
Pegasus Documentation

Release 0.17.1

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

Jul 01, 2020

CONTENTS

1	Version 0.17.2 <i>June 26, 2020</i>	3
2	Version 0.17.1 <i>April 6, 2020</i>	5
3	Version 0.17.0 <i>March 10, 2020</i>	7
4	Version 0.16.11 <i>February 28, 2020</i>	9
5	Version 0.16.9 <i>February 17, 2020</i>	11
6	Version 0.16.8 <i>January 30, 2020</i>	13
7	Version 0.16.7 <i>January 28, 2020</i>	15
8	Version 0.16.5 <i>January 23, 2020</i>	17
9	Version 0.16.3 <i>December 17, 2019</i>	19
10	Version 0.16.2 <i>December 8, 2019</i>	21
11	Version 0.16.1 <i>December 4, 2019</i>	23
12	Version 0.16.0 <i>December 3, 2019</i>	25
13	Version 0.15.0 <i>October 2, 2019</i>	27
14	Version 0.14.0 <i>September 17, 2019</i>	29
15	Version 0.10.0 <i>January 31, 2019</i>	31
16	Version 0.9.0 <i>January 17, 2019</i>	33
17	Version 0.8.0 <i>November 26, 2018</i>	35
18	Version 0.7.0 <i>October 26, 2018</i>	37
19	Version 0.6.0 <i>October 23, 2018</i>	39
20	Version 0.5.0 <i>August 21, 2018</i>	41
21	Version 0.4.0 <i>August 2, 2018</i>	43
22	Version 0.3.0 <i>June 26, 2018</i>	45

Bibliography	117
Index	119

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](#)

VERSION 0.17.2 JUNE 26, 2020

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

VERSION 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.

VERSION 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.

VERSION 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.

VERSION 0.16.9 *FEBRUARY 17, 2020*

Allow reading h5ad file with user-specified batch key.

VERSION 0.16.8 *JANUARY 30, 2020*

Allow input annotated loom file.

VERSION 0.16.7 *JANUARY 28, 2020*

Allow input `mtx` files of more filename formats.

VERSION 0.16.5 *JANUARY 23, 2020*

Add Harmony algorithm for data integration.

VERSION 0.16.3 *DECEMBER 17, 2019*

Add support for loading mtx files generated from BUStools.

VERSION 0.16.2 *DECEMBER 8, 2019*

Fix bug in 'subcluster' command.

VERSION 0.16.1 *DECEMBER 4, 2019*

Fix one bug in clustering pipeline.

VERSION 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.

VERSION 0.15.0 *OCTOBER 2, 2019*

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

VERSION 0.14.0 *SEPTEMBER 17, 2019*

Provide Python API for interactive analysis.

VERSION 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.

VERSION 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.

VERSION 0.8.0 *NOVEMBER 26, 2018*

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

VERSION 0.7.0 *OCTOBER 26, 2018*

Added support for CITE-Seq analysis.

VERSION 0.6.0 *OCTOBER 23, 2018*

Renamed scrtools to scCloud.

Added demuxEM module for cell/nuclei-hashing.

VERSION 0.5.0 *AUGUST 21, 2018*

Fixed a problem related AnnData.

Added support for BigQuery.

VERSION 0.4.0 *AUGUST 2, 2018*

Added mouse brain markers.

Allow aggregate matrix to take 'Sample' as attribute.

VERSION 0.3.0 *JUNE 26, 2018*

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

22.1 Installation

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

22.1.1 Linux

The installation has been tested on Ubuntu (18.04 and 19.04).

Prerequisites

First, install the following dependencies by:

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

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:

```
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  
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 pip
```

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
```

22.1.2 macOS

Prerequisites

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

```
brew install cmake libxml2 curl 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:

```
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 pip
```

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
```

22.1.3 Development Version

To install Pegasus development version directly from its [GitHub repository](#), 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.

22.2 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 14 sub-commands in 8 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, tSNE visualization. The final results will be written into h5ad-formatted 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.
- Plotting:

plot Make static plots, which includes plotting tSNEs by cluster labels and different groups.

ipplot Make interactive plots using plotly. The outputs are HTML pages. You can visualize diffusion maps with this sub-command.

- Subclustering:

view View attribute (e.g. cluster labels) and their values. This subcommand is used to determine cells to run subcluster analysis.

subcluster Perform sub-cluster analyses on a subset of cells from the analyzed data (i.e. 'cluster' output).

- Web-based visualization:

scp_output Generate output files for single cell portal.

parquet Generate a PARQUET file for web-based visualization.

- CITE-Seq:

merge_rna_adt Merge RNA and ADT matrices into one 10x-formatted hdf5 file.

- MISC:

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

22.2.1 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/filtered_gene_bc_matrices_h5.h5
sample_2,bone_marrow,NextSeq,2,GRCh38,/my_dir/sample_2/filtered_gene_bc_matrices_h5.h5
sample_3,pbmc,NextSeq,1,GRCh38,/my_dir/sample_3/filtered_gene_bc_matrices_h5.h5
sample_4,pbmc,NextSeq,2,GRCh38,/my_dir/sample_4/filtered_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
pegasus cluster -p 20 --correct-batch-effect --batch-group-by Source -run-louvain --
↪run-tsne example_10x.h5 example
pegasus de_analysis --labels louvain_labels -p 20 --fisher example.h5ad example_de.
↪xlsx
pegasus annotate_cluster example.h5ad example.anno.txt
pegasus plot composition --cluster-labels louvain_labels --attribute Donor --style_
↪normalized --not-stacked example.h5ad example.composition.pdf
pegasus plot scatter --basis tsne --attributes louvain_labels,Donor example.h5ad_
↪example.scatter.pdf
pegasus ipplot --attribute louvain_labels diffmap_pca example.h5ad example.diffmap.html
```

The above analysis will give you tSNE, louvain cluster labels and diffusion maps in `example.h5ad`. You can investigate donor-specific effects by looking at `example.composition.pdf`, `example.scatter.pdf` plotted tSNE colored by `louvain_labels` and Donor info side-by-side. You can explore the diffusion map in 3D by looking at `example.diffmap.html`. This html maps all diffusion components into 3D using PCA.

If you want to perform subcluster analysis by combining cluster 1 and 3, run the following command:

```
pegasus subcluster -p 20 --correct-batch-effect example.h5ad 1,3 example_sub
```

22.2.2 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 --minimum-number-of-genes <ngene>]
    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/filtered_gene_
↪bc_matrices_h5.h5
sample_2,bone_marrow,NextSeq,2,GRCh38,/my_dir/sample_2/filtered_gene_
↪bc_matrices_h5.h5
sample_3,pbmc,NextSeq,1,GRCh38,/my_dir/sample_3/filtered_gene_bc_
↪matrices_h5.h5
sample_4,pbmc,NextSeq,2,GRCh38,/my_dir/sample_4/filtered_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 specify 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.

--minimum-number-of-genes <ngene> Only keep barcodes with at least <ngene> expressed genes.

-h, --help Print out help information.

- Outputs:

output_name.h5sc A pegasus-formatted HDF5 file containing the count matrices and associated attributes.

- Examples:

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

22.2.3 pegasus demuxEM

If you have data generated by cell-hashing or nuclei-hashing, you can use `pegasus demuxEM` to demultiplex your data.

Type:

```
pegasus demuxEM -h
```

to see the usage information:

```
Usage:  
  pegasus demuxEM [options] <input_adt_csv_file> <input_raw_gene_bc_matrices_h5.  
↪h5> <output_name>  
  pegasus demuxEM -h
```

- Arguments:

input_adt_csv_file Input ADT (antibody tag) count matrix in CSV format.

input_raw_gene_bc_matrices_h5.h5 Input raw RNA expression matrix in 10x hdf5 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.
- h, --help** Print out help information.

- Outputs:

output_name_demux.h5sc RNA expression matrix with demultiplexed sample identities in pegasus HDF5 format.

output_name_ADts.h5ad Antibody tag matrix in h5ad format.

output_name_demux.h5ad Demultiplexed RNA count matrix in h5ad format.

