs.
ridgeplot(data, features[, donor_attr, ...])	Generate ridge plots, up to 8 features can be shown in one figure.
volcano(data, cluster_id[, de_key, de_test, ...])	Generate Volcano plots (-log10 p value vs.

Demultiplexing

estimate_background_probs(hashing_data[, ...])	For cell-hashing data, estimate antibody background probability using KMeans algorithm.
demultiplex(rna_data, hashing_data[, ...])	Demultiplexing cell/nucleus-hashing data, using the estimated antibody background probability calculated in demuxEM.estimate_background_probs.
attach_demux_results(input_rna_file, rna_data)	Write demultiplexing results into raw gene expression matrix.

pegasus.estimate_background_probs

pegasus.**estimate_background_probs** (*hashing_data*, *random_state=0*)

For cell-hashing data, estimate antibody background probability using KMeans algorithm.

Parameters

- **hashing_data** (UnimodalData) – Annotated data matrix for antibody.
- **random_state** (int, optional, default: 0) – Random seed set for reproducing results.

Return type None

Returns

- None
- Update hashing_data.uns –

- `hashing_data.uns["background_probs"]`: estimated antibody background probability.

Example

```
>>> estimate_background_probs(hashing_data)
```

pegasus.demultiplex

pegasus.**demultiplex**(*rna_data*, *hashing_data*, *min_signal=10.0*, *alpha=0.0*, *alpha_noise=1.0*, *tol=1e-06*, *n_threads=1*)

Demultiplexing cell/nucleus-hashing data, using the estimated antibody background probability calculated in `demuxEM.estimate_background_probs`.

Parameters

- **rna_data** (`UnimodalData`) – Data matrix for gene expression matrix.
- **hashing_data** (`UnimodalData`) – Data matrix for HTO count matrix.
- **min_signal** (`float`, optional, default: `10.0`) – Any cell/nucleus with less than `min_signal` hashtags from the signal will be marked as `unknown`.
- **alpha** (`float`, optional, default: `0.0`) – The Dirichlet prior concentration parameter (alpha) on samples. An alpha value < 1.0 will make the prior sparse.
- **alpha_noise** (`float`, optional, default: `1.0`) – The Dirichlet prior concenration parameter on the background noise.
- **tol** (`float`, optional, default: `1e-6`) – Threshold used for the EM convergence.
- **n_threads** (`int`, optional, default: `1`) – Number of threads to use. Must be a positive integer.

Returns

- `None`
- Update `data.obs` –
  - `data.obs["demux_type"]`: Demultiplexed types of the cells. Either `singlet`, `doublet`, or `unknown`.
  - `data.obs["assignment"]`: Assigned samples of origin for each cell barcode.
  - `data.obs["assignment.dedup"]`: Only exist if one sample name can correspond to multiple feature barcodes. In this case, each feature barcode is assigned a unique sample name.

Examples

```
>>> demultiplex(rna_data, hashing_data)
```

pegasus.attach_demux_results

pegasus.**attach_demux_results**(*input_rna_file*, *rna_data*)

Write demultiplexing results into raw gene expression matrix.

Parameters

- **input_rna_file** (str) – Input file for the raw gene count matrix.
- **rna_data** (UnimodalData) – Processed gene count matrix containing demultiplexing results

Return type MultimodalData

Returns

- MultimodalData
- *A multimodal data object.*

Examples

```
>>> data = attach_demux_results('raw_data.h5', rna_data)
```

Miscellaneous

search_genes(data, gene_list[, rec_key, measure])	Extract and display gene expressions for each cluster from an *anndata* object.
search_de_genes(data, gene_list[, rec_key, ...])	Extract and display differential expression analysis results of markers for each cluster.
find_markers(data, label_attr[, de_key, ...])	Find markers using gradient boosting method.

pegasus.search_genes

pegasus.**search_genes**(*data*, *gene_list*, *rec_key='de_res'*, *measure='percentage'*)

Extract and display gene expressions for each cluster from an *anndata* object.

This function helps to see marker expressions in clusters via the interactive python environment.

Parameters

- **data** (anndata.AnnData) – Annotated data matrix containing the expression matrix and differential expression results.
- **gene_list** (List[str]) – A list of gene symbols.
- **rec_key** (str, optional, default: "de_res") – Keyword of DE analysis result stored in data.varm.
- **measure** (str, optional, default: "percentage") –

Can be either `"percentage"` or `"mean_logExpr"`:

- `percentage` shows the percentage of cells expressed the genes;
- `mean_logExpr` shows the mean log expression.

Returns A data frame containing marker expressions in each cluster.

