Pegasus Documentation

Release 1.4.0

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

Jun 25, 2021

CONTENTS

1 Release Highlights in Current Stable 3

Bibliography 95

Index 97

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

Read documentation

CHAPTER

ONE

RELEASE HIGHLIGHTS IN CURRENT STABLE

1.1 1.4.0 June 24, 2021

- Add `nmf` and `integrative_nmf` functions to compute NMF and iNMF using nmf-torch package; `integrative_nmf` supports quantile normalization proposed in the *LIGER* papers ([Welch19], [Gao21]).
- Change the parameter defaults of function `qc_metrics`: Now all defaults are `None`, meaning not performing any filtration on cell barcodes.
- In **Annotate Clusters** API functions:
  - Improve human immune cell markers and auto cell type assignment for human immune cells. (`infer_cell_types` function)
  - Update mouse brain cell markers (`infer_cell_types` function)
  - `annotate` function now adds annotation as a categorical variable and sort categories in natural order.
- Add `find_outlier_clusters` function to detect if any cluster is an outlier regarding one of the qc attributes (n_genes, n_counts, percent_mito) using MWU test.
- In **Plotting** API functions:
  - `scatter` function now plots all cells if `attrs` == `None`; Add `fix_corners` option to fix the four corners when only a subset of cells is shown.
  - Fix a bug in `heatmap` plotting function.
- Fix bugs in functions `spectral_leiden` and `spectral_louvain`.
- Improvements:
  - Support *umap-learn* v0.5+. (`umap` and `net_umap` functions)
  - Update doublet detection algorithm. (`infer_doublets` function)
  - Improve message reporting execution time spent each step. (Pegasus command line tool)

1.1.1 Installation

Pegasus works with Python 3.7, 3.8 and 3.9.

Linux

Ubuntu/Debian

Prerequisites

On Ubuntu/Debian Linux, first install the following dependency by:

```
sudo apt install build-essential
```

Next, you can install Pegasus system-wide by PyPI (see Ubuntu/Debian install via PyPI), or within a Miniconda environment (see *Install via Conda*).

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
```

Ubuntu/Debian install via PyPI

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

```
sudo apt install python3 python3-pip
python3 -m pip install --upgrade pip
```

Now install Pegasus via *pip*:

```
python3 -m pip install pegasuspy
```

There are optional packages that you can install:

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

  ```
  python3 -m pip install mkl
  ```

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

  ```
  sudo apt install libfftw3-dev
  python3 -m pip install fitsne
  ```

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

  ```
  python3 -m pip install leidenalg
  ```

Note: If installing from Python 3.9, you'll need to install the following packages system-wide first in order to locally compile `louvain` package, one of Pegasus' dependencies:

```
sudo apt install flex bison libtool
```

Fedora

Prerequisites

On Fedora Linux, first install the following dependency by:

```
sudo dnf install gcc gcc-c++
```

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

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 Fedora official repository (e.g. `java-latest-openjdk`):

```
sudo dnf install java-latest-openjdk
```

or other OpenJDK version chosen from the searching result of command:

```
dnf search openjdk
```

Fedora install via PyPI

Fedora 33+ has set Python 3.9 as the default Python 3 version. However, Pegasus has not supported it yet. So we'll use Python 3.8 in this tutorial.

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

```
sudo dnf install python3.8
python3.8 -m ensurepip --user
python3.8 -m pip install --upgrade pip
```

Now install Pegasus via *pip*:

```
python3.8 -m pip install pegasuspy
```

There are optional packages that you can install:

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

  ```
  python3.8 -m pip install mkl
  ```

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

  ```
  sudo dnf install fftw-devel
  python3.8 -m pip install fitsne
  ```

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

```
python3.8 -m pip install leidenalg
```

Note: If installing from Python 3.9, you'll need to install the following packages system-wide first in order to locally compile `louvain` package, one of Pegasus' dependencies:

```
sudo dnf install flex bison libtool
```

macOS

Prerequisites

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

```
brew install 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 *Install via Conda*).

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 install java
```

macOS install via PyPI

1. You need to install Python and *pip* tool first:

   ```
   brew install python3
   python3 -m pip install --upgrade pip
   ```

2. Now install Pegasus:

   ```
   python3 -m pip install pegasuspy
   ```

There are optional packages that you can install:

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

  ```
  python3 -m pip 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:

```
python3 -m pip install fitsne
```

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

```
python3 -m pip install leidenalg
```

Install via Conda

Alternatively, you can install Pegasus via Conda, which is a separate virtual environment without touching your system-wide packages and settings.

You can install Anaconda, or Miniconda (a minimal installer of conda). In this tutorial, we'll use Miniconda.

1. Download Miniconda installer for your OS. For example, if on 64-bit Linux, then use the following commands to install Miniconda:

```
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
```

where `/home/foo` should be replaced by the directory to which you want to install Miniconda. Similarly for macOS.

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

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

Also notice that Python `3.8` is used in this tutorial. To choose a different version of Python, simply change the version number in the command above. Since Pegasus conda package only support Python 3.7 and 3.8 for now, you should choose your Python version from either of these two.

3. Enter `pegasus` environment by activating:

```
conda activate pegasus
```

4. Install the following dependency:

```
conda install -y -c conda-forge louvain
```

6. Install Pegasus:

```
pip install pegasuspy
```

7. (Optional) If you want to use the FIt-SNE plot functionality in Pegasus, do the following:

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

And use the following command to enable the Leiden clustering functionality:

```
pip install leidenalg
```

Install via Singularity

Singularity is a container engine similar to Docker. Its main difference from Docker is that Singularity can be used with unprivileged permissions.

Note: Please notice that Singularity Hub has been offline since April 26th, 2021 (see blog post). All existing containers held there are in archive, and we can no longer push new builds.

So if you fetch the container from Singularity Hub using the following command:

```
singularity pull shub://klarman-cell-observatory/pegasus
```

it will just give you a Singularity container of Pegasus v1.2.0 running on Ubuntu Linux 20.04 base with Python 3.8, in the name `pegasus_latest.sif` of about 2.4 GB.

On your local machine, first install Singularity, then you can use our Singularity spec file to build a Singularity container by yourself.

Say the built container file has name `pegasus.sif`. Now you can interact with it, e.g.:

```
singularity run pegasus.sif
```

Please refer to Singularity image interaction guide for details.

Development Version

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

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

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

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

1.1.2 Use Pegasus as a command line tool

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

```
pegasus -h
```

to see the help information:

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

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

- Preprocessing:

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

- Demultiplexing:

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

- Analyzing:

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

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

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

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

- Plotting:

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

- Web-based visualization:

  scp_output Generate output files for single cell portal.

- MISC:

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

Quick guide

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

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

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

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

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

`pegasus aggregate_matrix`

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

Type:

```
pegasus aggregate_matrix -h
```

to see the usage information:

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

- Arguments:

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

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

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

output_name The output file name.

- Options:

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

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

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

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

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

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

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

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

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

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

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

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

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

-h, --help Print out help information.

- Outputs:

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

- Examples:

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

---

pegasus demuxEM

Demultiplex cell-hashing/nucleus-hashing data.

Type:

```
pegasus demuxEM -h
```

to see the usage information:

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

- Arguments:

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

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

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

- Options:

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

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

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

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

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

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

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

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

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

-v, --version Show DemuxEM version.

-h, --help Print out help information.

- Outputs:

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

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

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

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

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

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

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

- Examples:

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

pegasus cluster

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

Type:

```
pegasus cluster -h
```

to see the usage information:

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

- Arguments:

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

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

- Options:

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

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

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

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

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

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

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

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

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

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

  --output-loom Output loom-formatted file.

--output-h5ad Output h5ad-formatted file.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

--plot-hvf Plot highly variable feature selection.

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

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

--batch-group-by <expression> Batch correction assumes the differences in gene expression between channels are due to batch effects. However, in many cases, we know that channels can be partitioned into several groups and each group is biologically different from others. In this case, we will only perform batch correction for channels within each group. This option defines the groups. If <expression> is None, we assume all channels are from one group. Otherwise, groups are defined according to <expression>. <expression> takes the form of either 'attr', or 'attr1+attr2+...+attrn', or 'attr=value11,...,value1n_1;value21,...,value2n_2;...;valuem1,...,valuemn_m'. In the first form, 'attr' should be an existing sample attribute, and groups are defined by 'attr'. In the second form, 'attr1',...,'attrn' are n existing sample attributes and groups are defined by the Cartesian product of these n attributes. In the last form, there will be m + 1 groups. A cell belongs to group

i (i > 0) if and only if its sample attribute 'attr' has a value among valuei1,...,valuein_i. A cell belongs to group 0 if it does not belong to any other groups.

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

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

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

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

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

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

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

--kBET Calculate kBET.

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

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

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

--diffmap Calculate diffusion maps.

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

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

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

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

--louvain Run louvain clustering algorithm.

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

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

--leiden Run leiden clustering algorithm.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

--umap Run umap for visualization.

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

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

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

--fle Run force-directed layout embedding.

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

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

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

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

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

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

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

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

--net-umap Run net umap for visualization.

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

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

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

--net-fle Run net FLE.

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

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

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

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

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

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

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

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

-h, --help Print out help information.

- Outputs:

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

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

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

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

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

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

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

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

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

- Examples:

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

pegasus de_analysis

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

Type:

```
pegasus de_analysis -h
```

to see the usage information:

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

- Arguments:

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

  output_spreadsheet Output spreadsheet with DE results.

- Options:

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

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

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

  --t Calculate Welch's t-test.

  --fisher Calculate Fisher's exact test.

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

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

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

  --quiet Do not show detailed intermediate outputs.

  -h, --help Print out help information.

- Outputs:

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

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

- Examples:

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

pegasus find_markers

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

Type:

```
pegasus find_markers -h
```

to see the usage information:

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

- Arguments:

  input_h5ad_file Single cell data after running the de_analysis.

  output_spreadsheet Output spreadsheet with LightGBM detected markers.

- Options:

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

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

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

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

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

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

  -h, --help Print out help information.

- Outputs:

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

- Examples:

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

---

pegasus annotate_cluster

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

Type:

```
pegasus annotate_cluster -h
```

to see the usage information:

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

- Arguments:

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

  output_file Output annotation file.

- Options:

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

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

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

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

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

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

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

  -h, --help Print out help information.

- Outputs:

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

- Examples:

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

pegasus plot

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

Type:

```
pegasus plot -h
```

to see the usage information:

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

- Arguments:

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

  input_file Single cell data in Zarr or H5ad format.

  output_file Output image file.

- Options:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

-h, --help Print out help information.

Examples:

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

---

pegasus scp_output

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

Type:

```
pegasus scp_output -h
```

to see the usage information:

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

- Arguments:

  input_data_file Analyzed single cell data in zarr format.

  output_name Name prefix for all outputted files.

- Options:

  --dense Output dense expression matrix instead.

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

  -h, --help Print out help information.

- Outputs:

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

- Examples:

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

---

pegasus check_indexes

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

Type:

```
pegasus check_indexes -h
```

to see the usage information:

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

- Arguments:

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

- Options:

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

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

  -h, --help Print out help information.

- Outputs:

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

- Examples:

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

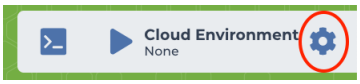
1.1.3 Use Pegasus on Terra Notebook

You need to first have a Terra account.

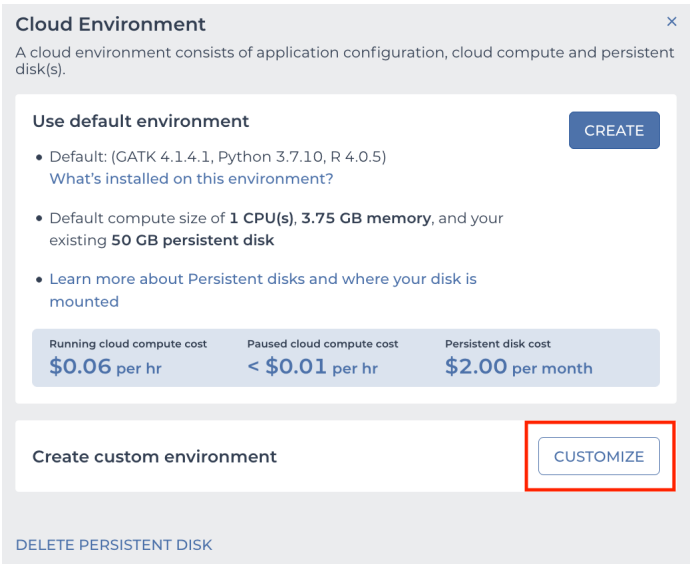
1. Start Notebook Runtime on Terra

The first time when you use Terra notebook, you need to create a Cloud Environment.