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 --hash-type cell-hashing --generate-diagnostic-plots example_
↪adt.csv example_raw_gene_bc_matrices_h5.h5 example_output
```

22.2.4 pegasus cluster

Once we collected the count matrix in 10x (`example_10x.h5`) or pegasus (`example.h5sc`) 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 10x or pegasus format. If first-pass analysis has been performed, but you want to run some additional analysis, you could also pass a h5ad-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 and thus no PCA & diffmap will be run.

--considered-refs <ref_list> A string contains comma-separated reference(e.g. genome) names. pegasus will read all groups associated with reference names in the list from the input file. If `<ref_list>` is None, all groups will be considered. 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, “” is used as genome name instead.

--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”.

--min-genes-on-row <number> If input are raw 10x matrix, which include all barcodes, perform a pre-filtration step to keep the data size small. In the pre-filtration step, only keep cells with at least `<number>` of genes. [default: 100]

--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 sample 1 and 3 using “sample1,sample3”.

--cite-seq Data are CITE-Seq data. pegasus will perform analyses on RNA count matrix first. Then it will attach the ADT matrix to the RNA matrix with all antibody names changing to ‘AD-’ +

antibody_name. Lastly, it will embed the antibody expression using FIt-SNE (the basis used for plotting is 'citeseq_fitsne').

- cite-seq-capping <percentile>** For CITE-Seq surface protein expression, make all cells with expression $> \text{<percentile>}$ to the value at <percentile> to smooth outlier. Set <percentile> to 100.0 to turn this option off. [default: 99.99]
- 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).
- output-seurat-compatible** Output seurat-compatible h5ad file. Caution: File size might be large, do not turn this option on for large data sets.
- output-loom** Output loom-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. [default: 100]
- max-umis <number>** Only keep cells with less than <number> of UMIs. [default: 600000]
- mito-prefix <prefix>** Prefix for mitochondrial genes. If multiple prefixes are provided, separate them by comma (e.g. "MT-,mt-"). [default: MT-]
- percent-mito <ratio>** Only keep cells with mitochondrial ratio less than <ratio> . [default: 0.1]
- gene-percent-cells <ratio>** Only use genes that are expressed in at $\text{<ratio>} * 100$ percent of cells to select variable genes. [default: 0.0005]
- 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). [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 $value_{i1}, \dots, value_{in_i}$.
A cell belongs to group 0 if it does not belong to any other groups.

- random-state <seed>** Random number generator seed. [default: 0]
- temp-folder <temp_folder>** Joblib temporary folder for memmapping numpy arrays.
- 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: 50]
- diffmap-alpha <alpha>** Power parameter for diffusion-based pseudotime. [default: 0.5]
- diffmap-solver <solver>** Solver for eigen decomposition, either ‘randomized’ or ‘eigsh’. [default: randomized]
- 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 AnnData. [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 AnnData. [default: leiden_labels]
- spectral-louvain** Run spectral-louvain clustering algorithm.
- spectral-louvain-basis <basis>** Basis used for KMeans clustering. Can be ‘pca’, or ‘diffmap’. [default: diffmap]
- spectral-louvain-nclusters <number>** Number of clusters for Kmeans initialization. [default: 30]
- spectral-louvain-ninit <number>** Number of Kmeans tries. [default: 20]
- spectral-louvain-resolution <resolution>**. Resolution parameter for louvain. [default: 1.3]

--spectral-louvain-class-label <label> Spectral-louvain label name in AnnData. [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'. [default: diffmap]

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

--spectral-leiden-ninit <number> Number of Kmeans tries. [default: 20]

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

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

--tsne Run multi-core t-SNE for visualization.

--fitsne Run FIt-SNE for visualization.

--tsne-perplexity <perplexity> t-SNE's perplexity parameter, used by both tSNE, FItSNE and net-tSNE. [default: 30]

--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 algorithm. [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-tsne Run net tSNE for visualization.

--net-tsne-polish-learning-frac <frac> After running the deep regressor to predict new coordinates, use <frac> * nsample as the learning rate to use to polish the coordinates. [default: 0.33]

--net-tsne-polish-niter <niter> Number of iterations for polishing tSNE run. [default: 150]

--net-tsne-out-basis <basis> Output basis for net-tSNE. [default: net_tsne]

--run-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]

-h, --help Print out help information.

- Outputs:

output_name.h5ad Output file in h5ad format. To load this file in python, use `import pegasus; data = pegasus.tools.read_input('output_name.h5ad', mode = 'a')`. 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 clustering 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 coordinates. 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']`). If `'--make-output-seurat-compatible'` is on, this file can be loaded into R and converted into a Seurat object.

output_name.seruat.h5ad Optional output. Only exists if `'--output-seruat-compatible'` is set. `'output_name.h5ad'` in seurat-compatible manner. This file can be loaded into R and converted into a Seurat object.

output_name.loom Optional output. Only exists if `'--output-loom'` is set. `'output_name.h5ad'` in loom format for visualization.

output_name.filt.xlsx Optional output. Only exists if `'--output-filtration-results'` is set. 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-row'` 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.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.

output_name.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.

output_name.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.

- Examples:

```
pegasus cluster -p 20 --correct-batch-effect --louvain --tsne example_10x.h5
↪example
pegasus cluster -p 20 --leiden --umap --net-fle example.h5sc example
```

22.2.5 pegasus de_analysis

Once we have the clusters, we can detect markers using `pegasus de_analysis`.

Type:

```
pegasus de_analysis -h
```

to see the usage information:

```
Usage:
  pegasus de_analysis [options] <input_h5ad_file> <output_spreadsheet>
  pegasus de_analysis -h
```

- Arguments:

input_h5ad_file Single cell data with clustering calculated. DE results would be written back.

output_spreadsheet Output spreadsheet with DE results.

- Options:

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

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

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

--auc Calculate area under ROC (AUROC) and area under Precision-Recall (AUPR).

--t Calculate Welch’s t-test.

--fisher Calculate Fisher’s exact test.

--mwu Calculate Mann-Whitney U 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_h5ad_file DE results would be written back to the 'varm' field with name set by '--result-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.

- Examples:

```
pegasus de_analysis -p 26 --labels louvain_labels --auc --t --fisher --mwu_
↪example.h5ad example_de.xlsx
```

22.2.6 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_h5ad_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.

--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.h5ad example.markers.xlsx
```

22.2.7 pegasus annotate_cluster

Once we have the DE results, we could optionally identify putative cell types for each cluster using `pegasus annotate_cluster`. Currently, this subcommand works for human/mouse immune/brain cells. This command has two forms: the first form generates putative annotations and the second form write annotations into the h5ad 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_h5ad_file> <output_file>
  pegasus annotate_cluster --annotation <annotation_string> <input_h5ad_file>
  pegasus annotate_cluster -h
```

- Arguments:

input_h5ad_file Single cell data with DE analysis done by `pegasus de_analysis`.

output_file Output annotation file.

- Options:

--marker-file <file> JSON file for markers. Could also be human/immune/mouse/immune/mouse/brain/human/brain, which triggers pegasus to markers included in the package. [default: human/immune]

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

--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_h5ad_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 h5ad 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.h5ad example.anno.txt
pegasus annotate_cluster --annotation "anno:louvain_labels:T cells;B cells;NK_
↳cells;Monocytes" example.h5ad
```

22.2.8 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>...] <plot_type> <input_
↳h5ad_file> <output_file>
  pegasus plot -h
```

- Arguments:

plot_type Only 2D plots, chosen from ‘composition’, ‘scatter’, ‘scatter_groups’, ‘scatter_genes’, ‘scatter_gene_groups’, ‘heatmap’, and ‘qc_violin’.

input_h5ad_file Single cell data in h5ad file format with clustering done by `pegasus cluster`.

output_file Output image file.

- Options:

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

--cluster-labels <attr> Use <attr> as cluster labels. This option is used in ‘composition’, ‘scatter_groups’, ‘heatmap’, and ‘qc_violin’.

--attribute <attr> Plot <attr> against cluster labels. This option is only used in ‘composition’ and ‘qc_violin’.

--basis <basis> Basis for 2D plotting, chosen from ‘tsne’, ‘fitsne’, ‘umap’, ‘pca’, ‘rpca’, ‘fle’, ‘diffmap_pca’, ‘net_tsne’, ‘net_fitsne’, ‘net_umap’ or ‘net_fle’. If CITE-Seq data is used, basis can also be ‘citeseq_fitsne’. This option is used in ‘scatter’, ‘scatter_groups’, ‘scatter_genes’, and ‘scatter_gene_groups’. [default: fitsne]

--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’.

- apply-to-each-figure** Indicate that the <restriction> strings are not applied to all attributes but for specific attributes. The string's 'attr' value should match the attribute you want to restrict.
- show-background** Show points that are not selected as gray.
- group <attr>** <attr> is used to make group plots. In group plots, the first one contains all components in the group and the following plots show each component separately. This option is used in 'scatter_groups' and 'scatter_gene_groups'. If <attr> is a semi-colon-separated string, parse the string as groups.
- genes <genes>** <genes> is a comma-separated list of gene names to visualize. This option is used in 'scatter_genes' and 'heatmap'.
- gene <gene>** Visualize <gene> in group plots. This option is only used in 'scatter_gene_groups'.
- style <style>** Composition plot styles. Can be either 'frequency', 'count', or 'normalized'. [default: frequency]
- not-stacked** Do not stack bars in composition plot.
- log-y** Plot y axis in log10 scale for composition plot.
- 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.
- subplot-size <sizes>** Sub-plot 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 subplot width.
- bottom <bottom>** Figure's bottom margin in fraction with respect to subplot height.
- wspace <wspace>** Horizontal space between subplots in fraction with respect to subplot width.
- hspace <hspace>** Vertical space between subplots in fraction with respect to subplot height.
- alpha <alpha>** Point transparent parameter.
- legend-fontsize <fontsize>** Legend font size.
- use-raw** Use anndata stored raw expression matrix. Only used by 'scatter_genes' and 'scatter_gene_groups'.
- do-not-show-all** Do not show all components in group for scatter_groups.
- show-zscore** If show zscore in heatmap.
- heatmap-title <title>** Title for heatmap.
- qc-type <type>** Plot qc_violin by annotation, <type> can be either 'gene', 'count' (UMI), or 'mito' (mitochondrial rate). [default: gene]
- qc-xtick-font ** x tick font for qc_violin. [default: 5]
- qc-xtick-rotation** If rotate x label.
- qc-line-width <width>** Line width for qc_violin. [default: 0.5]
- h, --help** Print out help information.

Examples:

```
pegasus plot composition --cluster-labels louvain_labels --attribute Donor --style_
↪normalized --not-stacked example.h5ad example.composition.pdf
pegasus plot scatter --basis tsne --attributes louvain_labels,Donor example.h5ad_
↪example.scatter.pdf
```

(continues on next page)

(continued from previous page)

```
pegasus plot scatter_groups --cluster-labels louvain_labels --group Donor example.  
↪h5ad example.scatter_groups.pdf  
pegasus plot scatter_genes --genes CD8A,CD4,CD3G,MS4A1,NCAM1,CD14,ITGAX,IL3RA,CD38,  
↪CD34,PPBP example.h5ad example.genes.pdf  
pegasus plot scatter_gene_groups --gene CD8A --group Donor example.h5ad example.gene_  
↪groups.pdf  
pegasus plot heatmap --cluster-labels louvain_labels --genes CD8A,CD4,CD3G,MS4A1,  
↪NCAM1,CD14,ITGAX,IL3RA,CD38,CD34,PPBP --heatmap-title 'markers' example.h5ad_  
↪example.heatmap.pdf  
pegasus plot qc_violin --qc-type gene --cluster-labels louvain_labels --attribute_  
↪Channel --subplot-size 7,5 --qc-xtick-font 5 --qc-line-width 0.5 example.h5ad_  
↪example.qc_violin.pdf
```

22.2.9 pegasus iplot

We can also make interactive plots in html format using `pegasus iplot`. These interactive plots are very helpful if you want to explore the diffusion maps.

Type:

```
pegasus iplot -h
```

to see the usage information:

```
Usage:  
    pegasus iplot --attribute <attr> [options] <basis> <input_h5ad_file> <output_  
↪html_file>  
    pegasus iplot -h
```

- Arguments:

basis Basis can be either 'tsne', 'fitsne', 'umap', 'diffmap', 'pca', or 'diffmap_pca'.

input_h5ad_file Single cell data with clustering done in h5ad file format.

output_html_file Output interactive plot in html format.

- Options:

--attribute <attr> Use attribute <attr> as labels in the plot.

--is-real <attr> is real valued.

--is-gene <attr> is a gene name.

--log10 If take log10 of real values.

-h, --help Print out help information.

- Examples:

```
pegasus iplot --attribute louvain_labels tsne example.h5ad example.tsne.html  
pegasus iplot --attribute louvain_labels diffmap_pca example.h5ad example.diffmap.  
↪html
```

22.2.10 pegasus view

We may want to further perform sub-cluster analysis on a subset of cells. This sub-command helps us to define the subset.

Type:

```
pegasus view -h
```

to see the usage information:

```
Usage:
    pegasus view [--show-attributes --show-gene-attributes --show-values-for-
    ↪attributes <attributes>] <input_h5ad_file>
    pegasus view -h
```

- Arguments:
 - input_h5ad_file** Analyzed single cell data in h5ad format.
- Options:
 - show-attributes** Show the available sample attributes in the input dataset.
 - show-gene-attributes** Show the available gene attributes in the input dataset.
 - show-values-for-attributes <attributes>** Show the available values for specified attributes in the input dataset. <attributes> should be a comma-separated list of attributes.
 - h, --help** Print out help information.
- Examples:

```
pegasus view --show-attributes example.h5ad
pegasus view --show-gene-attributes example.h5ad
pegasus view --show-values-for-attributeslouvain_labels,Donor example.h5ad
```

22.2.11 pegasus subcluster

If there is a subset of cells that we want to further cluster, we can run `pegasus subcluster`. This sub-command will outputs a new h5ad file that you can run `de_analysis`, `plot` and `iplot` on.

Type:

```
pegasus subcluster -h
```

to see the usage information:

```
Usage:
    pegasus subcluster [options] --subset-selection <subset-selection>... <input_
    ↪file> <output_name>
    pegasus subcluster -h
```

- Arguments:
 - input_file** Single cell data with clustering done in h5ad format.
 - output_name** Output file name. All outputs will use it as the prefix.

- Options:

- subset-selection <subset-selection>...** Specify which cells will be included in the subcluster analysis. Each <subset_selection> string takes the format of 'attr:value,...,value', which means select cells with attr in the values. If multiple <subset_selection> strings are specified, the subset of cells selected is the intersection of these strings.
- p <number>, --threads <number>** Number of threads. [default: 1]
- correct-batch-effect** Correct for batch effects.
- batch-group-by** 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+pegasus..+attrn', or 'attr=value11,pegasus..,value1n_1;value21,pegasus..,value2n_2;pegasus..;valuem1,pegasus..,valuemn_m'. In the first form, 'attr' should be an existing sample attribute, and groups are defined by 'attr'. In the second form, 'attr1', 'pegasus..', '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,pegasus..,valuein_i. A cell belongs to group 0 if it does not belong to any other groups.
- output-loom** Output loom-formatted file.
- 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 <nfeatures> 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.
- random-state <seed>** Random number generator seed. [default: 0]
- temp-folder <temp_folder>** Joblib temporary folder for memmapping numpy arrays.
- 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: 50]
- diffmap-alpha <alpha>** Power parameter for diffusion-based pseudotime. [default: 0.5]

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

--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 AnnData. [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 AnnData. [default: leiden_labels]

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

--spectral-louvain-basis <basis> Basis used for KMeans clustering. Can be ‘pca’ or ‘diffmap’. [default: diffmap]

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

--spectral-louvain-ninit <number> Number of Kmeans tries. [default: 20]

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

--spectral-louvain-class-label <label> Spectral-louvain label name in AnnData. [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’. [default: diffmap]

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

--spectral-leiden-ninit <number> Number of Kmeans tries. [default: 20]

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

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

--tsne Run multi-core t-SNE for visualization.

--run-fitsne Run Fit-SNE for visualization.

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

--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 algorithm. [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^α . [default: 1.0]

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

--net-tsne Run net tSNE for visualization.

--net-tsne-polish-learning-frac <frac> After running the deep regressor to predict new coordinates, use <frac> * nsample as the learning rate to use to polish the coordinates. [default: 0.33]

--net-tsne-polish-niter <niter> Number of iterations for polishing tSNE run. [default: 150]

--net-tsne-out-basis <basis> Output basis for net-tSNE. [default: net_tsne]

--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]

-h, --help Print out help information.

- Outputs:

output_name.h5ad Output file in h5ad format. The clustering results are stored in the ‘obs’ field (e.g. ‘louvain_labels’ for louvain cluster labels). The PCA, t-SNE and diffusion map coordinates are stored in the ‘obsm’ field.

output_name.loom Optional output. Only exists if ‘--output-loom’ is set. output_name.h5ad in loom format for visualization.

- Examples:

```
pegasus subcluster -p 20 --correct-batch-effect --subset-selection louvain_
↪ labels:3,6 --subset-selection Condition:CB_nonmix --tsne --louvain manton_bm.
↪ h5ad manton_bm_subset
```


22.2.12 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_h5ad_file> <output_name>
    pegasus scp_output -h
```

- Arguments:

input_h5ad_file Analyzed single cell data in h5ad 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.tsv
Files that single cell portal needs.

- Examples:

```
pegasus scp_output example.h5ad example
```

22.2.13 pegasus parquet

Generate a PARQUET file for web-based visualization.

Type:

```
pegasus parquet -h
```

to see the usage information:

```
Usage:
    pegasus parquet [options] <input_h5ad_file> <output_name>
    pegasus parquet -h
```

- Arguments:

input_h5ad_file Analyzed single cell data in h5ad format.

output_name Name prefix for the parquet file.

- Options:

-p <number>, --threads <number> Number of threads used to generate the PARQUET file. [default: 1]

-h, --help Print out help information.

- Outputs:

output_name.parquet Generated PARQUET file that contains metadata and expression levels for every gene.

- Examples:

```
pegasus parquet example.h5ad example.parquet
```

22.2.14 pegasus merge_rna_adt

If we have CITE-Seq data, we can merge RNA count matrix and ADT (antibody tag) count matrix into one file using this subcommand.

Type:

```
pegasus merge_rna_adt -h
```

to see the usage information:

```
Usage:
  pegasus merge_rna_adt <input_raw_gene_bc_matrices_h5.h5sc> <input_adt_csv_
↪file> <output_name>
  pegasus merge_rna_adt -h
```

- Arguments:

input_raw_gene_bc_matrices_h5.h5sc Input raw RNA expression matrix in pegasus hdf5 format.

input_adt_csv_file Input ADT (antibody tag) count matrix in CSV format.

output_name Merged output name.

- Options:

--antibody-control-csv <antibody_control_csv_file> A CSV file containing the IgG control information for each antibody.

-h, --help Print out help information.

- Outputs:

output_name.h5sc Output file in pegasus hdf5 format. This file contains two groups — one for RNAs, and the other for ADTs.