Return type `pandas.DataFrame`

Examples

```
>>> results = pg.search_genes(adata, ['CD3E', 'CD4', 'CD8'])
```

pegasus.search_de_genes

`pegasus.`**`search_de_genes`**(*data*, *gene_list*, *rec_key='de_res'*, *de_test='fisher'*, *de_alpha=0.05*, *thre=1.5*)

Extract and display differential expression analysis results of markers for each cluster.

This function helps to see if markers are up or down regulated in each cluster via the interactive python environment:

- ++ indicates up-regulated and fold change >= threshold;
- + indicates up-regulated but fold change < threshold;
- −− indicates down-regulated and fold change <= 1 / threshold;
- − indicates down-regulated but fold change > 1 / threshold;
- ? indicates not differentially expressed.

Parameters

- **`data`** (`anndata.Anndata`) – Annotated data matrix containing the expression matrix and differential expression results.
- **`gene_list`** (`List[str]`) – A list of gene symbols.
- **`rec_key`** (`str`, optional, default: `"de_res"`) – Keyword of DE analysis result stored in `data.varm`.
- **`de_test`** (`str`, optional, default: `"fisher"`) – Differential expression test to look at, could be either `t`, `fisher` or `mwu`.
- **`de_alpha`** (`float`, optional, default: `0.05`) – False discovery rate.
- **`thre`** (`float`, optional, default: `1.5`) – Fold change threshold to determine if the marker is a strong DE (++ or −−) or weak DE (+ or −).

Returns A data frame containing marker differential expression results for each cluster.

Return type `pandas.DataFrame`

Examples

```
>>> df = pegasus.misc.search_de_genes(adata, ['CD3E', 'CD4', 'CD8'], thre = 2.0)
```

pegasus.find_markers

pegasus.**find_markers**(*data*, *label_attr*, *de_key='de_res'*, *n_jobs=- 1*, *min_gain=1.0*, *random_state=0*, *remove_ribo=False*)

Find markers using gradient boosting method.

Parameters

- **data** (anndata.AnnData) – Annotated data matrix with rows for cells and columns for genes.
- **label_attr** (str) – Cluster labels used for finding markers. Must exist in data.obs.
- **de_key** (str, optional, default: "de_res") – Keyword of DE analysis result stored in data.varm.
- **n_jobs** (int, optional, default: -1) – Number of threads to used. If -1, use all available threads.
- **min_gain** (float, optional, default: 1.0) – Only report genes with a feature importance score (in gain) of at least min_gain.
- **random_state** (int, optional, default: 0) – Random seed set for reproducing results.
- **remove_ribo** (bool, optional, default: False) – If True, remove ribosomal genes with either RPL or RPS as prefixes.

Returns markers – A Python dictionary containing marker information in structure dict[cluster_id]['up' or 'down'][dataframe].

Return type Dict[str, Dict[str, List[str]]]

Examples

```
>>> marker_dict = pg.find_markers(adata, label_attr = 'leiden_labels')
```

1.1.6 Release Notes

Note: Also see the release notes of PegasusIO.

Version 1.2

1.2.0 December 25, 2020

- tSNE support:
  - `tsne` function in API: Use FIt-SNE for tSNE embedding calculation. No longer support MulticoreTSNE.
  - Determine `learning_rate` argument in `tsne` more dynamically. ([Belkina19], [Kobak19])
  - By default, use PCA embedding for initialization in `tsne`. ([Kobak19])
  - Remove `net_tsne` and `fitsne` functions from API.
  - Remove `--net-tsne` and `--fitsne` options from **pegasus cluster** command.
- Add multimodal support on RNA and CITE-Seq data back: `--citeseq`, `--citeseq-umap`, and `--citeseq-umap-exclude` in **pegasus cluster** command.
- Doublet detection:
  - Add automated doublet cutoff inference to `infer_doublets` function in API. ([Li20-2])
  - Expose doublet detection to command-line tool: `--infer-doublets`, `--expected-doublet-rate`, and `--dbl-cluster-attr` in **pegasus cluster** command.
  - Add doublet detection tutorial.
- Allow multiple marker files used in cell type annotation: `annotate` function in API; `--markers` option in **pegasus annotate_cluster** command.
- Rename *pc_regress_out* function in API to `regress_out`.
- Update the regress out tutorial.
- Bug fix.

Version 1.1

1.1.0 December 7, 2020

- Improve doublet detection in Scrublet-like way using automatic threshold selection strategy: `infer_doublets`, and `mark_doublets`. Remove *Scrublet* from dependency, and remove `run_scrublet` function.
- Enhance performance of log-normalization (`log_norm`) and signature score calculation (`calc_signature_score`).
- In `pegasus cluster` command, add `--genome` option to specify reference genome name for input data of `dge`, `csv`, or `loom` format.
- Update Regress out tutorial.
- Add `ridgeplot`.
- Improve plotting functions: `heatmap`, and `dendrogram`.
- Bug fix.