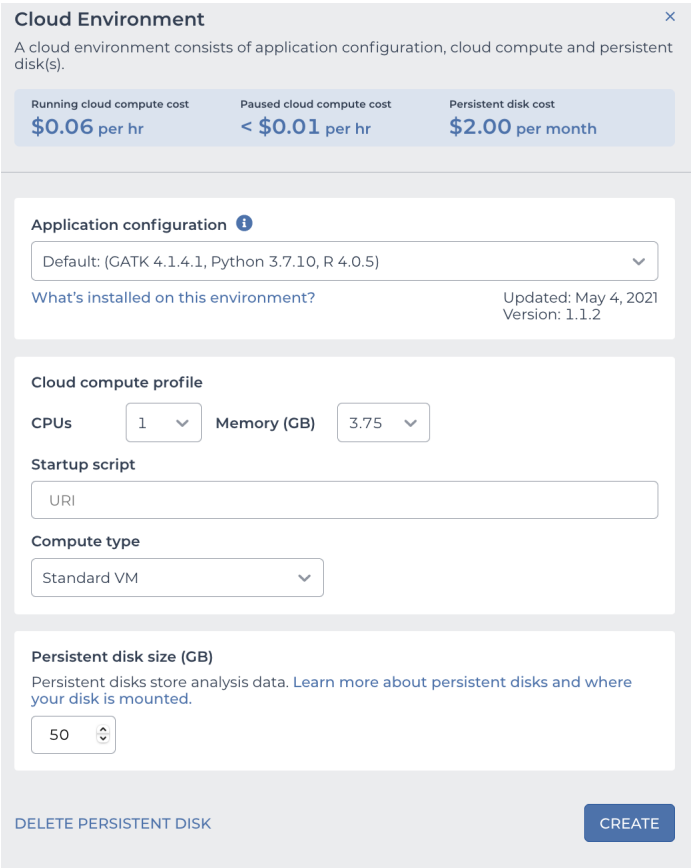
On the top-right panel of your workspace, click the following button within red circle:



Then click the CUSTOMIZE button shown in red regtangle:



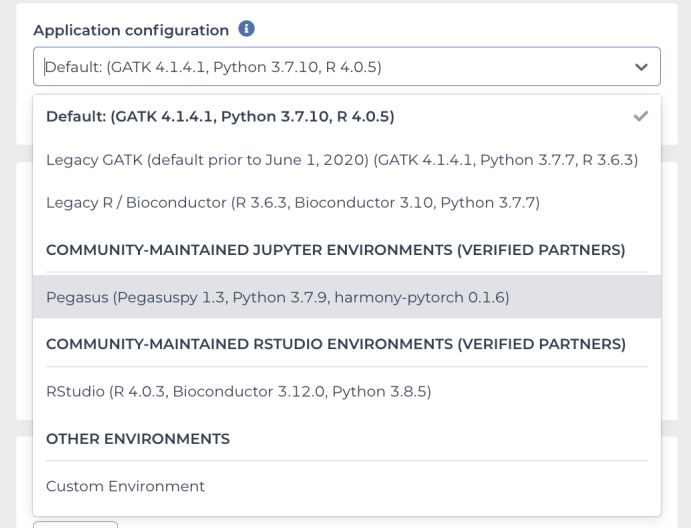
Then you'll need to set the configuration of your cloud environment in the pop-out dialog (see image below):



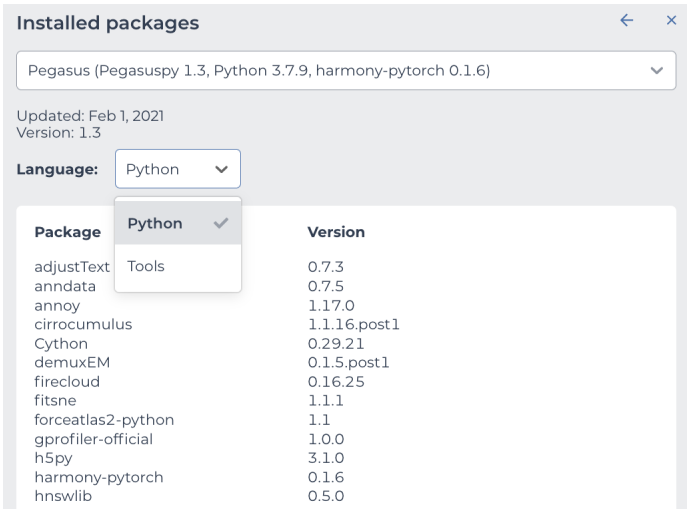
1.1. Create from Terra official environment

Terra team maintains a list of cloud environments for users to quickly set up their own. In this way, you'll use the most recent stable Pegasus cloud environment.

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



In case you are interested in looking at which packages and tools are included in the Pegasus cloud environment, please click the `What's installed on this environment?` link right below the drop-down menu:



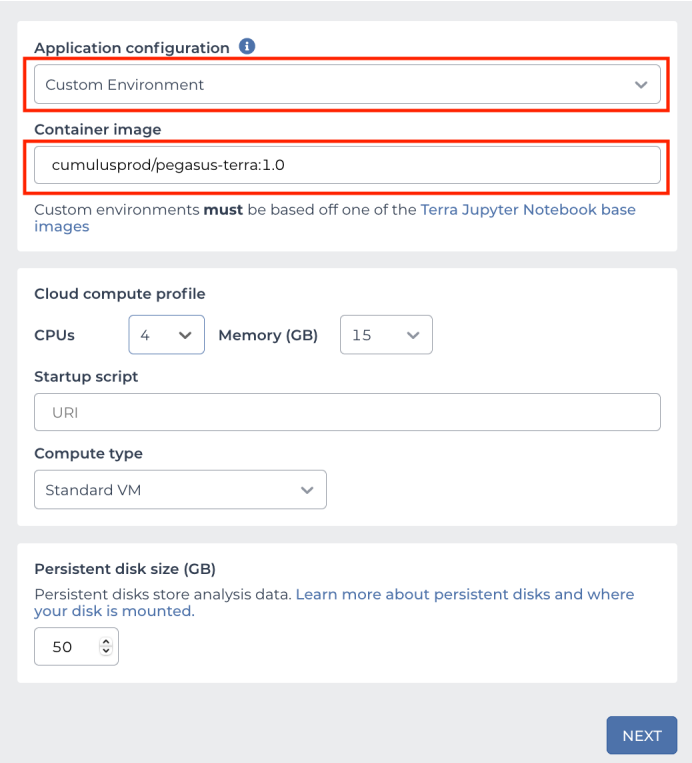
After that set other fields in the pop-out dialog:

- In *Cloud compute profile* field, you can set the computing resources (CPUs and Memory size) you want to use.
- In *Compute type* input, choose `Standard VM`, as this is the cheapest type and is enough for using Pegasus.
- In *Persistent disk size (GB)* field, choose the size of the persistent disk for your cloud environment. This disk space by default remains after you delete your cloud environment (unless you choose to delete the persistent disk as well), but it costs even if you stop your cloud environment.

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

1.2. Create from custom environment

Alternatively, you can create from a custom environment if you want to use an older version of Pegasus.



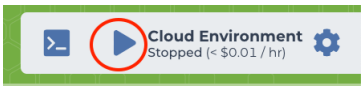
In the cloud environment setting page:

- In *Application configuration* field, choose `Custom Environment` (see the first red rectangle above).
- In *Container image* input, type `cumulusprod/pegasus-terra:<version>` (see the second red rectangle above), where `<version>` should be chosen from this list. All the tags are for different versions of Pegasus.
- In *Cloud compute profile* field, set the computing resources (CPUs and Memory size) you want to use.
- In *Compute type* field, choose `Standard VM`, as this is the cheapest type and is enough for using Pegasus.
- In *Persistent disk size (GB)* field, choose the size of the persistent disk for your cloud environment.

When finishing the configuration, click `NEXT` button. You'll see a warning page, then click `CREATE` button to start the creation. After waiting for 1-2 minutes, your cloud environment will be ready to use, and it's started automatically.

1.3. Start an environment already created

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



2. Create Your Terra Notebook

In the **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 cloud environment.

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

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

3. Load Data into Cloud Environment

To use your data on Cloud (i.e. from the Google Bucket of your workspace), you should first copy it into your notebook's cloud environment by Google Cloud SDK:

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

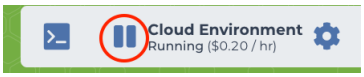
where `gs://link-to-count-matrix` is the Google Bucket URL to your count matrix data file, and `!` is the indicator of running terminal commands within Jupyter notebook.

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

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

4. Stop Notebook Runtime

When you are done with the interactive analysis, to avoid being charged by Google Cloud while not using it, don't forget to stop your cloud environment by clicking the following button of the top-right panel of your workspace within red circle:



If you forget to stop manually, as far as you've closed all the webpages related to your cloud environment (e.g. Terra notebooks, Terra terminals, etc.), you'll still be safe. In this case, Terra will automatically stop the cloud environment for you after waiting for a few minutes.

1.1.4 Tutorials

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

1.1.5 API

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

```
import pegasus as pg
```

Read and Write

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

pegasus.read_input

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

Load data into memory.

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

Parameters

- **input_file** (*str*) – Input file name.
- **file_type** (*str*, optional (default: None)) – File type, choosing from 'zarr', 'h5ad', 'loom', '10x', 'mtx', 'csv', 'tsv', 'fcs' (for flow/mass cytometry data) or 'nanostring'. If None, inferred from input_file.
- **mode** (*str*, optional (default: 'r')) – File open mode, options are 'r' or 'a'. If mode == 'a', file_type must be zarr and ngene/select_singlets cannot be set.
- **genome** (*str*, optional (default: None)) – For formats like loom, mtx, dge, csv and tsv, genome is used to provide genome name. In this case if genome is None, except mtx format, "unknown" is used as the genome name instead.
- **modality** (*str*, optional (default: None)) – Default modality, choosing from 'rna', 'atac', 'tcr', 'bcr', 'crispr', 'hashing', 'citeseq', 'cyto' (flow cytometry / mass cytometry) or 'nanostring'. If None, use 'rna' as default.
- **black_list** (*Set[str]*, optional (default: None)) – Attributes in black list will be poped out.
- **select_data** (*Set[str]*, optional (default: None)) – Only select data with keys in select_data. Select_data, select_genome and select_modality are mutually exclusive.
- **select_genome** (*Set[str]*, optional (default: None)) – Only select data with genomes in select_genome. Select_data, select_genome and select_modality are mutually exclusive.
- **select_modality** (*Set[str]*, optional (default: None)) – Only select data with modalities in select_modality. Select_data, select_genome and select_modality are mutually exclusive.

Returns

Return type A MultimodalData object.

Examples

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

pegasus.write_output

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

Write data back to disk.

This function is used to write data back to disk.

Parameters

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

Returns

Return type *None*

Examples

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

pegasus.aggregate_matrices

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

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

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

If csv_file is a dictionary, it can contains an alternative 2 columns - Sample, sample name; Object, Multimodal data object. In this case, the objects will be merged into one data object.

Parameters

- **csv_file** (*str*) – The CSV file containing information about each channel. Alternatively, a dictionary or pd.Dataframe can be passed.
- **restrictions** (*list[str]* or *str*, optional (default: [])) – A list of restrictions used to select channels, each restriction takes the format of name:value,...,value or name:~value,..,value, where ~ refers to not. If only one restriction is provided, it can be provided as a string instead of a list.
- **attributes** (*list[str]* or *str*, optional (default: [])) – A list of attributes need to be incorporated into the output count matrix. If only one attribute is provided, this attribute can be provided as a string instead of a list.
- **default_ref** (*str*, optional (default: None)) – Default reference name to use. If there is no Reference column in the csv_file, a Reference column will be added with default_ref as its value. This argument can also be used for replacing genome names. For example, if default_ref is 'hg19:GRCh38,GRCh38', we will change any genome with name 'hg19' to 'GRCh38' and if no genome is provided, 'GRCh38' is the default.
- **append_sample_name** (*bool*, optional (default: True)) – By default, append sample_name to each channel. Turn this option off if each channel has distinct barcodes.
- **select_singlets** (*bool*, optional (default: False)) – If we have demultiplexed data, turning on this option will make pegasus only include barcodes that are predicted as singlets.
- **remap_string** (`str`, optional, default `None`) – Remap singlet names using <remap_string>, where <remap_string> takes the format "new_name_i:old_name_1,old_name_2;new_name_ii:old_name_3;...". For example, if we hashed 5 libraries from 3 samples sample1_lib1, sample1_lib2, sample2_lib1, sample2_lib2 and sample3, we can remap them to 3 samples using this string: "sample1:sample1_lib1,sample1_lib2;sample2:sample2_lib1,sample2_lib2". In this way, the new singlet names will be in metadata field with key 'assignment', while the old names will be kept in metadata field with key 'assignment.orig'.
- **subset_string** (`str`, optional, default `None`) – If select singlets, only select singlets in the <subset_string>, which takes the format "name1,name2,...". Note that if –remap-singlets is specified, subsetting happens after remapping. For example, we can only select singlets from sampe 1 and 3 using "sample1,sample3".
- **min_genes** (`int`, optional, default: None) – Only keep cells with at least `min_genes` genes.
- **max_genes** (`int`, optional, default: None) – Only keep cells with less than `max_genes` genes.
- **min_umis** (`int`, optional, default: None) – Only keep cells with at least `min_umis` UMIs.
- **max_umis** (`int`, optional, default: None) – Only keep cells with less than `max_umis` UMIs.
- **mito_prefix** (`str`, optional, default: None) – Prefix for mitochondrial genes.
- **percent_mito** (`float`, optional, default: None) – Only keep cells with percent mitochondrial genes less than `percent_mito` % of total counts. Only when both mito_prefix and percent_mito set, the mitochondrial filter will be triggered.