- Examples:

```
pegasus merge_rna_adt example_raw_h5.h5sc example_adt.csv example_merged_raw
pegasus merge_rna_adt --antibody-control-csv antibody_control.csv example_raw_h5.
↪h5sc example_adt.csv example_merged_raw
```

22.2.15 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
```

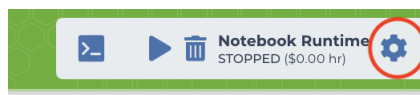
22.3 Use Pegasus on Terra Notebook

You need to first have a [Terra](#) account.

22.3.1 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):

RUNTIME CONFIGURATION ×

Create cloud compute to launch Jupyter Notebooks or a Project-Specific software application.

ENVIRONMENT ⓘ

Custom Environment ▼

CONTAINER IMAGE

cumulusprod/pegasus-terra:0.17

Custom environments **must** be based off one of the [Terra Jupyter Notebook base images](#) or a [Project-Specific image](#)

COMPUTE POWER

Select from one of the default runtime profiles or define your own

Profile

Default (Moderate) computer power ▼

CPUs

4

Memory (GB)

15

Disk size (GB)

50

Runtime type

Standard VM ▼

COST: \$0.20 per hour

CANCEL
CREATE

Here are the key settings:

- 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 minor versions of Pegasus. For example, if you want to use Pegasus version 0.16.x, you should type cumulusprod/pegasus-terra:0.16.
- 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 Red-Hat 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.

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:



22.3.2 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__
```

22.3.3 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 .
```

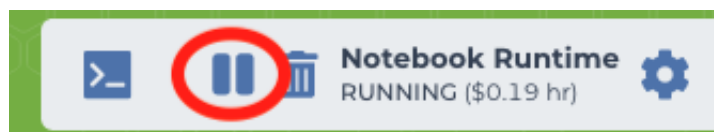
where `gs://link-to-count-matrix` is the Google Bucket URL to your count matrix data file.

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

Please refer to [tutorials](#) for how to use Pegasus on Terra notebook.

22.3.4 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.

22.4 Tutorials

- [Pegasus Tutorial](#)

22.5 API

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

```
import pegasus as pg
```

22.5.1 Analysis Tools

Read and Write

<code>read_input(input_file[, genome, ...])</code>	Load data into memory.
<code>write_output(data, output_file[, whitelist])</code>	Write data back to disk.
<code>aggregate_matrices(csv_file[, ...])</code>	Aggregate channel-specific count matrices into one big count matrix.

`pegasus.read_input`

```
pegasus.read_input(input_file, genome=None, return_type='AnnData', concat_matrices=False,
                   h5ad_mode='a', ngene=None, select_singlets=False, channel_attr=None,
                   chunk_size=None, black_list=[])
```

Load data into memory.

This function is used to load input data into memory. Inputs can be in 10x genomics v2 & v3 formats (hdf5 or mtx), HCA DCP mtx and csv formats, Drop-seq dge format, and CSV format.

Parameters

- **input_file** (*str*) – Input file name.
- **genome** (*str*, optional (default: `None`)) – A string contains comma-separated genome names. *pegasus* will read all matrices matching the genome names. If *genome* is `None`, all matrices will be considered. 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, “” is used as the genome name instead.
- **return_type** (*str*) – Return object type, can be either ‘MemData’ or ‘AnnData’.
- **concat_matrices** (*boolean*, optional (default: `False`)) – If input file contains multiple matrices, turning this option on will concatenate them into one `AnnData` object. Otherwise return a list of `AnnData` objects.
- **h5ad_mode** (*str*, optional (default: ‘a’)) – If input is in h5ad format, the backed mode for loading the data. Mode could be ‘a’, ‘r’, ‘r+’, where ‘a’ refers to load the whole matrix into memory.
- **ngene** (*int*, optional (default: `None`)) – Minimum number of genes to keep a barcode. Default is to keep all barcodes.

- **select_singlets** (*bool*, optional (default: False)) – If this option is on, only keep DemuxEM-predicted singlets when loading data.
- **channel_attr** (*str*, optional (default: None)) – Use channel_attr to represent different samples. This will set a ‘Channel’ column field with channel_attr.
- **chunk_size** (*int*, optional (default: None)) – Chunk size for reading dense matrices as sparse
- **black_list** (*List[str]*, optional (default: [])) – Attributes in black list will be popped out.

Returns An *MemData* object or *anndata* object or a list of *anndata* objects containing the count matrices.

Return type *MemData* object or *anndata* object or a list of *anndata* objects

Examples

```
>>> adata = pg.read_input('example_10x.h5', genome = 'mm10')
>>> adata = pg.read_input('example.h5ad', h5ad_mode = 'r+')
>>> adata = pg.read_input('example_ADT.csv')
```

pegasus.write_output

`pegasus.write_output` (*data*, *output_file*, *whitelist*=[*'obs'*, *'obsm'*, *'uns'*, *'var'*, *'varm'*])

Write data back to disk.

This function is used to write data back to disk.

Parameters

- **data** (*MemData* or *AnnData*) – data to write back, can be either an *MemData* or *AnnData* object.
- **output_file** (*str*) – output file name. If data is *MemData*, output_file should ends with suffix ‘.h5sc’. Otherwise, output_file can end with either ‘.h5ad’, ‘.loom’, or ‘.mtx.gz’. If output_file ends with ‘.loom’, a LOOM file will be generated. If no suffix is detected, an appropriate one will be appended.
- **whitelist** (*list*, optional, default = [*'obs'*, *'obsm'*, *'uns'*, *'var'*, *'varm'*]) – List that indicates changed fields when writing h5ad file in backed mode. For example, [*'uns/Groups'*, *'obs/PCA'*] will only write Groups in uns, and PCA in obsm; the rest of the fields will be unchanged.

Returns

Return type *None*

Examples

```
>>> pg.write_output(adata, 'test.h5ad')
```

pegasus.aggregate_matrices

```
pegasus.aggregate_matrices(csv_file, what_to_return='AnnData', restrictions=[], attributes=[],
                           default_ref=None, select_singlets=False, ngene=None, concat_matrices=False)
```

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. Depending on `what_to_return`, it can output the merged results into a pegasus-formatted HDF5 file or return as an `AnnData` or `MemData` object.

Parameters

- **csv_file** (*str*) – The CSV file containing information about each channel.
- **what_to_return** (*str*, optional (default: 'AnnData')) – If this value is equal to 'AnnData' or 'MemData', an `AnnData` or `MemData` object will be returned. Otherwise, results will be written into 'what_to_return.h5sc' file and `None` is returned.
- **restrictions** (*list[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.
- **attributes** (*list[str]*, optional (default: [])) – A list of attributes need to be incorporated into the output count matrix.
- **default_ref** (*str*, optional (default: None)) – Default reference name to use. If sample count matrix is in either DGE, mtx, csv or tsv format and there is no Reference column in the `csv_file`, `default_ref` will be used as the reference.
- **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.
- **ngene** (*int*, optional (default: None)) – The minimum number of expressed genes to keep one barcode.
- **concat_matrices** (*bool*, optional (default: False)) – If concatenate multiple matrices. If so, return only one `AnnData` object, otherwise, might return a list of `AnnData` objects.

Returns Either `None` or an `AnnData` object or a `MemData` object.

Return type `None` or `AnnData` or `MemData`

Examples

```
>>> pg.aggregate_matrix('example.csv', 'example_10x.h5', ['Source:pbmc', 'Donor:1
↳'], ['Source', 'Platform', 'Donor'])
```

Preprocess

<code>qc_metrics(data[, mito_prefix, min_genes, ...])</code>	Generate Quality Control (QC) metrics on the dataset.
<code>get_filter_stats(data)</code>	Calculate filtration stats on cell barcodes and genes, respectively.
<code>filter_data(data)</code>	Filter data based on qc_metrics calculated in pg.qc_metrics.
<code>log_norm(data[, norm_count])</code>	Normalization, and then apply natural logarithm to the data.
<code>highly_variable_features(data, con- sider_batch)</code>	Highly variable features (HVF) selection.
<code>select_features(data[, features])</code>	Subset the features and store the resulting matrix in dense format in data.uns with 'fmat_' prefix.
<code>pca(data[, n_components, features, ...])</code>	Perform Principle Component Analysis (PCA) to the data.

pegasus.qc_metrics

`pegasus.qc_metrics(data, mito_prefix='MT-', min_genes=500, max_genes=6000, min_umis=100, max_umis=600000, percent_mito=10.0, percent_cells=0.05)`
Generate Quality Control (QC) metrics on the dataset.

Parameters

- **data** (`anndata.AnnData`) – Annotated data matrix with rows for cells and columns for genes.
- **mito_prefix** (`str`, optional, default: "MT-") – Prefix for mitochondrial genes.
- **min_genes** (`int`, optional, default: 500) – Only keep cells with at least `min_genes` genes.
- **max_genes** (`int`, optional, default: 6000) – Only keep cells with less than `max_genes` genes.
- **min_umis** (`int`, optional, default: 100) – Only keep cells with at least `min_umis` UMIs.
- **max_umis** (`int`, optional, default: 600,000) – Only keep cells with less than `max_umis` UMIs.
- **percent_mito** (`float`, optional, default: 10.0) – Only keep cells with percent mitochondrial genes less than `percent_mito` % of total counts.
- **percent_cells** (`float`, optional, default: 0.05) – Only assign genes to be robust that are expressed in at least `percent_cells` % of cells.

Return type `None`

Returns

- `None`

- Update `data.obs` –
 - `n_genes`: Total number of genes for each cell.
 - `n_counts`: Total number of counts for each cell.
 - `percent_mito`: Percent of mitochondrial genes for each cell.
 - `passed_qc`: Boolean type indicating if a cell passes the QC process based on the QC metrics.
- Update `data.var` –
 - `n_cells`: Total number of cells in which each gene is measured.
 - `percent_cells`: Percent of cells in which each gene is measured.
 - `robust`: Boolean type indicating if a gene is robust based on the QC metrics.
 - `highly_variable_features`: Boolean type indicating if a gene is a highly variable feature. By default, set all robust genes as highly variable features.

Examples