Version 1.0

1.0.0 September 22, 2020

- New features:
  - Use `zarr` file format to handle data, which has a better I/O performance in general.
  - Multi-modality support:
    - Data are manipulated in Multi-modal structure in memory.
    - Support focus analysis on Unimodal data, and appending other Unimodal data to it. (`--focus` and `--append` options in `cluster` command)
  - Calculate signature / gene module scores. (calc_signature_score)
  - Doublet detection based on Scrublet: `run_scrublet`, `infer_doublets`, and `mark_doublets`.
  - Principal-Component-level regress out. (pc_regress_out)
  - Batch correction using Scanorama. (run_scanorama)
  - Allow DE analysis with sample attribute as condition. (Set `condition` argument in de_analysis)
  - Use static plots to show results (see Plotting):
    - Provide static plots: composition plot, embedding plot (e.g. tSNE, UMAP, FLE, etc.), dot plot, feature plot, and volcano plot;
    - Add more gene-specific plots: dendrogram, heatmap, violin plot, quality-control violin, HVF plot.
- Deprecations:
  - No longer support `h5sc` file format, which was the output format of `aggregate_matrix` command in Pegasus version `0.x`.
  - Remove `net_fitsne` function.
- API changes:
  - In cell quality-control, default percent of mitochondrial genes is changed from **10.0** to **20.0**. (`percent_mito` argument in qc_metrics; `--percent-mito` option in `cluster` command)
  - Move gene quality-control out of `filter_data` function to be a separate step. (identify_robust_genes)
  - DE analysis now uses MWU test by default, not t test. (de_analysis)
  - infer_cell_types uses MWU test as the default `de_test`.
- Performance improvement:
  - Speed up MWU test in DE analysis, which is inspired by Presto.
  - Integrate Fisher's exact test via Cython in DE analysis to improve speed.
- Other highlights:
  - Make I/O and count matrices aggregation a dedicated package PegasusIO.
  - Tutorials:
    - Update Analysis tutorial;
    - Add 3 more tutorials: one on plotting library, one on batch correction and data integration, and one on regress out.

Version 0.x

0.17.2 June 26, 2020

- Make Pegasus compatible with *umap-learn* v0.4+.
- Use *louvain* 0.7+ for Louvain clustering.
- Update tutorial.

0.17.1 April 6, 2020

- Improve pegasus command-line tool log.
- Add human lung markers.
- Improve log-normalization speed.
- Provide robust version of PCA calculation as an option.
- Add signature score calculation API.
- Fix bugs.

0.17.0 March 10, 2020

- Support *anndata* 0.7 and *pandas* 1.0.
- Better `loom` format output writing function.
- Bug fix on `mtx` format output writing function.
- Update human immune cell markers.
- Improve `pegasus scp_output` command.

0.16.11 February 28, 2020

- Add `--remap-singlets` and `--subset-singlets` options to 'cluster' command.
- Allow reading `loom` file with user-specified batch key and black list.

0.16.9 February 17, 2020

Allow reading `h5ad` file with user-specified batch key.

0.16.8 January 30, 2020

Allow input annotated `loom` file.

0.16.7 January 28, 2020

Allow input `mtx` files of more filename formats.

0.16.5 January 23, 2020

Add Harmony algorithm for data integration.

0.16.3 December 17, 2019

Add support for loading mtx files generated from BUStools.

0.16.2 December 8, 2019

Fix bug in 'subcluster' command.

0.16.1 December 4, 2019

Fix one bug in clustering pipeline.

0.16.0 December 3, 2019

- Change options in 'aggregate_matrix' command: remove '–google-cloud', add '–default-reference'.
- Fix bug in '–annotation' option of 'annotate_cluster' command.
- Fix bug in 'net_fle' function with 3-dimension coordinates.
- Use **fisher** package version 0.1.9 or above, as modifications in our forked **fisher-modified** package has been merged into it.

0.15.0 October 2, 2019

Rename package to *PegasusPy*, with module name *pegasus*.

0.14.0 September 17, 2019

Provide Python API for interactive analysis.

0.10.0 January 31, 2019

Added 'find_markers' command to find markers using LightGBM.

Improved file loading speed and enabled the parsing of channels from barcode strings for cellranger aggregated h5 files.

0.9.0 January 17, 2019

In 'cluster' command, changed '–output-seurat-compatible' to '–make-output-seurat-compatible'. Do not generate output_name.seurat.h5ad. Instead, output_name.h5ad should be able to convert to a Seurat object directly. In the seurat object, raw.data slot refers to the filtered count data, data slot refers to the log-normalized expression data, and scale.data refers to the variable-gene-selected, scaled data.