Returns The aggregated count matrix as an MultimodalData object.

Return type *MultimodalData* object.

Examples

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

Analysis Tools

Preprocess

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

pegasus.qc_metrics

pegasus.**qc_metrics**(*data*, *select_singlets=False*, *remap_string=None*, *subset_string=None*, *min_genes=None*, *max_genes=None*, *min_umis=None*, *max_umis=None*, *mito_prefix=None*, *percent_mito=None*)

Generate Quality Control (QC) metrics regarding cell barcodes on the dataset.

Parameters

- **data** (`pegasusio.MultimodalData`) – Use current selected modality in data, which should contain one RNA expression matrix.
- **select_singlets** (`bool`, optional, default `False`) – If select only singlets.
- **remap_string** (`str`, optional, default `None`) – Remap singlet names using <remap_string>, where <remap_string> takes the format "new_name_i:old_name_1,old_name_2;new_name_ii:old_name_3;...". For example, if we hashed 5 libraries from 3 samples sample1_lib1, sample1_lib2, sample2_lib1, sample2_lib2 and sample3, we can remap them to 3 samples using this string: "sample1:sample1_lib1,sample1_lib2;sample2:sample2_lib1,sample2_lib2". In this way, the new singlet names will be in metadata field with key 'assignment', while the old names will be kept in metadata field with key 'assignment.orig'.

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

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.
  - demux_type: this column might be deleted if select_singlets is on.

Examples

```
>>> pg.qc_metrics(data, min_genes=500, max_genes=6000, mito_prefix="MT-", percent_
↪mito=10)
```

pegasus.get_filter_stats

pegasus.**get_filter_stats**(*data*, *min_genes_before_filt=100*)

Calculate filtration stats on cell barcodes.

Parameters

- **data** (pegasusio.MultimodalData) – Use current selected modality in data, which should contain one RNA expression matrix.
- **min_genes_before_filt** (int, optional, default 100) – If raw data matrix is input, empty barcodes will dominate pre-filtration statistics. To avoid this, for raw matrix, only consider barcodes with at least <number> genes for pre-filtration condition.

Returns **df_cells** – Data frame of stats on cell filtration.

Return type pandas.DataFrame

Examples

```
>>> df = pg.get_filter_stats(data)
```

pegasus.filter_data

pegasus.**filter_data**(*data*, *focus_list=None*)

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

Parameters

- **data** (`pegasusio.MultimodalData`) – Use current selected modality in data, which should contain one RNA expression matrix.
- **focus_list** (`List[str]`, optional, default None) – UnimodalData objects with keys in focus_list were qc_metrics marked. Filter them and make sure other modalities' barcodes are consistent with filtered barcodes. If focus_list is None and self._selected's modality is "rna", focus_list = [self._selected]

Return type `None`

Returns

- `None`
- Update `data` with cells after filtration.

Examples

```
>>> pg.filter_data(data)
```

pegasus.identify_robust_genes

pegasus.**identify_robust_genes**(*data*, *percent_cells=0.05*)

Identify robust genes as candidates for HVG selection and remove genes that are not expressed in any cells.

Parameters

- **data** (`pegasusio.MultimodalData`) – Use current selected modality in data, which should contain one RNA expression matrix.
- **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.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.identify_robust_genes(data, percent_cells = 0.05)
```

pegasus.log_norm

pegasus.**log_norm**(*data*, *norm_count=100000.0*, *backup_matrix='raw.X'*)

Normalization, and then apply natural logarithm to the data.

Parameters

- **data** (pegasusio.MultimodalData) – Use current selected modality in data, which should contain one RNA expression matrix.
- **norm_count** (int, optional, default: 1e5.) – Total counts of one cell after normalization.
- **backup_matrix** (str, optional, default: raw.X.) – The key name of the backup count matrix, usually the raw counts.

Return type None

Returns

- None
- Update data.X with count matrix after log-normalization. In addition, back up the original count matrix as backup_matrix.
- In case of rerunning normalization while backup_matrix already exists, use backup_matrix instead of data.X for normalization.

Examples

```
>>> pg.log_norm(data)
```

pegasus.highly_variable_features

pegasus.**highly_variable_features**(*data*, *consider_batch=False*, *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** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **consider_batch** (bool, optional, default: False) – 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 physical CPU cores will be used.

Return type None

Returns

- None
- Update data.var –
  - highly_variable_features: replace with Boolean type array indicating the selected highly variable features.

Examples

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

pegasus.select_features

pegasus.**select_features**(*data*, *features='highly_variable_features'*, *standardize=True*, *max_value=10.0*)

Subset the features and store the resulting matrix in dense format in data.uns with *'_tmp_fmat_'* prefix, with the option of standardization and truncating based on max_value. *'_tmp_fmat_*'* will be removed before writing out the disk. :type data: MultimodalData :param data: Annotated data matrix with rows for cells and columns for genes. :type data: pegasusio.MultimodalData :type features: str :param features: a keyword in data.var, which refers to a boolean array. If None, all features will be selected. :type features: str, optional, default: highly_variable_features. :type standardize: bool :param standardize: Whether to scale the data to unit variance and zero mean. :type standardize: bool, optional, default: True. :type max_value: float :param max_value: The threshold to truncate data after scaling. If None, do not truncate. :type max_value: float, optional, default: 10.

Return type str

Returns

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

Examples

```
>>> pg.select_features(data)
```

pegasus.pca

pegasus.**pca**(*data*, *n_components=50*, *features='highly_variable_features'*, *standardize=True*, *max_value=10.0*, *n_jobs=- 1*, *random_state=0*)

Perform Principle Component Analysis (PCA) to the data.

The calculation uses *scikit-learn* implementation.

Parameters

- **data** (pegasusio.MultimodalData) – 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.
- **max_value** (float, optional, default: 10.) – The threshold to truncate data after scaling. If None, do not truncate.
- **n_jobs** (*int*, optional (default: -1)) – Number of threads to use. -1 refers to using all physical CPU cores.
- **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.
  - data.uns["pca_features"]: Record the features used to generate PCA components.

Examples

```
>>> pg.pca(data)
```

pegasus.nmf

pegasus.**nmf**(*data*, *n_components=20*, *features='highly_variable_features'*, *space='log'*, *init='nndsvdar'*, *algo='halsvar'*, *mode='batch'*, *tol=0.0001*, *use_gpu=False*, *alpha_W=0.0*, *l1_ratio_W=0.0*, *alpha_H=0.0*, *l1_ratio_H=0.0*, *fp_precision='float'*, *n_jobs=- 1*, *random_state=0*)

Perform Nonnegative Matrix Factorization (NMF) to the data using Frobenius norm. Steps include select features and L2 normalization and NMF and L2 normalization of resulting coordinates.

The calculation uses nmf-torch package.

Parameters

- **data** (pegasusio.MultimodalData) – 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 nmf.
- **max_value** (float, optional, default: None.) – The threshold to truncate data symmetrically after scaling. If None, do not truncate.
- **space** (str, optional, default: log.) – Choose from log and expression. log works on log-transformed expression space; expression works on the original expression space (normalized by total UMIs).
- **init** (str, optional, default: nndsvdar.) – Method to initialize NMF. Options are 'random', 'nndsvd', 'nndsvda' and 'nndsvdar'.
- **algo** (str, optional, default: halsvar) – Choose from mu (Multiplicative Update), hals (Hierarchical Alternative Least Square), halsvar (HALS variant, use HALS to mimic bpp and can get better convergence for sometimes) and bpp (alternative non-negative least squares with Block Principal Pivoting method).
- **mode** (str, optional, default: batch) – Learning mode. Choose from batch and online. Notice that online only works when beta=2.0. For other beta loss, it switches back to batch method.
- **tol** (float, optional, default: 1e-4) – The toleration used for convergence check.
- **use_gpu** (bool, optional, default: False) – If True, use GPU if available. Otherwise, use CPU only.
- **alpha_W** (float, optional, default: 0.0) – A numeric scale factor which multiplies the regularization terms related to W. If zero or negative, no regularization regarding W is considered.
- **l1_ratio_W** (float, optional, default: 0.0) – The ratio of L1 penalty on W, must be between 0 and 1. And thus the ratio of L2 penalty on W is (1 - l1_ratio_W).
- **alpha_H** (float, optional, default: 0.0) – A numeric scale factor which multiplies the regularization terms related to H. If zero or negative, no regularization regarding H is considered.
- **l1_ratio_H** (float, optional, default: 0.0) – The ratio of L1 penalty on W, must be between 0 and 1. And thus the ratio of L2 penalty on H is (1 - l1_ratio_H).

- **fp_precision** (str, optional, default: float) – The numeric precision on the results. Choose from float and double.
- **n_jobs** (*int*, optional (default: -1)) – Number of threads to use. -1 refers to using all physical CPU cores.
- **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_nmf"]: Scaled NMF coordinates. Each column has a unit variance.
- Update data.uns –
  - data.uns["W"]: The feature factor matrix.
  - data.uns["H"]: The coordinate factor matrix.
  - data.uns["nmf_err"]: The NMF loss.
  - data.uns["nmf_features"]: Record the features used to perform NMF analysis.

Examples

```
>>> pg.nmf(data)
```

pegasus.regress_out

pegasus.**regress_out**(*data*, *attrs*, *rep='pca'*)

Regress out effects due to specific observational attributes.

Parameters

- **data** (MultimodalData or UnimodalData object) – Annotated data matrix with rows for cells and columns for genes.
- **attrs** (List[str]) – List of numeric cell attributes to be regressed out. They must exist in data.obs field.
- **rep** (str, optional, default: pca) – This is to specify which embedding to be used for regressing out. The key 'X_'+rep must exist in data.obsm field. By default, use PCA embedding.

Return type str

Returns

- **res_key** (str) – The key to the resulting new embedding matrix in data.obsm. It's 'X_'+rep+'_regressed'.
- Update data.obsm –
  - data.obsm[pca_key]: The PCA matrix with effects of attributes specified regressed out.

Examples

```
>>> pg.regress_out(data, attrs=['G1/S', 'G2/M'])
```

Batch Correction

set_group_attribute(data, attribute_string)	Set group attributes used in batch correction.
correct_batch(data[, features])	Batch correction on data using Location-Scale (L/S) Adjustment method.
run_harmony(data[, batch, rep, n_jobs, ...])	Batch correction on PCs using Harmony.
run_scanorama(data[, batch, n_components, ...])	Batch correction using Scanorama.
integrative_nmf(data[, batch, n_components, ...])	Perform Integrative Nonnegative Matrix Factorization (iNMF) for data integration.

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** (pegasusio.MultimodalData) – 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_m** 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(data, attr_string = "Individual")
```

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

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

pegasus.correct_batch

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

Batch correction on data using Location-Scale (L/S) Adjustment method. ([Li-and-Wong03], [Li20]). If L/S adjustment method is used, users must call this function every time before they call the pca function.

Parameters

- **data** (pegasusio.MultimodalData) – 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(data, features = "highly_variable_features")
```

pegasus.run_harmony

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

Batch correction on PCs using Harmony.

This is a wrapper of harmony-pytorch package, which is a Pytorch implementation of Harmony algorithm [Korsunsky19].

Parameters

- **data** (MultimodalData.) – Annotated data matrix with rows for cells and columns for genes.
- **batch** (str, optional, default: "Channel".) – Which attribute in data.obs field represents batches, default is "Channel".
- **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 in Harmony. -1 refers to using all physical CPU cores.

- **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(data, rep = "pca", n_jobs = 10, random_state = 25)
```

pegasus.run_scanorama

pegasus.**run_scanorama**(*data*, *batch='Channel'*, *n_components=50*, *features='highly_variable_features'*, *standardize=True*, *max_value=10.0*, *random_state=0*)

Batch correction using Scanorama.

This is a wrapper of Scanorama package. See [Hie19] for details on the algorithm.