```
>>> pg.qcmetrics(adata)
```

`pegasus.get_filter_stats`

`pegasus.get_filter_stats(data)`

Calculate filtration stats on cell barcodes and genes, respectively.

Parameters `data` (`anndata.AnnData`) – Annotated data matrix with rows for cells and columns for genes.

Return type `Tuple[DataFrame, DataFrame]`

Returns

- `df_cells` (`pandas.DataFrame`) – Data frame of stats on cell filtration.
- `df_genes` (`pandas.DataFrame`) – Data frame of stats on gene filtration.

Examples

```
>>> pg.get_filter_stats(adata)
```

`pegasus.filter_data`

`pegasus.filter_data(data)`

Filter data based on `qc_metrics` calculated in `pg.qc_metrics`.

Parameters `data` (`anndata.AnnData`) – Annotated data matrix with rows for cells and columns for genes.

Return type `None`

Returns

- None
- Update data with cells and genes after filtration.

Examples

```
>>> pg.filter_data(adata)
```

pegasus.log_norm

`pegasus.log_norm(data, norm_count=100000.0)`

Normalization, and then apply natural logarithm to the data.

Parameters

- **data** (`anndata.AnnData`) – Annotated data matrix with rows for cells and columns for genes.
- **norm_count** (`int`, optional, default: `1e5`) – Total count of cells after normalization.

Return type None

Returns

- None
- Update data.X with count matrix after log-normalization.

Examples

```
>>> pg.log_norm(adata)
```

pegasus.highly_variable_features

`pegasus.highly_variable_features(data, consider_batch, flavor='pegasus', n_top=2000, span=0.02, min_disp=0.5, max_disp=inf, min_mean=0.0125, max_mean=7, n_jobs=-1)`

Highly variable features (HVF) selection. The input data should be logarithmized.

Parameters

- **data** (`anndata.AnnData`) – Annotated data matrix with rows for cells and columns for genes.
- **consider_batch** (`bool`.) – Whether consider batch effects or not.
- **flavor** (`str`, optional, default: `"pegasus"`) – The HVF selection method to use. Available choices are `"pegasus"` or `"Seurat"`.
- **n_top** (`int`, optional, default: `2000`) – Number of genes to be selected as HVF. if None, no gene will be selected.
- **span** (`float`, optional, default: `0.02`) – Only applicable when `flavor` is `"pegasus"`. The smoothing factor used by *scikit-learn loess* model in pegasus HVF selection method.
- **min_disp** (`float`, optional, default: `0.5`) – Minimum normalized dispersion.

- **max_disp** (float, optional, default: `np.inf`) – Maximum normalized dispersion. Set it to `np.inf` for infinity bound.
- **min_mean** (float, optional, default: `0.0125`) – Minimum mean.
- **max_mean** (float, optional, default: `7`) – Maximum mean.
- **n_jobs** (int, optional, default: `-1`) – Number of threads to be used during calculation. If `-1`, all available threads will be used.

Examples

```
>>> pg.highly_variable_features(adata, consider_batch = False)
```

Return type None

pegasus.select_features

`pegasus.select_features(data, features=None)`

Subset the features and store the resulting matrix in dense format in `data.uns` with `'fmat_'` prefix. `'fmat_*`' will be removed before writing out the disk.

Parameters

- **data** (`anndata.AnnData`) – Annotated data matrix with rows for cells and columns for genes.
- **features** (str, optional, default: `None`.) – a keyword in `data.var`, which refers to a boolean array. If `None`, all features will be selected.

Return type str

Returns

- **keyword** (str) – The keyword in `data.uns` referring to the features selected.
- Update `data.uns` –
 - `data.uns[keyword]`: A submatrix of the data containing features selected.

Examples

```
>>> pg.select_features(adata)
```

pegasus.pca

`pegasus.pca(data, n_components=50, features='highly_variable_features', standardize=True, max_value=10, robust=False, random_state=0)`

Perform Principle Component Analysis (PCA) to the data.

The calculation uses *scikit-learn* implementation.

Parameters

- **data** (`anndata.AnnData`) – Annotated data matrix with rows for cells and columns for genes.

- **n_components** (int, optional, default: 50.) – Number of Principal Components to get.
- **features** (str, optional, default: "highly_variable_features".) – Keyword in `data.var` to specify features used for PCA.
- **standardize** (bool, optional, default: True.) – Whether to scale the data to unit variance and zero mean or not.
- **max_value** (float, optional, default: 10.) – The threshold to truncate data after scaling. If None, do not truncate.
- **robust** (bool, optional, default: False.) – If true, use ‘arpack’ instead of ‘randomized’ for large sparse matrices (i.e. $\max(X.\text{shape}) > 500$ and $n_components < 0.8 * \min(X.\text{shape})$)
- **random_state** (int, optional, default: 0.) – Random seed to be set for reproducing result.

Return type None

Returns

- None.
- Update `data.obsm` –
 - `data.obsm["X_pca"]`: PCA matrix of the data.
- Update `data.uns` –
 - `data.uns["PCs"]`: The principal components containing the loadings.
 - `data.uns["pca_variance"]`: Explained variance, i.e. the eigenvalues of the covariance matrix.
 - `data.uns["pca_variance_ratio"]`: Ratio of explained variance.

Examples

```
>>> pg.pca(adata)
```

Batch Correction

<code>set_group_attribute(data, attribute_string)</code>	Set group attributes used in batch correction.
<code>correct_batch(data[, features])</code>	Batch correction on data.
<code>run_harmony(data[, rep, n_jobs, n_clusters, ...])</code>	Batch correction PCs using Harmony

pegasus.set_group_attribute

`pegasus.set_group_attribute(data, attribute_string)`

Set group attributes used in batch correction.

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, *pegasus* will only perform batch correction for channels within each group.

Parameters

- **data** (`anndata.AnnData`) – Annotated data matrix with rows for cells and columns for

genes.

- **attribute_string** (*str*) – Attributes used to construct groups:
 - **None** Assume all channels are from one group.
 - **attr** Define groups by sample attribute *attr*, which is a keyword in *data.obs*.
 - **att1+att2+...+attrn** Define groups by the Cartesian product of these *n* attributes, which are keywords in *data.obs*.
 - **attr=value_11,...,value_1n_1;value_21,...,value_2n_2;...;value_m1,...,value_mn_1**
In this form, there will be $(m+1)$ groups. A cell belongs to group *i* ($i > 1$) if and only if its sample attribute *attr*, which is a keyword in *data.obs*, has a value among *value_i1*,... *value_in_i*. A cell belongs to group 0 if it does not belong to any other groups.

Returns

Update *data.obs*:

- *data.obs*["Group"]: Group ID for each cell.

Return type None

Examples

```
>>> pg.set_group_attribute(adata, attr_string = "Individual")
```

```
>>> pg.set_group_attribute(adata, attr_string = "Individual+assignment")
```

```
>>> pg.set_group_attribute(adata, attr_string = "Channel=1,3,5;2,4,6,8")
```

pegasus.correct_batch

pegasus.correct_batch (*data*, *features=None*)

Batch correction on data.

Parameters

- **data** (*anndata.AnnData*) – Annotated data matrix with rows for cells and columns for genes.
- **features** (*str*, optional, default: *None*) – Features to be included in batch correction computation. If *None*, simply consider all features.

Return type None

Returns

- None
- Update *data.X* by the corrected count matrix.

Examples

```
>>> pg.correct_batch(adata, features = "highly_variable_features")
```

pegasus.run_harmony

`pegasus.run_harmony` (*data*, *rep*='pca', *n_jobs*=-1, *n_clusters*=None, *random_state*=0)

Batch correction PCs using Harmony

Parameters

- **data** (`anndata.AnnData`.) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (`str`, optional, default: "pca".) – Which representation to use as input of Harmony, default is PCA.
- **n_jobs** (`int`, optional, default: -1.) – Number of threads to use for the KMeans clustering used in Harmony. -1 refers to using all available threads.
- **n_clusters** (`int`, optional, default: None.) – Number of Harmony clusters. Default is None, which asks Harmony to estimate this number from the data.
- **random_state** (`int`, optional, default: 0.) – Seed for random number generator

Return type `str`

Returns

- **out_rep** (`str`) – The keyword in `data.obsm` referring to the embedding calculated by Harmony algorithm.
This keyword is `rep + '_harmony'`, where `rep` is the input parameter above.
- Update `data.obsm` –
– `data.obsm['X_' + out_rep]`: The embedding calculated by Harmony algorithm.

Examples

```
>>> pg.run_harmony(adata, rep = "pca", n_jobs = 10, random_state = 25)
```

Nearest Neighbors

<code>neighbors(data[, K, rep, n_jobs, ...])</code>	Compute k nearest neighbors and affinity matrix, which will be used for diffmap and graph-based community detection algorithms.
<code>calc_kBET(data, attr[, rep, K, alpha, ...])</code>	Calculate the kBET metric of the data w.r.t.
<code>calc_kSIM(data, attr[, rep, K, min_rate, ...])</code>	Calculate the kSIM metric of the data w.r.t.

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 hnsplib introduced by [Malkov16].

Parameters

- **data** (`anndata.AnnData`) – 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 itself), sorted from smallest to largest.
 - `data.uns["W_" + rep]`: kNN graph of the data in terms of affinity matrix.

Examples