In 'cluster' command, added '–min-umis' and '–max-umis' options to filter cells based on UMI counts.

In 'cluster' command, '–output-filtration-results' option does not require a spreadsheet name anymore. In addition, added more statistics such as median number of genes per cell in the spreadsheet.

In 'cluster' command, added '–plot-filtration-results' and '–plot-filtration-figsize' to support plotting filtration results. Improved documentation on 'cluster command' outputs.

Added 'parquet' command to transfer h5ad file into a parquet file for web-based interactive visualization.

0.8.0 November 26, 2018

Added support for checking index collision for CITE-Seq/hashing experiments.

0.7.0 October 26, 2018

Added support for CITE-Seq analysis.

0.6.0 October 23, 2018

Renamed scrtools to scCloud.

Added demuxEM module for cell/nuclei-hashing.

0.5.0 August 21, 2018

Fixed a problem related AnnData.

Added support for BigQuery.

0.4.0 August 2, 2018

Added mouse brain markers.

Allow aggregate matrix to take 'Sample' as attribute.

0.3.0 June 26, 2018

scrtools supports fast preprocessing, batch-correction, dimension reduction, graph-based clustering, diffusion maps, force-directed layouts, and differential expression analysis, annotate clusters, and plottings.

1.1.7 References

1.1.8 Contributors

Besides the Pegasus team, we would like to sincerely give our thanks to the following contributors to this project:

Name	Commits	Date	Note
Tariq Daouda	774c2cc	2019/10/17	Decorator for time logging and garbage collection.

1.1.9 Contact us

If you have any questions related to Pegasus, please feel free to contact us via Cumulus Support Google Group.

BIBLIOGRAPHY

[Belkina19] A. C. Belkina, C. O. Ciccolella, R. Anno, R. Halpert, J. Spidlen, and J. E. Snyder-Cappione, "Automated optimized parameters for T-distributed stochastic neighbor embedding improve visualization and analysis of large datasets", In Nature Communications, 2019.

[Blondel08] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks", In Journal of Statistical Mechanics: Theory and Experiment, 2008.

[Büttner18] M. Büttner, et al., "A test metric for assessing single-cell RNA-seq batch correction", In Nature Methods, 2018.

[Hie19] B. Hie, B. Bryson, and B. Berger, "Efficient integration of heterogeneous single-cell transcriptomes using Scanorama", In Nature Biotechnology, 2019.

[Jacomy14] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian, "ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software", In PLoS ONE, 2014.

[Kobak19] D. Kobak, and P. Berens, "Kobak, D., & Berens, P. (2019). The art of using t-SNE for single-cell transcriptomics", In Nature Communications, 2019.

[Korsunsky19] I. Korsunsky, et al., "Fast, sensitive and accurate integration of single-cell data with Harmony", In Nature Methods, 2019.

[Li-and-Wong03] C. Li, and W.H. Wong, "DNA-Chip analyzer (dChip)", In The Analysis of Gene Expression Data, Page 120-141, Springer, 2003.

[Li20] B. Li, J. Gould, Y. Yang, et al., "Cumulus provides cloud-based data analysis for large-scale single-cell and single-nucleus RNA-seq", In Nature Methods, 2020.

[Li20-1] B. Li, "A streamlined method for signature score calculation", In Pegasus repository, 2020.

[Li20-2] B. Li, "Doublet detection in Pegasus", In Pegasus repository, 2020.

[Linderman19] G.C. Linderman, et al., "Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data", In Nature Methods, 2019.

[Jerby-Arnon18] L. Jerby-Arnon, et al., "A cancer cell program promotes T cell exclusion and resistance to checkpoint blockade", In Cell, 2018.

[Maaten08] L. van der Maaten, and G. Hinton, "Visualizing data using t-SNE", In Journal of Machine Learning Research, 2008.

[Malkov16] Yu. A. Malkov, and D.A. Yashunin, "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs", In arXiv, 2016.

[McInnes18] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction", Preprint at arXiv, 2018.

[Pijuan-Sala19] B. Pijuan-Sala, et al., "A single-cell molecular map of mouse gastrulation and early organogenesis", In Nature, 2019.

[Popescu19] D-M. Popescu, et al., "Decoding human fetal liver haematopoiesis", In Nature, 2019.

[Traag19] V. A. Traag, L. Waltman, and N. J. van Eck, "From Louvain to Leiden: guaranteeing well-connected communities", In Scientific Reports, 2019.

[Wolock18] S.L. Wolock, R. Lopez, and A.M. Klein, "Scrublet: computational identification of cell doublets in single-cell transcriptomic Data", In Cell Systems, 2018.

INDEX