Parameters

- **data** (MultimodalData.) – Annotated data matrix with rows for cells and columns for genes.
- **batch** (str, optional, default: "Channel".) – Which attribute in data.obs field represents batches, default is "Channel".
- **n_components** (int, optional default: 50.) – Number of integrated embedding components to keep. This sets Scanorama's dimred parameter.
- **features** (str, optional, default: "highly_variable_features".) – Keyword in data.var to specify features used for Scanorama.
- **standardize** (bool, optional, default: True.) – Whether to scale the data to unit variance and zero mean.
- **max_value** (float, optional, default: 10.) – The threshold to truncate data after scaling. If None, do not truncate.
- **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 Scanorama algorithm. out_rep is always equal to "scanorama"
- Update data.obsm –

  – data.obsm['X_scanorama']: The embedding calculated by Scanorama algorithm.

Examples

```
>>> pg.run_scanorama(data, random_state = 25)
```

pegasus.integrative_nmf

pegasus.**integrative_nmf**(*data*, *batch='Channel'*, *n_components=20*, *features='highly_variable_features'*, *space='log'*, *algo='halsvar'*, *mode='online'*, *tol=0.0001*, *use_gpu=False*, *lam=5.0*, *fp_precision='float'*, *n_jobs=- 1*, *random_state=0*, *quantile_norm=True*)

Perform Integrative Nonnegative Matrix Factorization (iNMF) for data integration.

The calculation uses nmf-torch package.

This function assumes that cells in each batch are adjacent to each other. In addition, it will scale each batch with L2 norm separately. The resulting Hs will also be scaled with L2 norm. If `quantile_norm=True`, quantile normalization will be additionally performed.

See [Welch19] and [Gao21] for details on the algorithms.

Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **batch** (`str`, optional, default: `"Channel"`.) – Which attribute in data.obs field represents batches, default is "Channel".
- **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 integrative_nmf.
- **space** (`str`, optional, default: `log`.) – Choose from `log` and `expression`. `log` works on log-transformed expression space; `expression` works on the original expression space (normalized by total UMIs).
- **algo** (`str`, optional, default: `halsvar`) – Choose from `mu` (Multiplicative Update), `halsvar` (HALS variant that mimic bpp but faster) and `bpp` (alternative non-negative least squares with Block Principal Pivoting method).
- **mode** (`str`, optional, default: `online`) – Learning mode. Choose from `batch` and `online`. Notice that `online` only works when `beta=2.0`. For other beta loss, it switches back to `batch` method.
- **tol** (`float`, optional, default: `1e-4`) – The toleration used for convergence check.
- **use_gpu** (`bool`, optional, default: `False`) – If `True`, use GPU if available. Otherwise, use CPU only.
- **lam** (`float`, optional, default: `5.0`) – The coefficient for regularization terms. If `0`, then no regularization will be performed.
- **fp_precision** (`str`, optional, default: `float`) – The numeric precision on the results. Choose from `float` and `double`.
- **n_jobs** (*int*, optional (default: -1)) – Number of threads to use. -1 refers to using all physical CPU cores.
- **random_state** (`int`, optional, default: `0`.) – Random seed to be set for reproducing result.

- **quantile_norm** (bool, optioanl, default: True.) – Perform quantile normalization as described in Gao et al. Nature Biotech 2021. Cluster refinement K=20; min_cells=20; quantiles = 50.

Return type str

Returns

- None.
- Update data.obsm –
  - data.obsm["X_inmf"]: Scaled and possibly quantile normalized iNMF coordinates.
- Update data.uns –
  - data.uns["W"]: The feature factor matrix.
  - data.uns["Hs"]: The coordinate factor matrices.
  - data.uns["Vs"]: The batch specific feature factor matrices.
  - data.uns["inmf_err"]: The iNMF loss.
  - data.uns["inmf_features"]: Record the features used to perform iNMF analysis.

Examples

```
>>> pg.integrative_nmf(data)
```

Nearest Neighbors

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

pegasus.neighbors

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

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

The kNN calculation uses hnswlib introduced by [Malkov16].

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **K** (int, optional, default: 100) – Number of neighbors, including the data point itself.

- **rep** (str, optional, default: "pca") – Embedding representation used to calculate kNN. If None, use data.X; otherwise, keyword 'X_' + rep must exist in data.obsm.
- **n_jobs** (int, optional, default: -1) – Number of threads to use. If -1, use all physical CPU cores.
- **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.
- **use_cache** (bool, optional, default: True) –
  - If True and found cached knn results, Pegasus will use cached results and do not recompute.
  - Otherwise, compute kNN irrespective of caching status.

Return type None

Returns

- None
- Update data.uns –
  - data.uns[rep + "_knn_indices"]: kNN index matrix. Row i is the index list of kNN of cell i (excluding itself), sorted from nearest to farthest.
  - data.uns[rep + "_knn_distances"]: kNN distance matrix. Row i is the distance list of kNN of cell i (excluding itselt), sorted from smallest to largest.
  - data.uns["W_" + rep]: kNN graph of the data in terms of affinity matrix.

Examples

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

pegasus.get_neighbors

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

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

Parameters

- **data** (*pegasusio.MultimodalData*) – An AnnData object.
- **K** (*int*, optional (default: 100)) – Number of neighbors, including the data point itself.
- **rep** (*str*, optional (default: 'pca')) – Representation used to calculate kNN. If *None* use data.X
- **n_jobs** (*int*, optional (default: -1)) – Number of threads to use. -1 refers to using all physical CPU cores.
- **random_state** (*int*, optional (default: 0)) – Random seed for random number generator.

- **full_speed** (*bool*, optional (default: False)) – If full_speed, use multiple threads in constructing hnsw index. However, the kNN results are not reproducible. If not full_speed, use only one thread to make sure results are reproducible.
- **use_cache** (*bool*, optional (default: True)) – If use_cache and found cached knn results, will not recompute.

Returns

Return type kNN indices and distances arrays.

Examples

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

pegasus.calc_kBET

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

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

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

Parameters

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

Return type `Tuple[float, float, float]`

Returns

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

Examples

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

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

pegasus.calc_kSIM

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

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

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

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **attr** (str) – The sample attribute to consider. Must exist in data.obs.
- **rep** (str, optional, default: "pca") – The embedding representation to consider. The key 'X_' + rep must exist in data.obsm.
- **K** (int, optional, default: 25) – The number of nearest neighbors to be considered.
- **min_rate** (float, optional, default: 0.9) – Acceptance rate threshold. A cell is accepted if its kSIM rate is larger than or equal to min_rate.
- **n_jobs** (int, optional, default: -1) – Number of threads used. If -1, use all physical CPU cores.
- **random_state** (int, optional, default: 0) – Random seed set for reproducing results.
- **use_cache** (bool, optional, default: True) – If use cache results for kNN.

Return type Tuple[float, float]

Returns

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

Examples

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

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

Diffusion Map

diffmap(data[, n_components, rep, solver, ...])	Calculate Diffusion Map.
calc_pseudotime(data, roots)	Calculate Pseudotime based on Diffusion Map.
infer_path(data, cluster, clust_id, path_name)	Inference on path of a cluster.

pegasus.diffmap

pegasus.**diffmap**(*data*, *n_components=100*, *rep='pca'*, *solver='eigsh'*, *max_t=5000*, *n_jobs=- 1*, *random_state=0*)

Calculate Diffusion Map.

Parameters

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

  Solver for eigen decomposition:

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

Return type None

Returns

- None
- Update data.obsm –

  – data.obsm["X_diffmap"]: Diffusion Map matrix of the data.
- Update data.uns –

  – data.uns["diffmap_evals"]: Eigenvalues corresponding to Diffusion Map matrix.

Examples

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

pegasus.calc_pseudotime

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

Calculate Pseudotime based on Diffusion Map.

Parameters

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

Return type None

Returns

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

Examples

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

pegasus.infer_path

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

Inference on path of a cluster.

Parameters

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

Returns

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

Examples

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

Cluster Algorithms

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

pegasus.cluster

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

Cluster the data using the chosen algorithm.

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

Parameters

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

- **n_init** (int, optional, default: 10) – Number of kmeans tries for the first level clustering. Default is set to be the same as scikit-learn Kmeans function.

Return type None

Returns

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

Examples

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

pegasus.louvain

pegasus.**louvain**(*data*, *rep='pca'*, *resolution=1.3*, *n_clust=None*, *random_state=0*, *class_label='louvain_labels'*)

Cluster the cells using Louvain algorithm. [Blondel08]

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X_' + rep must exist in data.obsm and nearest neighbors must be calculated so that affinity matrix 'W_' + rep exists in data.uns. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters with smaller sizes.
- **n_clust** (int, optional, default: None) – This option only takes effect if 'resolution = None'. Try to find an appropriate resolution by binary search such that the total number of clusters matches 'n_clust'. The range of resolution to search is (0.01, 2.0].
- **random_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class_label** (str, optional, default: "louvain_labels") – Key name for storing cluster labels in data.obs.

Return type None

Returns

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

Examples

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

pegasus.leiden

pegasus.**leiden**(*data*, *rep='pca'*, *resolution=1.3*, *n_clust=None*, *n_iter=- 1*, *random_state=0*, *class_label='leiden_labels'*)

Cluster the data using Leiden algorithm. [Traag19]

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X_' + rep must exist in data.obsm and nearest neighbors must be calculated so that affinity matrix 'W_' + rep exists in data.uns. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters.
- **n_clust** (int, optional, default: None) – This option only takes effect if 'resolution = None'. Try to find an appropriate resolution by binary search such that the total number of clusters matches 'n_clust'. The range of resolution to search is (0.01, 2.0].
- **n_iter** (int, optional, default: -1) – Number of iterations that Leiden algorithm runs. If -1, run the algorithm until reaching its optimal clustering.
- **random_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class_label** (str, optional, default: "leiden_labels") – Key name for storing cluster labels in data.obs.

Return type None

Returns

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

Examples

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

pegasus.spectral_louvain

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

Cluster the data using Spectral Louvain algorithm. [Li20]

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X_' + rep must exist in data.obsm. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters with smaller sizes.
- **rep_kmeans** (str, optional, default: "diffmap") – The embedding representation on which the KMeans runs. Keyword must exist in data.obsm. By default, use Diffusion Map coordinates. If diffmap is not calculated, use PCA coordinates instead.
- **n_clusters** (int, optional, default: 30) – The number of first level clusters.
- **n_clusters2** (int, optional, default: 50) – The number of second level clusters.
- **n_init** (int, optional, default: 10) – Number of kmeans tries for the first level clustering. Default is set to be the same as scikit-learn Kmeans function.
- **n_jobs** (*int*, optional (default: -1)) – Number of threads to use for the KMeans step. -1 refers to using all physical CPU cores.
- **random_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class_label** (str, optional, default: "spectral_louvain_labels") – Key name for storing cluster labels in data.obs.

Return type None

Returns

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

Examples

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

pegasus.spectral_leiden

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

Cluster the data using Spectral Leiden algorithm. [Li20]

Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X_' + rep must exist in data.obsm. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters.
- **rep_kmeans** (str, optional, default: "diffmap") – The embedding representation on which the KMeans runs. Keyword must exist in data.obsm. By default, use Diffusion Map coordinates. If diffmap is not calculated, use PCA coordinates instead.
- **n_clusters** (int, optional, default: 30) – The number of first level clusters.
- **n_clusters2** (int, optional, default: 50) – The number of second level clusters.
- **n_init** (int, optional, default: 10) – Number of kmeans tries for the first level clustering. Default is set to be the same as scikit-learn Kmeans function.
- **n_jobs** (*int*, optional (default: -1)) – Number of threads to use for the KMeans step. -1 refers to using all physical CPU cores.
- **random_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class_label** (str, optional, default: "spectral_leiden_labels") – Key name for storing cluster labels in data.obs.

Return type None

Returns

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

Examples

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

Visualization Algorithms

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

pegasus.tsne

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

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

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

Parameters

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

Return type None

Returns

- None
- Update data.obsm –

  – data.obsm['X_' + out_basis]: FI-tSNE coordinates of the data.

Examples

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

pegasus.umap

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

Calculate UMAP embedding of cells.

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

Parameters

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

Return type None

Returns

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

Examples

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

pegasus.fle

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

Construct the Force-directed (FLE) graph.

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

See [Jacomy14] for details on FLE.

Parameters

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

Return type None

Returns

- None
- Update data.obsm –

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

Examples

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

pegasus.net_umap

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

Calculate Net-UMAP embedding of cells.

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

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

See [Li20] for details.

Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (`str`, optional, default: `"pca"`) – Representation of data used for the calculation. By default, use PCA coordinates. If `None`, use the count matrix `data.X`.
- **n_jobs** (`int`, optional, default: `-1`) – Number of threads to use. If `-1`, use all physical CPU cores.
- **n_components** (`int`, optional, default: `2`) – Dimension of calculated UMAP coordinates. By default, generate 2-dimensional data for 2D visualization.
- **n_neighbors** (`int`, optional, default: `15`) – Number of nearest neighbors considered during the computation.
- **min_dist** (`float`, optional, default: `0.5`) – The effective minimum distance between embedded data points.
- **spread** (`float`, optional, default: `1.0`) – The effective scale of embedded data points.
- **random_state** (`int`, optional, default: `0`) – Random seed set for reproducing results.
- **select_frac** (`float`, optional, default: `0.1`) – Down sampling fraction on the cells.
- **select_K** (`int`, optional, default: `25`) – Number of neighbors to be used to estimate local density for each data point for down sampling.
- **select_alpha** (`float`, optional, default: `1.0`) – Weight the down sample to be proportional to `radius ** select_alpha`.
- **full_speed** (`bool`, optional, default: `False`) –
  - If `True`, use multiple threads in constructing `hnsw` index. However, the kNN results are not reproducible.
  - Otherwise, use only one thread to make sure results are reproducible.
- **net_alpha** (`float`, optional, default: `0.1`) – L2 penalty (regularization term) parameter of the deep regressor.

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

Return type None

Returns

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

Examples

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

pegasus.net_fle

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

Construct Net-Force-directed (FLE) graph.

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

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

See [Li20] for details.

Parameters

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

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

Return type None

Returns

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

Examples

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

Doublet Detection

infer_doublets(data[, channel_attr, ...])	Infer doublets by first calculating Scrublet-like [Wolock18] doublet scores and then smartly determining an appropriate doublet score cutoff [Li20-2] .
mark_doublets(data[, demux_attr, dbl_clusts])	Convert doublet prediction into doublet annotations that Pegasus can recognize.

pegasus.infer_doublets

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

Infer doublets by first calculating Scrublet-like [Wolock18] doublet scores and then smartly determining an appropriate doublet score cutoff [Li20-2] .

This function should be called after clustering if clust_attr is not None. In this case, we will test if each cluster is significantly enriched for doublets using Fisher's exact test.

Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **channel_attr** (`str`, optional, default: None) – Attribute indicating sample channels. If set, calculate scrublet-like doublet scores per channel.
- **clust_attr** (`str`, optional, default: None) – Attribute indicating cluster labels. If set, estimate proportion of doublets in each cluster and statistical significance.
- **min_cell** (`int`, optional, default: 100) – Minimum number of cells per sample to calculate doublet scores. For samples having less than 'min_cell' cells, doublet score calculation will be skipped.
- **expected_doublet_rate** (`float`, optional, default: `None`) – The expected doublet rate for the experiment. By default, calculate the expected rate based on number of cells from the 10x multiplet rate table
- **sim_doublet_ratio** (`float`, optional, default: `2.0`) – The ratio between synthetic doublets and observed cells.
- **n_prin_comps** (`int`, optional, default: `30`) – Number of principal components.
- **k** (`int`, optional, default: `None`) – Number of observed cell neighbors. If None, k = round(0.5 * sqrt(number of observed cells)). Total neighbors k_adj = round(k * (1.0 + sim_doublet_ratio)).
- **n_jobs** (`int`, optional, default: `-1`) – Number of threads to use. If `-1`, use all physical CPU cores.
- **alpha** (`float`, optional, default: `0.05`) – FDR significant level for cluster-level fisher exact test.
- **random_state** (`int`, optional, default: `0`) – Random seed for reproducing results.
- **plot_hist** (`str`, optional, default: `sample`) – If not None, plot diagnostic histograms using `plot_hist` as the prefix. If *channel_attr* is None, `plot_hist.dbl.png` is generated;

Otherwise, `plot_hist.channel_name.dbl.png` files are generated. Each figure consists of 4 panels showing histograms of doublet scores for observed cells (panel 1, density in log scale), simulated doublets (panel 2, density in log scale), KDE plot (panel 3) and signed curvature plot (panel 4) of log doublet scores for simulated doublets.

Return type None

Returns

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

Examples

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

pegasus.mark_doublets

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

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

Must run `infer_doublets` first.

Parameters

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

Return type None

Returns

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

Examples

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

Gene Module Score

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

pegasus.calc_signature_score

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

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

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

Parameters

- **data** (`MultimodalData`, `UnimodalData`, or `anndata.AnnData` object.) – Single cell expression data.
- **signatures** (`Dict[str, List[str]]` or `str`) – This argument accepts either a dictionary or a string. If `signatures` is a dictionary, it can contain multiple signature score calculation requests. Each key in the dictionary represents a separate signature score calculation and its corresponding value contains a list of gene symbols. Each score will be stored in data.obs field with key as the keyword. If `signatures` is a string, it should refer to a Gene Matrix Transposed (GMT)-formatted file. Pegasus will load signatures from the GMT file.

  Pegasus also provide 5 default signature panels for each of human and mouse. They are `cell_cycle_human`,

  - `cell_cycle_human` contains two cell-cycle signatures, `G1/S` and `G2/M`, obtained from Tirosh et al. 2016. We also updated gene symbols according to Seurat's `cc.genes.updated.2019` vector. We additionally calculate signature scores `cycling` and `cycle_diff`, which are `max{G2/M, G1/S}` and `G2/M - G1/S` respectively. We provide predicted cell cycle phases in `data.obs['predicted_phase']` in case it is useful. `predicted_phase` is predicted based on `G1/S` and `G2/M` scores. First, we identify `G0` cells. We apply KMeans algorithm to obtain 2 clusters based on the `cycling` signature. `G0` cells are from the cluster with smallest mean value. For each cell from the other cluster, if `G1/S > G2/M`, it is a `G1/S` cell, otherwise it is a `G2/M` cell.
  - `gender_human` contains two gender-specific signatures, `female_score` and `male_score`. Genes were selected based on DE analysis between genders based on 8 channels of bone marrow data from HCA Census of Immune Cells and the brain nuclei data from Gaublomme and Li et al, 2019, Nature Communications. After calculation, three signature scores will be calculated: `female_score`, `male_score` and `gender_score`. `female_score` and `male_score` are calculated based on female and male signatures respectively and a larger score represent a higher likelihood of that gender. `gender_score` is calculated as `male_score - female_score`. A large positive score likely represents male and a large negative score likely represents female. Pegasus also provides predicted gender for each cell based on `gender_score`, which is stored in `data.obs['predicted_gender']`. To predict genders, we apply the KMeans algorithm to the `gender_score` and ask for 3 clusters. The clusters with a minimum and maximum clauster centers are predicted as `female` and `male`

respectively and the cluster in the middle is predicted as `uncertain`. Note that this approach is conservative and it is likely that users can predict genders based on `gender_score` for cells in the `uncertain` cluster with a reasonable accuracy.

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

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

Return type `None`

Returns

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

Examples

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

Differential Expression Analysis

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

pegasus.de_analysis

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

Perform Differential Expression (DE) Analysis on data.

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

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

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

Parameters

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

Return type `None`

Returns

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

Examples

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

pegasus.markers

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

Extract DE results into a human readable structure.

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

Parameters

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

Returns results – A Python dictionary containing markers. If DE is performed between clusters, the structure is dict[cluster_id]['up' or 'down'][dataframe]. If DE is performed between conditions within each cluster, the structure is dict[cluster_id][condition_id]['up' or 'down'][dataframe]. 'up' refers to up-regulated genes, which should have 'auroc' > 0.5. 'down' refers to down-regulated genes, which should have 'auroc' < 0.5.

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

Examples

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

pegasus.write_results_to_excel

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

Write DE analysis results into Excel workbook.

Parameters

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

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

Return type None

Returns

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

Examples

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

Annotate clusters

infer_cell_types(data, markers[, de_test, ...])	Infer putative cell types for each cluster using legacy markers.
annotate(data, name, based_on, anno_dict)	Add annotation to the data object as a categorical variable.

pegasus.infer_cell_types

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

Infer putative cell types for each cluster using legacy markers.

Parameters

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

* 'mouse_brain' for mouse brain cells;
* 'human_lung' for human lung cells.

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

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

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

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

Examples

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

pegasus.annotate

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

Add annotation to the data object as a categorical variable.

Parameters

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

Returns

Return type None

Examples

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

Plotting

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

pegasus.scatter

pegasus.**scatter**(*data*, *attrs=None*, *basis='umap'*, *matkey=None*, *restrictions=None*, *show_background=False*, *fix_corners=True*, *alpha=1.0*, *legend_loc='right margin'*, *legend_ncol=None*, *palettes=None*, *cmaps='YlOrRd'*, *vmin=None*, *vmax=None*, *nrows=None*, *ncols=None*, *panel_size=(4, 4)*, *left=0.2*, *bottom=0.15*, *wspace=0.4*, *hspace=0.15*, *return_fig=False*, *dpi=300.0*, ***kwargs*)

Generate scatter plots for different attributes

Parameters

- **data** (pegasusio.MultimodalData) – Use current selected modality in data.
- **attrs** (str or List[str], default: None) – Color scatter plots by attrs. Each attribute in attrs can be one key in data.obs, data.var_names (e.g. one gene) or data.obsm (attribute has the format of 'obsm_key@component', like 'X_pca@0'). If one attribute is categorical, a palette will be used to color each category separately. Otherwise, a color map will be used. If no attributes are provided, plot the basis for all data.
- **basis** (str, optional, default: umap) – Basis to be used to generate scatter plots. Can be either 'umap', 'tsne', 'fitsne', 'fle', 'net_tsne', 'net_fitsne', 'net_umap' or 'net_fle'.
- **matkey** (str, optional, default: None) – If matkey is set, select matrix with matkey as keyword in the current modality. Only works for MultimodalData or UnimodalData objects.
- **restrictions** (str or List[str], optional, default: None) – A list of restrictions to subset data for plotting. There are two types of restrictions: global restriction and attribute-specific restriction. Global restriction appiles to all attributes in attrs and takes the format of 'key:value,value...', or 'key:~value,value...'. This restriction selects cells with the data.obs[key] values belong to 'value,value...' (or not belong to if

'~' shows). Attribute-specific restriction takes the format of 'attr:key:value,value...', or 'attr:key:~value,value...'. It only applies to one attribute 'attr'. If 'attr' and 'key' are the same, one can use '.' to replace 'key' (e.g. `cluster_labels:.:value1,value2`).

- **show_background** (`bool`, optional, default: False) – Only applicable if *restrictions* is set. By default, only data points selected are shown. If show_background is True, data points that are not selected will also be shown.
- **fix_corners** (`bool`, optional, default: True) – If True, fix the corners of the plots as defined using all data points.
- **alpha** (`float` or `List[float]`, optional, default: `1.0`) – Alpha value for blending, from 0.0 (transparent) to 1.0 (opaque). If this is a list, the length must match attrs, which means we set a separate alpha value for each attribute.
- **legend_loc** (`str` or `List[str]`, optional, default: `right margin`) – Legend location. Can be either "right margin" or "on data". If a list is provided, set 'legend_loc' for each attribute in 'attrs' separately.
- **legend_ncol** (`str`, optional, default: None) – Only applicable if legend_loc == "right margin". Set number of columns used to show legends.
- **palettes** (`str` or `List[str]`, optional, default: None) – Used for setting colors for every categories in categorical attributes. Each string in `palettes` takes the format of 'attr:color1,color2,...,colorn'. 'attr' is the categorical attribute and 'color1' - 'colorn' are the colors for each category in 'attr' (e.g. 'cluster_labels:black,blue,red,...,yellow'). If there is only one categorical attribute in 'attrs', `palletes` can be set as a single string and the 'attr' keyword can be omitted (e.g. "blue,yellow,red").
- **cmaps** (`str` or `List[str]`, optional, default: `YlOrRd`) – Used for setting colormap for numeric attributes. Each string in `cmaps` takes the format of 'colormap' or 'attr:colormap'. 'colormap' sets the default colormap for all numeric attributes. 'attr:colormap' overwrites attribute 'attr's colormap as 'colormap'.
- **vmin** (`float`, optional, default: None) – Minimum value to show on a numeric scatter plot (feature plot).
- **vmax** (`float`, optional, default: None) – Maximum value to show on a numeric scatter plot (feature plot).
- **nrows** (`int`, optional, default: None) – Number of rows in the figure. If not set, pegasus will figure it out automatically.
- **ncols** (`int`, optional, default: None) – Number of columns in the figure. If not set, pegasus will figure it out automatically.
- **panel_size** (*tuple*, optional (default: *(6, 4)*)) – The panel size (width, height) in inches.
- **left** (*float*, optional (default: *0.2*)) – This parameter sets the figure's left margin as a fraction of panel's width (left * panel_size[0]).
- **bottom** (*float*, optional (default: *0.15*)) – This parameter sets the figure's bottom margin as a fraction of panel's height (bottom * panel_size[1]).
- **wspace** (*float*, optional (default: *0.4*)) – This parameter sets the width between panels and also the figure's right margin as a fraction of panel's width (wspace * panel_size[0]).
- **hspace** (*float*, optional (defualt: *0.15*)) – This parameter sets the height between panels and also the figure's top margin as a fraction of panel's height (hspace * panel_size[1]).
- **return_fig** (`bool`, optional, default: `False`) – Return a `Figure` object if `True`; return `None` otherwise.