```
>>> pg.neighbors(adata)
```

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 w.r.t. a specific sample attribute and embedding.

This kBET metric is based on paper “A test metric for assessing single-cell RNA-seq batch correction” [Büttner18] in Nature Methods, 2018.

Parameters

- **data** (anndata.AnnData) – 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(adata, attr = 'Channel')
```

```
>>> pg.calc_kBET(adata, 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 w.r.t. a specific sample attribute and embedding.

This kSIM metric measures if attr are not diffused too much.

Parameters

- **data** (anndata.AnnData) – 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(adata, attr = 'cell_type')
```

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

Diffusion Map

<code>diffmap(data[, n_components, rep, solver, ...])</code>	Calculate Diffusion Map.
<code>reduce_diffmap_to_3d(data[, random_state])</code>	Reduce high-dimensional Diffusion Map matrix to 3-dimensional.
<code>calc_pseudotime(data, roots)</code>	Calculate Pseudotime based on Diffusion Map.
<code>infer_path(data, cluster, clust_id, path_name)</code>	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** (`anndata.AnnData`) – 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 eigenvalues 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(adata)
```

pegasus.reduce_diffmap_to_3d

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

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

Parameters `data` (`anndata.AnnData`) – 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(adata)
```

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

<code>cluster(data[, algo, rep, resolution, ...])</code>	Cluster the data using the chosen algorithm.
<code>louvain(data[, rep, resolution, ...])</code>	Cluster the cells using Louvain algorithm.
<code>leiden(data[, rep, resolution, n_iter, ...])</code>	Cluster the data using Leiden algorithm.
<code>spectral_louvain(data[, rep, resolution, ...])</code>	Cluster the data using Spectral Louvain algorithm.
<code>spectral_leiden(data[, rep, resolution, ...])</code>	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,
                n_jobs=-1)
```

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** (anndata.AnnData) – 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.
- **n_jobs** (int, optional, default: -1) – Number of threads to use. If -1, use all available threads.

Return type None

Returns

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

Examples

```
>>> pg.cluster(adata, 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.

Parameters

- **data** (`anndata.AnnData`) – 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.
- **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(adata)
```

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.

Parameters

- **data** (`anndata.AnnData`) – 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.
- **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(adata)
```

pegasus.spectral_louvain

`pegasus.spectral_louvain` (*data*, *rep*='pca', *resolution*=1.3, *rep_kmeans*='diffmap', *n_clusters*=30, *n_clusters2*=50, *n_init*=10, *n_jobs*=-1, *random_state*=0, *class_label*='spectral_louvain_labels')

Cluster the data using Spectral Louvain algorithm.

Parameters

- **data** (`anndata.AnnData`) – 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.
- **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 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(adata)
```

pegasus.spectral_leiden

```
pegasus.spectral_leiden(data, rep='pca', resolution=1.3, rep_kmeans='diffmap', n_clusters=30,  
                        n_clusters2=50, n_init=10, n_jobs=-1, random_state=0,  
                        class_label='spectral_leiden_labels')
```

Cluster the data using Spectral Leiden algorithm.

Parameters

- **data** (`anndata.AnnData`) – 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.
- **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 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(adata)
```

Visualization Algorithms

<code>tsne(data[, rep, n_jobs, n_components, ...])</code>	Calculate tSNE embedding using MulticoreTSNE_ package.
<code>fitsne(data[, rep, n_jobs, n_components, ...])</code>	Calculate FIt-SNE embedding using fitsne_ package.
<code>umap(data[, rep, n_components, n_neighbors, ...])</code>	Calculate UMAP embedding using umap-learn_ package.
<code>fle(data[, file_name, n_jobs, rep, K, ...])</code>	Construct the Force-directed (FLE) graph using ForceAtlas2_ implementation, with Python wrapper as forceatlas2-python_ .
<code>net_tsne(data[, rep, n_jobs, n_components, ...])</code>	Calculate approximated tSNE embedding using Deep Learning model to improve the speed.
<code>net_fitsne(data[, rep, n_jobs, ...])</code>	Calculate approximated FI-tSNE embedding using Deep Learning model to improve the speed.
<code>net_umap(data[, rep, n_jobs, n_components, ...])</code>	Calculate approximated UMAP embedding using Deep Learning model to improve the speed.
<code>net_fle(data[, file_name, n_jobs, rep, K, ...])</code>	Construct the approximated Force-directed (FLE) graph using Deep Learning model to improve the speed.

pegasus.tsne

`pegasus.tsne(data, rep='pca', n_jobs=-1, n_components=2, perplexity=30, early_exaggeration=12, learning_rate=1000, random_state=0, out_basis='tsne')`
Calculate tSNE embedding using **MulticoreTSNE** package.

Parameters

- **data** (`anndata.AnnData`) – 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 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: 1000) – The learning rate can be a critical parameter, which should be between 100 and 1000.
- **random_state** (`int`, optional, default: 0) – Random seed set for reproducing results.

- **out_basis** (str, optional, default: "tsne") – Key name for calculated tSNE coordinates to store.

Return type None

Returns

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

Examples

```
>>> pg.tsne(adata)
```

pegasus.fitsne

`pegasus.fitsne` (*data*, *rep*='pca', *n_jobs*=- 1, *n_components*=2, *perplexity*=30, *early_exaggeration*=12, *learning_rate*=1000, *random_state*=0, *out_basis*='fitsne')

Calculate FI-t-SNE embedding using [fitsne](#) package.

Parameters

- **data** (`anndata.AnnData`) – 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: 1000) – The learning rate can be a critical parameter, which should be between 100 and 1000.
- **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.fitsne(adata)
```

pegasus.umap

```
pegasus.umap(data, rep='pca', n_components=2, n_neighbors=15, min_dist=0.5, spread=1.0, random_state=0, out_basis='umap')
```

Calculate UMAP embedding using [umap-learn](#) package.

Parameters

- **data** (`anndata.AnnData`) – 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.
- **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(adata)
```

pegasus.file

```
pegasus.file(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 using [ForceAtlas2](#) implementation, with Python wrapper as [forceatlas2-python](#).

Parameters

- **data** (`anndata.AnnData`) – 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 `hns` 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.file(adata)
```

pegasus.net_tsne

```
pegasus.net_tsne(data, rep='pca', n_jobs=-1, n_components=2, perplexity=30,
                  early_exaggeration=12, learning_rate=1000, random_state=0, select_frac=0.1,
                  select_K=25, select_alpha=1.0, net_alpha=0.1, polish_learning_frac=0.33, polish_n_iter=150, out_basis='net_tsne')
```

Calculate approximated tSNE 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.

Parameters

- **data** (`anndata.AnnData`) – Annotated data matrix with rows for cells (`n_obs`) and columns for genes (`n_feature`).
- **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 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: 1000) – The learning rate can be a critical parameter, which should be between 100 and 1000.
- **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_learning_frac** (`float`, optional, default: 0.33) – 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: 150) – Number of iterations for polishing tSNE run.

- **out_basis** (str, optional, default: "net_tsne") – Key name for the approximated tSNE coordinates calculated.

Return type None

Returns

- None
- Update `data.obsm` –
 - `data.obsm['X_' + out_basis]`: Net tSNE 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_tsne(adata)
```

pegasus.net_fitsne

```
pegasus.net_fitsne(data, rep='pca', n_jobs=-1, n_components=2, perplexity=30,  
                    early_exaggeration=12, learning_rate=1000, random_state=0, select_frac=0.1,  
                    select_K=25, select_alpha=1.0, net_alpha=0.1, polish_learning_frac=0.5,  
                    polish_n_iter=150, out_basis='net_fitsne')
```

Calculate approximated FI-tSNE 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.

Parameters

- **data** (`anndata.AnnData`) – Annotated data matrix with rows for cells (`n_obs`) and columns for genes (`n_feature`).
- **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 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: 1000) – The learning rate can be a critical parameter, which should be between 100 and 1000.
- **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_learning_frac** (float, optional, default: 0.5) – 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: 150) – Number of iterations for polishing FI-tSNE run.
- **out_basis** (str, optional, default: "net_fitsne") – Key name for the approximated FI-tSNE coordinates calculated.

Return type None

Returns

- None
- Update `data.obsm` –
– `data.obsm['X_' + out_basis]`: Net FI-tSNE 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_fitsne(adata)
```

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 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.

Parameters

- **data** (`anndata.AnnData`) – 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 $\text{radius} ** \text{select_alpha}$.
- **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 $\text{polish_learning_frac} * \text{n_obs}$ 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(adata)
```


pegasus.net_file

```
pegasus.net_file(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_file')
```

Construct the approximated Force-directed (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.

Parameters

- **data** (`anndata.AnnData`) – 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_file"`) – 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(adata)
```

Differential Expression Analysis

<code>de_analysis(data, cluster[, condition, ...])</code>	Perform Differential Expression (DE) Analysis on data.
<code>markers(data[, head, de_key, sort_by, alpha])</code>	
type data AnnData	
<code>write_results_to_excel(results, output_file)</code>	Write results into Excel workbook.

pegasus.de_analysis

`pegasus.de_analysis(data, cluster, condition=None, subset=None, result_key='de_res', n_jobs=-1, auc=True, t=True, fisher=False, mwu=False, temp_folder=None, verbose=True)`

Perform Differential Expression (DE) Analysis on data.