- **dpi** (float, optional, default: 300.0) – The resolution of the figure in dots-per-inch.

Returns A matplotlib.figure.Figure object containing the dot plot if return_fig == True

Return type *Figure* object

Examples

```
>>> pg.scatter(data, attrs=['louvain_labels', 'Channel'], basis='fitsne')
>>> pg.scatter(data, attrs=['CD14', 'TRAC'], basis='umap')
```

pegasus.scatter_groups

pegasus.**scatter_groups**(*data*, *attr*, *groupby*, *basis='umap'*, *matkey=None*, *restrictions=None*, *show_full=True*, *categories=None*, *alpha=1.0*, *legend_loc='right margin'*, *legend_ncol=None*, *palette=None*, *cmap='YlOrRd'*, *vmin=None*, *vmax=None*, *nrows=None*, *ncols=None*, *panel_size=(4, 4)*, *left=0.2*, *bottom=0.15*, *wspace=0.4*, *hspace=0.15*, *return_fig=False*, *dpi=300.0*, ***kwargs*)

Generate scatter plots of attribute 'attr' for each category in attribute 'group'. Optionally show scatter plot containing data points from all categories in 'group'.

Parameters

- **data** (pegasusio.MultimodalData) – Use current selected modality in data.
- **attr** (str) – Color scatter plots by attribute 'attr'. This attribute should be one key in data.obs, data.var_names (e.g. one gene) or data.obsm (attribute has the format of 'obsm_key@component', like 'X_pca@0'). If it is categorical, a palette will be used to color each category separately. Otherwise, a color map will be used.
- **groupby** (str) – Generate separate scatter plots of 'attr' for data points in each category in 'groupby', which should be a key in data.obs representing one categorical variable.
- **basis** (str, optional, default: umap) – Basis to be used to generate scatter plots. Can be either 'umap', 'tsne', 'fitsne', 'fle', 'net_tsne', 'net_fitsne', 'net_umap' or 'net_fle'.
- **matkey** (str, optional, default: None) – If matkey is set, select matrix with matkey as keyword in the current modality. Only works for MultimodalData or UnimodalData objects.
- **restrictions** (str or List[str], optional, default: None) – A list of restrictions to subset data for plotting. Each restriction takes the format of 'key:value,value…', or 'key:~value,value…'. This restriction selects cells with the data.obs[key] values belong to 'value,value…' (or not belong to if '~' shows).
- **show_full** (bool, optional, default: True) – Show the scatter plot with all categories in 'groupby' as the first plot.
- **categories** (List[str], optional, default: None) – Redefine group structure based on attribute 'groupby'. If 'categories' is not None, each string in the list takes the format of 'category_name:value,value', or 'category_name:~value,value…", where 'category_name' refers to new category name, 'value' refers to one of the category in 'groupby' and '~' refers to exclude values.
- **alpha** (float, optional, default: 1.0) – Alpha value for blending, from 0.0 (transparent) to 1.0 (opaque).

- **legend_loc** (`str`, optional, default: `right margin`) – Legend location. Can be either "right margin" or "on data".
- **legend_ncol** (`str`, optional, default: None) – Only applicable if legend_loc == "right margin". Set number of columns used to show legends.
- **palette** (`str`, optional, default: None) – Used for setting colors for one categorical attribute (e.g. "black,blue,red,...,yellow").
- **cmap** (`str`, optional, default: `YlOrRd`) – Used for setting colormap for one numeric attribute.
- **vmin** (`float`, optional, default: None) – Minimum value to show on a numeric scatter plot (feature plot).
- **vmax** (`float`, optional, default: None) – Maximum value to show on a numeric scatter plot (feature plot).
- **nrows** (`int`, optional, default: None) – Number of rows in the figure. If not set, pegasus will figure it out automatically.
- **ncols** (`int`, optional, default: None) – Number of columns in the figure. If not set, pegasus will figure it out automatically.
- **panel_size** (*tuple*, optional (default: *(6, 4)*)) – The panel size (width, height) in inches.
- **left** (*float*, optional (default: *0.2*)) – This parameter sets the figure's left margin as a fraction of panel's width (left * panel_size[0]).
- **bottom** (*float*, optional (default: *0.15*)) – This parameter sets the figure's bottom margin as a fraction of panel's height (bottom * panel_size[1]).
- **wspace** (*float*, optional (default: *0.4*)) – This parameter sets the width between panels and also the figure's right margin as a fraction of panel's width (wspace * panel_size[0]).
- **hspace** (*float*, optional (defualt: *0.15*)) – This parameter sets the height between panels and also the figure's top margin as a fraction of panel's height (hspace * panel_size[1]).
- **return_fig** (`bool`, optional, default: `False`) – Return a `Figure` object if `True`; return `None` otherwise.
- **dpi** (`float`, optional, default: 300.0) – The resolution of the figure in dots-per-inch.

Returns A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

Return type *Figure* object

Examples

```
>>> pg.scatter_groups(data, attr='louvain_labels', groupby='Individual', basis='tsne
↪', nrows = 2, ncols = 4, alpha = 0.5)
>>> pg.scatter_groups(data, attr='anno', groupby='Channel', basis='umap',␣
↪categories=['new_cat1:channel1,channel2', 'new_cat2:channel3'])
```

pegasus.compo_plot

pegasus.**compo_plot**(*data*, *groupby*, *condition*, *style='frequency'*, *restrictions=None*, *switch_axes=False*, *groupby_label=None*, *sort_function='natsorted'*, *panel_size=(6, 4)*, *palette=None*, *color_unused=False*, *left=0.15*, *bottom=0.15*, *wspace=0.3*, *hspace=0.15*, *return_fig=False*, *dpi=300.0*, ***kwargs*)

Generate a composition plot, which shows the percentage of cells from each condition for every cluster.

This function is used to generate composition plots, which are bar plots showing the cell compositions (from different conditions) for each cluster. This type of plots is useful to fast assess library quality and batch effects.

Parameters

- **data** (`AnnData` or `UnimodalData` or `MultimodalData` object) – Single cell expression data.
- **groupby** (`str`) – A categorical variable in data.obs that is used to categorize the cells, e.g. cell type.
- **condition** (`str`) – A categorical variable in data.obs that is used to calculate frequency within each category defined by `groupby`, e.g. donor.
- **style** (`str`, optional (default: `frequency`)) – Composition plot style. Can be either `frequency`, or `normalized`. Within each cluster, the `frequency` style show the percentage of cells from each `condition` within each category in `groupby` (stacked), the `normalized` style shows for each category in `groupby` the percentage of cells that are also in each `condition` over all cells that are in the same `condition` (not stacked).
- **restrictions** (`str` or `List[str]`, optional, default: None) – A list of restrictions to subset data for plotting. Each restriction takes the format of 'key:value,value…', or 'key:~value,value…'. This restriction selects cells with the `data.obs[key]` values belong to 'value,value…' (or not belong to if '~' shows).
- **switch_axes** (`bool`, optional, default: `False`) – By default, X axis is for groupby, and Y axis for frequencies with respect to condition. If this parameter is `True`, switch the axes.
- **groupby_label** (`str`, optional (default `None`)) – Label for the axis displaying `groupby` categories. If `None`, use `groupby`.
- **sort_function** (`Union[Callable[List[str], List[str]], str]`, optional, default: `natsorted`) – Function used for sorting both groupby and condition labels. If `natsorted`, apply natsorted function to sort by natural order. If `None`, don't sort. Otherwise, a callable function will be applied to the labels for sorting.
- **panel_size** (`tuple`, optional (default: `(6, 4)`)) – The plot size (width, height) in inches.
- **palette** (`List[str]`, optional (default: `None`)) – Used for setting colors for categories in `condition`. Within the list, each string is the color for one category.
- **left** (`float`, optional (default: `0.15`)) – This parameter sets the figure's left margin as a fraction of panel's width (left * panel_size[0]).
- **bottom** (`float`, optional (default: `0.15`)) – This parameter sets the figure's bottom margin as a fraction of panel's height (bottom * panel_size[1]).
- **wspace** (`float`, optional (default: `0.3`)) – This parameter sets the width between panels and also the figure's right margin as a fraction of panel's width (wspace * panel_size[0]).
- **hspace** (`float`, optional (defualt: `0.15`)) – This parameter sets the height between panels and also the figure's top margin as a fraction of panel's height (hspace * panel_size[1]).

- **return_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.

Returns A matplotlib.figure.Figure object containing the dot plot if return_fig == True

Return type Figure object

Examples

```
>>> fig = pg.compo_plot(data, 'louvain_labels', 'Donor', style = 'normalized')
```

pegasus.violin

pegasus.**violin**(*data*, *attrs*, *groupby*, *hue=None*, *matkey=None*, *stripplot=False*, *inner=None*, *scale='width'*, *panel_size=(8, 0.5)*, *palette=None*, *left=0.15*, *bottom=0.15*, *wspace=0.1*, *ylabel=None*, *return_fig=False*, *dpi=300.0*, ***kwargs*)

Generate a stacked violin plot.

Parameters

- **data** (AnnData or MultimodalData or UnimodalData object) – Single-cell expression data.
- **attrs** (str or List[str]) – Cell attributes or features to plot. Cell attributes must exist in data.obs and must be numeric. Features must exist in data.var.
- **groupby** (str) – A categorical variable in data.obs that is used to categorize the cells, e.g. Clusters.
- **hue** (str, optional, default: None) – 'hue' should be a categorical variable in data.obs that has only two levels. Set 'hue' will show us split violin plots.
- **matkey** (str, optional, default: None) – If matkey is set, select matrix with matkey as keyword in the current modality. Only works for MultimodalData or UnimodalData objects.
- **stripplot** (bool, optional, default: False) – Attach a stripplot to the violinplot or not. This option will be automatically turn off if 'hue' is set.
- **inner** (str, optional, default: None) –

  Representation of the datapoints in the violin interior:

  - If box, draw a miniature boxplot.
  - If quartiles, draw the quartiles of the distribution.
  - If point or stick, show each underlying datapoint.
  - If None, will draw unadorned violins.
- **scale** (str, optional, default: width) –

  The method used to scale the width of each violin:

  - If width, each violin will have the same width.
  - If area, each violin will have the same area.

  - If `count`, the width of the violins will be scaled by the number of observations in that bin.

- **panel_size** (`Tuple[float, float]`, optional, default: `(8, 0.5)`) – The size (width, height) in inches of each violin panel.
- **palette** (`List[str]`, optional (default: `None`)) – Used for setting colors for categories in `groupby`. Within the list, each string is the color for one category.
- **left** (`float`, optional, default: `0.15`) – This parameter sets the figure's left margin as a fraction of panel's width (left * panel_size[0]).
- **bottom** (`float`, optional, default: `0.15`) – This parameter sets the figure's bottom margin as a fraction of panel's height (bottom * panel_size[1]).
- **wspace** (`float`, optional, default: `0.1`) – This parameter sets the width between panels and also the figure's right margin as a fraction of panel's width (wspace * panel_size[0]).
- **ylabel** (`str`, optional, default: `None`) – Y-axis label. No label to show if `None`.
- **return_fig** (`bool`, optional, default: `False`) – Return a `Figure` object if `True`; return `None` otherwise.
- **dpi** (`float`, optional, default: `300.0`) – The resolution in dots per inch.
- **kwargs** – Are passed to `seaborn.violinplot`.

Returns A `matplotlib.figure.Figure` object containing the dot plot if `show == False`

Return type `Figure` object

Examples

```
>>> pg.violin(data, attrs=['CD14', 'TRAC', 'CD34'], groupby='louvain_labels')
```

pegasus.heatmap

pegasus.**heatmap**(*data*, *attrs*, *groupby*, *matkey=None*, *on_average=True*, *switch_axes=False*, *attrs_cluster=False*, *attrs_dendrogram=True*, *groupby_cluster=True*, *groupby_dendrogram=True*, *attrs_labelsize=10.0*, *groupby_labelsize=10.0*, *cbar_labelsize=10.0*, *panel_size=(10, 10)*, *return_fig=False*, *dpi=300.0*, ***kwargs*)

Generate a heatmap.

Parameters

- **data** (`AnnData` or `MultimodalData` or `UnimodalData` object) – Single-cell expression data.
- **attrs** (`str` or `List[str]`) – Cell attributes or features to plot. Cell attributes must exist in `data.obs` and must be numeric. Features must exist in `data.var`. By default, attrs are plotted as columns.
- **groupby** (`str`) – A categorical variable in data.obs that is used to categorize the cells, e.g. Clusters. By default, data.obs['groupby'] is plotted as rows.
- **matkey** (`str`, optional, default: `None`) – If matkey is set, select matrix with matkey as keyword in the current modality. Only works for MultimodalData or UnimodalData objects.

- **on_average** (bool, optional, default: True) – If True, plot cluster average gene expression (i.e. show a Matrixplot); otherwise, plot a general heatmap.
- **switch_axes** (bool, optional, default: False) – By default, X axis is for attributes, and Y axis for clusters. If this parameter is True, switch the axes. Moreover, with on_average being False, if switch_axes is False, row_cluster is enforced to be False; if switch_axes is True, col_cluster is enforced to be False.
- **attrs_cluster** (bool, optional, default: False) – Cluster attributes and generate a attribute-wise dendrogram.
- **attrs_dendrogram** (bool, optional, default: True) – Only matters if attrs_cluster is True. Show the dendrogram if this option is True.
- **groupby_cluster** (bool, optional, default: True) – Cluster data.obs['groupby'] and generate a cluster-wise dendrogram.
- **groupby_dendrogram** (bool, optional, default: True) – Only matters if groupby_cluster is True. Show the dendrogram if this option is True.
- **attrs_labelsize** (float, optional, default: 10.0) – Fontsize for labels of attrs.
- **groupby_labelsize** (float, optional, default: 10.0) – Fontsize for labels of data.obs['groupby'].
- **cbar_labelsize** (float, optional, default: 10.0) – Fontsize of the color bar.
- **panel_size** (Tuple[float, float], optional, default: (10, 10)) – Overall size of the heatmap in (width, height) form.
- **return_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.
- **kwargs** – Are passed to seaborn.heatmap.
- **_colormap documentation** (.) –

Returns A matplotlib.figure.Figure object containing the dot plot if return_fig == True

Return type Figure object

Examples

```
>>> pg.heatmap(data, genes=['CD14', 'TRAC', 'CD34'], groupby='louvain_labels')
```

pegasus.dotplot

pegasus.**dotplot**(*data*, *genes*, *groupby*, *reduce_function=<function mean>*, *fraction_min=0*, *fraction_max=None*, *dot_min=0*, *dot_max=20*, *switch_axes=False*, *cmap='Reds'*, *sort_function='natsorted'*, *grid=True*, *return_fig=False*, *dpi=300.0*, ***kwds*)

Generate a dot plot.

Parameters

- **data** (AnnData or UnimodalData or MultimodalData object) – Single cell expression data.
- **genes** (str or List[str]) – Features to plot.

- **groupby** (str) – A categorical variable in data.obs that is used to categorize the cells, e.g. Clusters.
- **reduce_function** (Callable[[np.ndarray], float], optional, default: np.mean) – Function to calculate statistic on expression data. Default is mean.
- **fraction_min** (float, optional, default: 0.) – Minimum fraction of expressing cells to consider.
- **fraction_max** (float, optional, default: None.) – Maximum fraction of expressing cells to consider. If None, use the maximum value from data.
- **dot_min** (int, optional, default: 0.) – Minimum size in pixels for dots.
- **dot_max** (int, optional, default: 20.) – Maximum size in pixels for dots.
- **switch_axes** (bool, optional, default: False.) – If True, switch X and Y axes.
- **cmap** (str or List[str] or Tuple[str], optional, default: Reds) – Color map.
- **sort_function** (Union[Callable[List[str], List[str]], str], optional, default: natsorted) – Function used for sorting groupby labels. If natsorted, apply natsorted function to sort by natural order. If None, don't sort. Otherwise, a callable function will be applied to the labels for sorting.
- **grid** (bool, optional, default: True) – If True, plot grids.
- **return_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.
- ****kwds** – Are passed to matplotlib.pyplot.scatter.

Returns A matplotlib.figure.Figure object containing the dot plot if return_fig == True

Return type Figure object

Examples

```
>>> pg.dotplot(data, genes = ['CD14', 'TRAC', 'CD34'], groupby = 'louvain_labels')
```

pegasus.dendrogram

pegasus.**dendrogram**(*data*, *groupby*, *rep='pca'*, *genes=None*, *correlation_method='pearson'*, *n_clusters=None*, *affinity='euclidean'*, *linkage='complete'*, *compute_full_tree='auto'*, *distance_threshold=0*, *panel_size=(6, 6)*, *orientation='top'*, *color_threshold=None*, *return_fig=False*, *dpi=300.0*, ***kwargs*)

Generate a dendrogram on hierarchical clustering result.

The metrics used here are consistent with SCANPY's dendrogram implementation.

scikit-learn Agglomerative Clustering implementation is used for hierarchical clustering.

Parameters

- **data** (MultimodalData, UnimodalData, or AnnData object) – Single cell expression data.

- **genes** (List[str], optional, default: None) – List of genes to use. Gene names must exist in data.var. If set, use the counts in data.X for plotting; if set as None, use the embedding specified in rep for plotting.
- **rep** (str, optional, default: pca) – Cell embedding to use. It only works when genes``is ``None, and its key "X_"+rep must exist in data.obsm. By default, use PCA coordinates.
- **groupby** (str) – Categorical cell attribute to plot, which must exist in data.obs.
- **correlation_method** (str, optional, default: pearson) – Method of correlation between categories specified in data.obs. Available options are: pearson, kendall, spearman. See pandas corr documentation for details.
- **n_clusters** (int, optional, default: None) – The number of clusters to find, used by hierarchical clustering. It must be None if distance_threshold is not None.
- **affinity** (str, optional, default: correlation) –

  Metric used to compute the linkage, used by hierarchical clustering. Valid values for metric are:

  - From scikit-learn: cityblock, cosine, euclidean, l1, l2, manhattan.
  - From scipy.spatial.distance: braycurtis, canberra, chebyshev, correlation, dice, hamming, jaccard, kulsinski, mahalanobis, minkowski, rogerstanimoto, russellrao, seuclidean, sokalmichener, sokalsneath, sqeuclidean, yule.

  Default is the correlation distance. See scikit-learn distance documentation for details.
- **linkage** (str, optional, default: complete) –

  Which linkage criterion to use, used by hierarchical clustering. Below are available options:

  - ward minimizes the variance of the clusters being merged.
  - avarage uses the average of the distances of each observation of the two sets.
  - complete uses the maximum distances between all observations of the two sets. (Default)
  - single uses the minimum of the distances between all observations of the two sets.

  See scikit-learn documentation for details.
- **compute_full_tree** (str or bool, optional, default: auto) – Stop early the construction of the tree at n_clusters, used by hierarchical clustering. It must be True if distance_threshold is not None. By default, this option is auto, which is True if and only if distance_threshold is not None, or n_clusters is less than min(100, 0.02 * n_groups), where n_groups is the number of categories in data.obs[groupby].
- **distance_threshold** (float, optional, default: 0) – The linkage distance threshold above which, clusters will not be merged. If not None, n_clusters must be None and compute_full_tree must be True.
- **panel_size** (Tuple[float, float], optional, default: (6, 6)) – The size (width, height) in inches of figure.
- **orientation** (str, optional, default: top) – The direction to plot the dendrogram. Available options are: top, bottom, left, right. See scipy dendrogram documentation for explanation.

- **color_threshold** (float, optional, default: None) – Threshold for coloring clusters. See scipy dendrogram documentation for explanation.
- **return_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.
- ****kwargs** – Are passed to scipy.cluster.hierarchy.dendrogram.

Returns A matplotlib.figure.Figure object containing the dot plot if return_fig == True

Return type Figure object

Examples

```
>>> pg.dendrogram(data, genes=data.var_names, groupby='louvain_labels')
>>> pg.dendrogram(data, rep='pca', groupby='louvain_labels')
```

pegasus.hvfplot

pegasus.**hvfplot**(*data*, *top_n=20*, *panel_size=(6, 4)*, *return_fig=False*, *dpi=300.0*)

Generate highly variable feature plot. Only works for HVGs returned by highly_variable_features method with flavor=='pegasus'.

Parameters

- **data** (MultimodalData, UnimodalData, or anndata.AnnData object.) – Single cell expression data.
- **top_n** (int, optional, default: 20) – Number of top highly variable features to show names.
- **panel_size** (Tuple[float, float], optional, default: (6, 4)) – The size (width, height) in inches of figure.
- **return_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.

Returns A matplotlib.figure.Figure object containing the dot plot if return_fig == True

Return type Figure object

Examples

```
>>> pg.hvfplot(data)
>>> pg.hvfplot(data, top_n=10, dpi=150)
```

pegasus.qcviolin

pegasus.**qcviolin**(*data*, *plot_type*, *min_genes_before_filt=100*, *n_violin_per_panel=8*, *panel_size=(6, 4)*, *left=0.2*, *bottom=0.15*, *wspace=0.3*, *hspace=0.35*, *return_fig=False*, *dpi=300.0*)

Plot quality control statistics (before filtration vs. after filtration) as violin plots. Require statistics such as "n_genes", "n_counts" and "percent_mito" precomputed.

Parameters

- **data** (`MultimodalData`, `UnimodalData`, or `anndata.AnnData` object.) – Single cell expression data.
- **plot_type** (`str`) – Choose from `gene`, `count` and `mito`, which shows number of expressed genes, number of UMIs and percentage of mitochondrial rate.
- **min_genes_before_filt** (`int`, optional, default: 100) – If data loaded are raw data (i.e. min(n_genes) == 0), filter out cell barcodes with less than `min_genes_before_filt` for better visual effects.
- **n_violin_per_panel** (`int`, optional, default: 8) – Number of violin plots (samples) shown in one panel.
- **panel_size** (*tuple*, optional (default: *(6, 4)*)) – The panel size (width, height) in inches.
- **left** (*float*, optional (default: *0.2*)) – This parameter sets the figure's left margin as a fraction of panel's width (left * panel_size[0]).
- **bottom** (*float*, optional (default: *0.15*)) – This parameter sets the figure's bottom margin as a fraction of panel's height (bottom * panel_size[1]).
- **wspace** (*float*, optional (default: *0.4*)) – This parameter sets the width between panels and also the figure's right margin as a fraction of panel's width (wspace * panel_size[0]).
- **hspace** (*float*, optional (defualt: *0.15*)) – This parameter sets the height between panels and also the figure's top margin as a fraction of panel's height (hspace * panel_size[1]).
- **return_fig** (`bool`, optional, default: `False`) – Return a `Figure` object if `True`; return `None` otherwise.
- **dpi** (`float`, optional, default: `300.0`) – The resolution in dots per inch.

Returns A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

Return type `Figure` object

Examples

```
>>> pg.qcviolin(data, "mito", dpi = 500)
```

pegasus.ridgeplot

pegasus.**ridgeplot**(*data*, *features*, *donor_attr=None*, *qc_attr=None*, *overlap=0.5*, *left_adjust=0.35*, *panel_size=(6, 4)*, *return_fig=False*, *dpi=300.0*, ***kwargs*)

Generate ridge plots, up to 8 features can be shown in one figure.

Parameters

- **data** (*UnimodalData* or *MultimodalData* object) – CITE-Seq or Cyto data.
- **features** (*str* or *List[str]*) – One or more features to display.
- **donor_attr** (*str*, optional, default None) – If not None, *features* must contain only one feature, plot this feature by donor indicated as *donor_attr*.
- **qc_attr** (*str*, optional, default None) – If not None, only data.obs[qc_attr] == True are used.
- **overlap** (*float*, default 0.5) – Overlap between adjacent ridge plots (top and bottom).
- **left_adjust** (*float*, default 0.35) – Left margin for displaying labels.
- **panel_size** (*tuple*, optional (default: *(6, 4)*)) – The plot size (width, height) in inches.
- **return_fig** (`bool`, optional, default: `False`) – Return a `Figure` object if `True`; return `None` otherwise.
- **dpi** (`float`, optional, default: `300.0`) – The resolution in dots per inch.

Returns A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

Return type *Figure* object

Examples

```
>>> fig = pg.ridgeplot(data, features = ['CD8', 'CD4', 'CD3'], show = False, dpi =␣
↪500)
>>> fig = pg.ridgeplot(data, features = 'CD3', donor_attr = 'assignment', show =␣
↪False, dpi = 500)
```

pegasus.volcano

pegasus.**volcano**(*data*, *cluster_id*, *de_key='de_res'*, *de_test='mwu'*, *qval_threshold=0.05*, *log2fc_threshold=1.0*, *top_n=20*, *panel_size=(6, 4)*, *return_fig=False*, *dpi=300.0*)

Generate Volcano plots (-log10 p value vs. log2 fold change) for visualizing DE results.

Parameters

- **data** (`MultimodalData`, `UnimodalData`, or `anndata.AnnData` object.) – Single cell expression data.
- **cluster_id** (`str`) –

  Cluster ID for the cluster we want to show DE results. There are two cases:

  - If `condition` is `None` in `pg.de_analysis`: Just specify one cluster label in the cluster attribute used in `pg.de_analysis`.

  - If `condition` is not `None` in `pg.de_analysis`: Specify cluster ID in this format: **"cluster_label:cond_level"**, where **cluster_label** is the cluster label, and **cond_level** is the condition ID. And this shows result of cells within the cluster under the specific condition.