Parameters

- **data** (`anndata.AnnData`) – Annotated 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`.
- **subset** (`str`, optional, default: `None`) – Perform DE analysis on only a subset of cluster IDs. Cluster ID subset is specified as "`clust_id1, clust_id2, ..., clust_idn`", where all IDs must exist in `data.obs[cluster]`.
- **result_key** (`str`, optional, default: `"de_res"`) – Key name of DE analysis result stored.
- **n_jobs** (`int`, optional, default: `-1`) – Number of threads to use. If `-1`, use all available threads.
- **auc** (`bool`, optional, default: `True`) – If `True`, calculate area under ROC (AUROC) and area under Precision-Recall (AUPR).

- **t** (bool, optional, default: True) – If True, calculate Welch's t test.
- **fisher** (bool, optional, default: False) – If True, calculate Fisher's exact test.
- **mwu** (bool, optional, default: False) – If True, calculate Mann-Whitney U 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[result_key]`: DE analysis result.

Examples

```
>>> pg.de_analysis(adata, cluster = 'spectral_leiden_labels')
```

subset: a comma-separated list of cluster labels. Then de will be performed only on these subsets.

pegasus.markers

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

Parameters

- **data** (`anndata.AnnData`) – Annotated 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`.
- **sort_by** (str, optional, default: "auROC,WAD_score") – Sort the resulting marker dictionary by auROC and WAD_score.
- **alpha** (float, optional, default: 0.05) – q-value threshold for getting valid DE genes. Only those with q-value of any test below alpha are significant, and thus considered as DE genes.

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

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

Examples

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

pegasus.write_results_to_excel

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

Write results into Excel workbook.

Parameters

- **results** (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`.

Examples

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

Marker Detection based on Gradient Boost Machine

<code>find_markers(data, label_attr[, de_key, ...])</code>	Find markers using gradient boosting method.
--	--

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')
```

22.5.2 Annotate clusters:

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

pegasus.infer_cell_types

`pegasus.infer_cell_types` (*data*, *markers*, *de_test*, *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** (`anndata.AnnData`) – AnnData object of count matrix and DE analysis results.
- **markers** (str or Dict) –
 - If str, it
 - * either refers to a JSON file containing legacy markers, or
 - * 'human_immune' for predefined pegasus markers on human immune cells;
 - * 'mouse_immune' for mouse immune cells;
 - * 'human_brain' for human brain cells;
 - * 'mouse_brain' for mouse brain cells.
 - If Dict, it refers to a Python dictionary describing the markers.
- **de_test** (str) – pegasus determines cell types using DE test results. This argument indicates which DE test result to use, can be either 't', 'fisher' or 'mwu'.
- **de_alpha** (float, optional, default: 0.05) – False discovery rate for controlling family-wide error.
- **de_key** (str, optional, default: "de_res") – The keyword in `data.varm` that stores DE analysis results.

- **threshold** (float, optional, default: 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 sorted by scores being values.

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

Examples

```
>>> cell_type_dict = pg.infer_cell_types(adata, markers = 'human_immune', de_test_
↳= 't')
```

pegasus.annotate

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

Add annotation to AnnData obj.

Parameters

- **data** (AnnData) – AnnData object.
- **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]) – Dictionary mapping from cluster id to cell type.

Returns

Return type None

Examples

```
>>> annotate_cluster.annotate(data, 'anno', 'spectral_louvain_labels', {'1': 'T_
↳cell', '2': 'B cell'})
```

22.5.3 Plotting

Interactive Plots

<code>embedding(adata, basis[, keys, cmap, ...])</code>	Generate an embedding plot.
<code>composition_plot(adata, by, condition[, ...])</code>	Generate a composition plot, which shows the percentage of observations from every condition within each cluster (by).
<code>variable_feature_plot(adata, **kwds)</code>	Generate a variable feature plot.
<code>heatmap(adata, keys, by[, reduce_function, ...])</code>	Generate a heatmap.

continues on next page

Table 11 – continued from previous page

<code>dotplot(adata, keys, by[, reduce_function, ...])</code>	Generate a dot plot.
---	----------------------

pegasus.embedding

`pegasus.embedding` (*adata*, *basis*, *keys=None*, *cmap=None*, *palette=None*, *alpha=1*, *size=None*, *width=400*, *height=400*, *sort=True*, *cols=None*, *use_raw=None*, *nbins=-1*, *reduce_function=<function amax>*, *brush_categorical=False*, *legend='right'*, *tooltips=None*, *legend_font_size=None*, *opacity_min=0*, *opacity_max=1*, ***kwargs*)

Generate an embedding plot.

Parameters

- **adata** (AnnData) – Annotated data matrix.
- **keys** (Union[None, str, List[str], Tuple[str]]) – Key for accessing variables of `adata.var_names` or a field of `adata.obs` used to color the plot. Can also use *count* to plot cell count when binning.
- **basis** (Union[str, List[str], Tuple[str]]) – String in `adata.obsm` containing coordinates.
- **alpha** (float) – Points alpha value.
- **size** (Optional[float]) – Point pixel size.
- **sort** (bool) – Plot higher values on top of lower values. Disable for linked brushing.
- **brush_categorical** (bool) – Enable linked brushing on categorical variables (disables categorical legend).
- **cmap** (Union[str, List[str], Tuple[str], None]) – Color map for continuous variables.
- **palette** (Union[str, List[str], Tuple[str], None]) – Color map for categorical variables.
- **nbins** (int) – Number of bins used to summarize plot on a grid. Useful for large datasets. Negative one means automatically bin the plot.
- **reduce_function** (Callable[[array], float]) – Function used to summarize overlapping cells if *nbins* is specified.
- **cols** (Optional[int]) – Number of columns for laying out multiple plots
- **width** (int) – Plot width.
- **height** (int) – Plot height.
- **tooltips** (Union[str, List[str], Tuple[str], None]) – List of additional fields to show on hover.
- **legend** (str) – ‘top’, ‘bottom’, ‘left’, ‘right’, or ‘data’ to draw labels for categorical features on the plot.
- **legend_font_size** (Union[int, str, None]) – Font size for *labels_on_data*
- **use_raw** (Optional[bool]) – Use *raw* attribute of *adata* if present.
- **opacity_min** (float) – Minimum value for encoding categorical data purity when binning using opacity.

- **opacity_max** (float) – Maximum value for encoding categorical data purity when binning using opacity.

Return type Element

pegasus.composition_plot

`pegasus.composition_plot` (*adata*, *by*, *condition*, *stacked=True*, *normalize=True*, *condition_sort_by=None*, *cmap=None*, ***kws*)

Generate a composition plot, which shows the percentage of observations from every condition within each cluster (*by*).

Parameters

- **adata** (AnnData) – Annotated data matrix.
- **by** (str) – Key for accessing variables of `adata.var_names` or a field of `adata.obs` used to group the data.
- **condition** (str) – Key for accessing variables of `adata.var_names` or a field of `adata.obs` used to compute counts within a group.
- **stacked** (bool) – Whether bars are stacked.
- **normalize** (bool) – Normalize counts within each group to sum to one.
- **condition_sort_by** (Optional[str]) – Sort condition within each group by max, mean, natsorted, or None.
- **cmap** (Union[str, List[str], Tuple[str], None]) – Color map name (`hv.plotting.list_cmaps()`) or a list of hex colors. See http://holoviews.org/user_guide/Styling_Plots.html for more information.

Return type Element

pegasus.variable_feature_plot

`pegasus.variable_feature_plot` (*adata*, ***kws*)

Generate a variable feature plot.

Parameters **adata** (AnnData) – Annotated data matrix.

Return type Element

pegasus.heatmap

`pegasus.heatmap` (*adata*, *keys*, *by*, *reduce_function=<function mean>*, *use_raw=None*, *cmap='Reds'*, ***kws*)

Generate a heatmap.

Parameters

- **adata** (AnnData) – Annotated data matrix.
- **keys** (Union[str, List[str], Tuple[str]]) – Keys for accessing variables of `adata.var_names`
- **by** (str) – Group plot by specified observation.

- **cmap** (Union[str, List[str], Tuple[str]]) – Color map name (hv.plotting.list_cmaps()) or a list of hex colors. See http://holoviews.org/user_guide/Styling_Plots.html for more information.
- **reduce_function** (Callable[[ndarray], float]) – Function to summarize an element in the heatmap
- **use_raw** (Optional[bool]) – Use *raw* attribute of *adata* if present.

Return type Element

pegasus.dotplot

`pegasus.dotplot(adata, keys, by, reduce_function=<function mean>, fraction_min=0, fraction_max=None, dot_min=1, dot_max=26, use_raw=None, cmap='Reds', sort_function=None, **kws)`

Generate a dot plot.

Parameters

- **adata** (AnnData) – Annotated data matrix.
- **keys** (Union[str, List[str], Tuple[str]]) – Keys for accessing variables of `adata.var_names`
- **by** (str) – Group plot by specified observation.
- **cmap** (Union[str, List[str], Tuple[str]]) – Color map name (hv.plotting.list_cmaps()) or a list of hex colors. See http://holoviews.org/user_guide/Styling_Plots.html for more information.
- **reduce_function** (Callable[[ndarray], float]) – Function to summarize an element in the heatmap
- **fraction_min** (float) – Minimum fraction expressed value.
- **fraction_max** (Optional[float]) – Maximum fraction expressed value.
- **dot_min** (int) – Minimum pixel dot size.
- **dot_max** (int) – Maximum pixel dot size.
- **use_raw** (Optional[bool]) – Use *raw* attribute of *adata* if present.
- **sort_function** (Optional[Callable[[DataFrame], List[str]]]) – Optional function that accepts summarized data frame and returns a list of row indices in the order to render in the heatmap.

Return type Element

Quality Control Plots

<code>violin(adata, keys[, by, width, cmap, cols, ...])</code>	Generate a violin plot.
<code>scatter(adata, x, y[, color, size, dot_min, ...])</code>	Generate a scatter plot.
<code>scatter_matrix(adata, keys[, color, use_raw])</code>	Generate a scatter plot matrix.

pegasus.violin

`pegasus.violin` (*adata*, *keys*, *by=None*, *width=300*, *cmap=None*, *cols=None*, *use_raw=None*, ***kws*)
Generate a violin plot.

Parameters

- **adata** (AnnData) – Annotated data matrix.
- **keys** (Union[str, List[str], Tuple[str]]) – Keys for accessing variables of `adata.var_names`, field of `adata.var`, or field of `adata.obs`
- **by** (Optional[str]) – Group plot by specified observation.
- **width** (int) – Plot width.
- **cmap** (Union[str, List[str], Tuple[str], None]) – Color map name (`hv.plotting.list_cmaps()`) or a list of hex colors. See http://holoviews.org/user_guide/Styling_Plots.html for more information.
- **cols** (Optional[int]) – Number of columns for laying out multiple plots
- **use_raw** (Optional[bool]) – Use *raw* attribute of *adata* if present.

Return type Element

pegasus.scatter

`pegasus.scatter` (*adata*, *x*, *y*, *color=None*, *size=None*, *dot_min=2*, *dot_max=14*, *use_raw=None*, *sort=True*, *width=400*, *height=400*, *nbins=-1*, *reduce_function=<function amax>*, *cmap=None*, *palette=None*, ***kws*)
Generate a scatter plot.

Parameters

- **adata** (AnnData) – Annotated data matrix.
- **x** (str) – Key for accessing variables of `adata.var_names`, field of `adata.var`, or field of `adata.obs`
- **y** (str) – Key for accessing variables of `adata.var_names`, field of `adata.var`, or field of `adata.obs`
- **cmap** (Union[str, List[str], Tuple[str], None]) – Color map for continuous variables.
- **palette** (Union[str, List[str], Tuple[str], None]) – Color map for categorical variables.
- **color** (Optional[str]) – Field in `.var_names`, `adata.var`, or `adata.obs` to color the points by.
- **sort** (bool) – Plot higher color by values on top of lower values.
- **width** (int) – Chart width.
- **height** (int) – Chart height.
- **size** (Union[int, str, None]) – Field in `.var_names`, `adata.var`, or `adata.obs` to size the points by or a pixel size.
- **dot_min** – Minimum dot size when sizing points by a field.
- **dot_max** – Maximum dot size when sizing points by a field.

- **use_raw** (Optional[bool]) – Use *raw* attribute of *adata* if present.
- **nbins** (int) – Number of bins used to summarize plot on a grid. Useful for large datasets. Negative one means automatically bin the plot.
- **reduce_function** (Callable[[array], float]) – Function used to summarize overlapping cells if nbins is specified

Return type Element

pegasus.scatter_matrix

`pegasus.scatter_matrix(adata, keys, color=None, use_raw=None, **kwargs)`

Generate a scatter plot matrix.

Parameters

- **adata** (AnnData) – Annotated data matrix.
- **keys** (Union[str, List[str], Tuple[str]]) – Key for accessing variables of `adata.var_names` or a field of `adata.obs`
- **color** – Key in `adata.obs` to color points by.
- **use_raw** (Optional[bool]) – Use *raw* attribute of *adata* if present.

Return type Element

22.5.4 Demultiplexing

<code>estimate_background_probs(adf[, ran-</code>		For cell-hashing data, estimate antibody background probability using EM algorithm.
<code>dom_state])</code>		
<code>demultiplex(data, adf[, min_signal, alpha, ...])</code>		Demultiplexing cell-hashing data, using the estimated antibody background probability calculated in <code>pg.estimate_background_probs</code> .

pegasus.estimate_background_probs

`pegasus.estimate_background_probs(adf, random_state=0)`

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

Parameters

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

Returns

- None
- Update `adf.uns` –
 - `adf.uns["background_probs"]`: estimated antibody background probability.

Example

```
>>> pg.estimate_background_probs(adf)
```

pegasus.demultiplex

`pegasus.demultiplex(data, adf, min_signal=10.0, alpha=0.0, alpha_noise=1.0, tol=1e-06, n_threads=1)`

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

Parameters

- **data** (`anndata.AnnData`) – Annotated data matrix for gene expression matrix.
- **adf** (`anndata.AnnData`) – Annotated data matrix for antibody 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) –
- **tol** (float, optional, default: 1e-6) – Tolerance threshold when judging equivalence of two floating point values.
- **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 array, each feature barcode is assigned a unique sample name.

Examples

```
>>> pg.demultiplex(adata, adf)
```

22.5.5 Miscellaneous

<code>calc_signature_score(data, n_bins)</code>	<code>signatures[,</code>	Calculate signature / gene module score.
<code>search_genes(data, gene_list[, rec_key, measure])</code>		Extract and display gene expressions for each cluster from an <i>anndata</i> object.
<code>search_de_genes(data, gene_list[, rec_key, ...])</code>		Extract and display differential expression analysis results of markers for each cluster.

pegasus.calc_signature_score

`pegasus.calc_signature_score(data, signatures, n_bins=50)`

Calculate signature / gene module score.

This is an improved version of Livnat et al. 2018 Cell implementation.

Parameters

- **data** (`anndata.AnnData`) – Annotated data matrix with rows for cells and columns for genes.
- **signatures** (`Dict[str, List[str]]`) – A dictionary containing multiple signature score calculation requests. Each score will be stored in `data.obs` field with key as the keyword. The value of the dict is a list of gene symbols.
- **n_bins** (`int`, optional, default: 50) –

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"]})
↪
```

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 \geq threshold;
- + indicates up-regulated but fold change $<$ threshold;
- -- indicates down-regulated and fold change $\leq 1 / \text{threshold}$;
- - indicates down-regulated but fold change $> 1 / \text{threshold}$;
- ? 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)
```

22.6 References

22.7 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.

22.8 Contact us

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

BIBLIOGRAPHY

- [Büttner18] M. Büttner, et al., “A test metric for assessing single-cell RNA-seq batch correction”, In [Nature Method](#), 2018.
- [Malkov16] Yu. A. Malkov, and D.A. Yashunin, “Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs”, In [arXiv](#), 2016.

A

`aggregate_matrices()` (in module *pegasus*), 76
`annotate()` (in module *pegasus*), 106

C

`calc_kBET()` (in module *pegasus*), 84
`calc_kSIM()` (in module *pegasus*), 85
`calc_pseudotime()` (in module *pegasus*), 87
`calc_signature_score()` (in module *pegasus*), 113
`cluster()` (in module *pegasus*), 89
`composition_plot()` (in module *pegasus*), 108
`correct_batch()` (in module *pegasus*), 82

D

`de_analysis()` (in module *pegasus*), 102
`demultiplex()` (in module *pegasus*), 112
`diffmap()` (in module *pegasus*), 86
`dotplot()` (in module *pegasus*), 109

E

`embedding()` (in module *pegasus*), 107
`estimate_background_probs()` (in module *pegasus*), 111

F

`filter_data()` (in module *pegasus*), 78
`find_markers()` (in module *pegasus*), 104
`fitsne()` (in module *pegasus*), 94
`fle()` (in module *pegasus*), 96

G

`get_filter_stats()` (in module *pegasus*), 78

H

`heatmap()` (in module *pegasus*), 108
`highly_variable_features()` (in module *pegasus*), 79

I

`infer_cell_types()` (in module *pegasus*), 105

`infer_path()` (in module *pegasus*), 88

L

`leiden()` (in module *pegasus*), 90
`log_norm()` (in module *pegasus*), 79
`louvain()` (in module *pegasus*), 90

M

`markers()` (in module *pegasus*), 103

N

`neighbors()` (in module *pegasus*), 84
`net_fitsne()` (in module *pegasus*), 98
`net_fle()` (in module *pegasus*), 101
`net_tsne()` (in module *pegasus*), 97
`net_umap()` (in module *pegasus*), 99

P

`pca()` (in module *pegasus*), 80

Q

`qc_metrics()` (in module *pegasus*), 77

R

`read_input()` (in module *pegasus*), 74
`reduce_diffmap_to_3d()` (in module *pegasus*), 87
`run_harmony()` (in module *pegasus*), 83

S

`scatter()` (in module *pegasus*), 110
`scatter_matrix()` (in module *pegasus*), 111
`search_de_genes()` (in module *pegasus*), 114
`search_genes()` (in module *pegasus*), 114
`select_features()` (in module *pegasus*), 80
`set_group_attribute()` (in module *pegasus*), 81
`spectral_leiden()` (in module *pegasus*), 92
`spectral_louvain()` (in module *pegasus*), 91

T

`tsne()` (in module *pegasus*), 93

U

`umap()` (*in module pegasus*), [95](#)

V

`variable_feature_plot()` (*in module pegasus*),
[108](#)

`violin()` (*in module pegasus*), [110](#)

W

`write_output()` (*in module pegasus*), [75](#)

`write_results_to_excel()` (*in module pegasus*),
[104](#)