- **de_key** (`str`, optional, default: `de_res`) – The varm keyword for DE results. data.varm[de_key] should store the full DE result table.
- **de_test** (`str`, optional, default: `mwu`) – Which DE test results to show. Use MWU test result by default.
- **qval_threshold** (`float`, optional, default: 0.05.) – Selected FDR rate. A horizontal line indicating this rate will be shown in the figure.
- **log2fc_threshold** (`float`, optional, default: 1.0) – Log2 fold change threshold to highlight biologically interesting genes. Two vertical lines representing negative and positive log2 fold change will be shown.
- **top_n** (`int`, optional, default: `20`) – Number of top DE genes to show names. Genes are ranked by Log2 fold change.
- **panel_size** (`Tuple[float, float]`, optional, default: `(6, 4)`) – The size (width, height) in inches of figure.
- **return_fig** (`bool`, optional, default: `False`) – Return a `Figure` object if `True`; return `None` otherwise.
- **dpi** (`float`, optional, default: `300.0`) – The resolution in dots per inch.

Returns A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

Return type `Figure` object

Examples

```
>>> pg.volcano(data, cluster_id = '1', dpi=200)
```

Demultiplexing

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

pegasus.estimate_background_probs

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

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

Parameters

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

Return type None

Returns

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

Example

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

pegasus.demultiplex

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

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

Parameters

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

Returns

- None
- Update data.obs –
  - data.obs["demux_type"]: Demultiplexed types of the cells. Either singlet, doublet, or unknown.

- `data.obs["assignment"]`: Assigned samples of origin for each cell barcode.
- `data.obs["assignment.dedup"]`: Only exist if one sample name can correspond to multiple feature barcodes. In this case, each feature barcode is assigned a unique sample name.

Examples

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

pegasus.attach_demux_results

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

Write demultiplexing results into raw gene expression matrix.

Parameters

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

Return type MultimodalData

Returns

- MultimodalData
- *A multimodal data object.*

Examples

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

Miscellaneous

search_genes(data, gene_list[, rec_key, measure])	Extract and display gene expressions for each cluster.
search_de_genes(data, gene_list[, rec_key, ...])	Extract and display differential expression analysis results of markers for each cluster.
find_outlier_clusters(data, cluster_attr, ...)	Using MWU test to detect if any cluster is an outlier regarding one of the qc attributes: n_genes, n_counts, percent_mito.
find_markers(data, label_attr[, de_key, ...])	Find markers using gradient boosting method.

pegasus.search_genes

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

Extract and display gene expressions for each cluster.

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

Parameters

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

  Can be either "percentage" or "mean_logExpr":

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

Returns A data frame containing marker expressions in each cluster.

Return type `pandas.DataFrame`

Examples

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

pegasus.search_de_genes

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

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

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

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

Parameters

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

- **de_test** (str, optional, default: "fisher") – Differential expression test to look at, could be either t, fisher or mwu.
- **de_alpha** (float, optional, default: 0.05) – False discovery rate.
- **thre** (float, optional, default: 1.5) – Fold change threshold to determine if the marker is a strong DE (++ or --) or weak DE (+ or -).

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

Return type pandas.DataFrame

Examples

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

pegasus.find_outlier_clusters

pegasus.**find_outlier_clusters**(*data*, *cluster_attr*, *qc_attr*)

Using MWU test to detect if any cluster is an outlier regarding one of the qc attributes: n_genes, n_counts, percent_mito.

Parameters

- **data** (MultimodalData or UnimodalData object) – Annotated data matrix with rows for cells and columns for genes.
- **cluster_attr** (str) – Attribute in data.obs representing cluster assignment, e.g. 'louvain_labels'
- **qc_attr** (str) – One of the QC attribute, choosing from 'n_genes', 'n_counts' and 'percent_mito'.

Return type DataFrame

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. -1 refers to using all physical CPU cores.
- **min_gain** (float, optional, default: 1.0) – Only report genes with a feature importance score (in gain) of at least min_gain.
- **random_state** (int, optional, default: 0) – Random seed set for reproducing results.

- **remove_ribo** (bool, optional, default: False) – If True, remove ribosomal genes with either RPL or RPS as prefixes.

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

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

Examples

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

1.1.6 Release Notes

Note: Also see the release notes of PegasusIO.

Version 1.4

1.4.0 June 24, 2021

- Add nmf and integrative_nmf functions to compute NMF and iNMF using nmf-torch package; integrative_nmf supports quantile normalization proposed in the *LIGER* papers ([Welch19], [Gao21]).
- Change the parameter defaults of function qc_metrics: Now all defaults are None, meaning not performing any filtration on cell barcodes.
- In **Annotate Clusters** API functions:
  - Improve human immune cell markers and auto cell type assignment for human immune cells. (infer_cell_types function)
  - Update mouse brain cell markers (infer_cell_types function)
  - annotate function now adds annotation as a categorical variable and sort categories in natural order.
- Add find_outlier_clusters function to detect if any cluster is an outlier regarding one of the qc attributes (n_genes, n_counts, percent_mito) using MWU test.
- In **Plotting** API functions:
  - scatter function now plots all cells if attrs == None; Add fix_corners option to fix the four corners when only a subset of cells is shown.
  - Fix a bug in heatmap plotting function.
- Fix bugs in functions spectral_leiden and spectral_louvain.
- Improvements:
  - Support *umap-learn* v0.5+. (umap and net_umap functions)
  - Update doublet detection algorithm. (infer_doublets function)
  - Improve message reporting execution time spent each step. (Pegasus command line tool)

Version 1.3

1.3.0 February 2, 2021

- Make PCA more reproducible. No need to keep options for robust PCA calculation:
  - In `pca` function, remove argument `robust`.
  - In `infer_doublets` function, remove argument `robust`.
  - In **pegasus cluster** command, remove option `--pca-robust`.
- Add control on number of parallel threads for OpenMP/BLAS.
  - Now n_jobs = -1 refers to use all physical CPU cores instead of logcal CPU cores.
- Remove function `reduce_diffmap_to_3d`. In **pegasus cluster** command, remove option `--diffmap-to-3d`.
- Enhance compo_plot and dotplot functions' usability.
- Bug fix.

Version 1.2

1.2.0 December 25, 2020

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

Version 1.1

1.1.0 December 7, 2020

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

Version 1.0

1.0.0 September 22, 2020

- New features:
  - Use `zarr` file format to handle data, which has a better I/O performance in general.
  - Multi-modality support:
    - Data are manipulated in Multi-modal structure in memory.
    - Support focus analysis on Unimodal data, and appending other Unimodal data to it. (`--focus` and `--append` options in `cluster` command)
  - Calculate signature / gene module scores. (calc_signature_score)
  - Doublet detection based on Scrublet: `run_scrublet`, `infer_doublets`, and `mark_doublets`.
  - Principal-Component-level regress out. (pc_regress_out)
  - Batch correction using Scanorama. (run_scanorama)
  - Allow DE analysis with sample attribute as condition. (Set `condition` argument in de_analysis)
  - Use static plots to show results (see Plotting):
    - Provide static plots: composition plot, embedding plot (e.g. tSNE, UMAP, FLE, etc.), dot plot, feature plot, and volcano plot;
    - Add more gene-specific plots: dendrogram, heatmap, violin plot, quality-control violin, HVF plot.
- Deprecations:
  - No longer support `h5sc` file format, which was the output format of `aggregate_matrix` command in Pegasus version `0.x`.
  - Remove `net_fitsne` function.
- API changes:
  - In cell quality-control, default percent of mitochondrial genes is changed from **10.0** to **20.0**. (`percent_mito` argument in qc_metrics; `--percent-mito` option in `cluster` command)

  - Move gene quality-control out of `filter_data` function to be a separate step. (identify_robust_genes)
  - DE analysis now uses MWU test by default, not t test. (de_analysis)
  - infer_cell_types uses MWU test as the default `de_test`.
- Performance improvement:
  - Speed up MWU test in DE analysis, which is inspired by Presto.
  - Integrate Fisher's exact test via Cython in DE analysis to improve speed.
- Other highlights:
  - Make I/O and count matrices aggregation a dedicated package PegasusIO.
  - Tutorials:
    * Update Analysis tutorial;
    * Add 3 more tutorials: one on plotting library, one on batch correction and data integration, and one on regress out.

Version 0.x

0.17.2 June 26, 2020

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

0.17.1 April 6, 2020

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

0.17.0 March 10, 2020

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

0.16.11 February 28, 2020

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

0.16.9 February 17, 2020

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

0.16.8 January 30, 2020

Allow input annotated `loom` file.

0.16.7 January 28, 2020

Allow input `mtx` files of more filename formats.

0.16.5 January 23, 2020

Add Harmony algorithm for data integration.

0.16.3 December 17, 2019

Add support for loading mtx files generated from BUStools.

0.16.2 December 8, 2019

Fix bug in 'subcluster' command.

0.16.1 December 4, 2019

Fix one bug in clustering pipeline.

0.16.0 December 3, 2019

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

0.15.0 October 2, 2019

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

0.14.0 September 17, 2019

Provide Python API for interactive analysis.

0.10.0 January 31, 2019

Added 'find_markers' command to find markers using LightGBM.

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

0.9.0 January 17, 2019

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

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

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

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

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

0.8.0 November 26, 2018

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

0.7.0 October 26, 2018

Added support for CITE-Seq analysis.

0.6.0 October 23, 2018

Renamed scrtools to scCloud.

Added demuxEM module for cell/nuclei-hashing.

0.5.0 August 21, 2018

Fixed a problem related AnnData.

Added support for BigQuery.

0.4.0 August 2, 2018

Added mouse brain markers.

Allow aggregate matrix to take 'Sample' as attribute.

0.3.0 June 26, 2018

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

1.1.7 References

1.1.8 Contributors

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

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

1.1.9 Contact us

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

BIBLIOGRAPHY

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

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

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

[Gao21] Ch. Gao, et al., "Iterative single-cell multi-omic integration using online learning", In Nature Biotechnology, 2021.

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

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

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

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

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

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

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

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

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

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

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

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

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

[Welch19] J. D. Welch, et al., "Single-Cell Multi-omic Integration Compares and Contrasts Features of Brain Cell Identity" In Cell, 2019.

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

INDEX

A

aggregate_matrices() (*in module pegasus*), 31
annotate() (*in module pegasus*), 69
attach_demux_results() (*in module pegasus*), 85

C

calc_kBET() (*in module pegasus*), 47
calc_kSIM() (*in module pegasus*), 48
calc_pseudotime() (*in module pegasus*), 50
calc_signature_score() (*in module pegasus*), 64
cluster() (*in module pegasus*), 51
compo_plot() (*in module pegasus*), 74
correct_batch() (*in module pegasus*), 42

D

de_analysis() (*in module pegasus*), 66
demultiplex() (*in module pegasus*), 84
dendrogram() (*in module pegasus*), 78
diffmap() (*in module pegasus*), 49
dotplot() (*in module pegasus*), 77

E

estimate_background_probs() (*in module pegasus*), 84

F

filter_data() (*in module pegasus*), 35
find_markers() (*in module pegasus*), 87
find_outlier_clusters() (*in module pegasus*), 87
fle() (*in module pegasus*), 58

G

get_filter_stats() (*in module pegasus*), 34
get_neighbors() (*in module pegasus*), 46

H

heatmap() (*in module pegasus*), 76
highly_variable_features() (*in module pegasus*), 36
hvfplot() (*in module pegasus*), 80

I

identify_robust_genes() (*in module pegasus*), 35
infer_cell_types() (*in module pegasus*), 68
infer_doublets() (*in module pegasus*), 62
infer_path() (*in module pegasus*), 50
integrative_nmf() (*in module pegasus*), 44

L

leiden() (*in module pegasus*), 53
log_norm() (*in module pegasus*), 36
louvain() (*in module pegasus*), 52

M

mark_doublets() (*in module pegasus*), 63
markers() (*in module pegasus*), 67

N

neighbors() (*in module pegasus*), 45
net_fle() (*in module pegasus*), 60
net_umap() (*in module pegasus*), 59
nmf() (*in module pegasus*), 39

P

pca() (*in module pegasus*), 38

Q

qc_metrics() (*in module pegasus*), 33
qcviolin() (*in module pegasus*), 81

R

read_input() (*in module pegasus*), 30
regress_out() (*in module pegasus*), 40
ridgeplot() (*in module pegasus*), 82
run_harmony() (*in module pegasus*), 42
run_scanorama() (*in module pegasus*), 43

S

scatter() (*in module pegasus*), 70
scatter_groups() (*in module pegasus*), 72
search_de_genes() (*in module pegasus*), 86
search_genes() (*in module pegasus*), 86

select_features() (*in module pegasus*), 37
set_group_attribute() (*in module pegasus*), 41
spectral_leiden() (*in module pegasus*), 55
spectral_louvain() (*in module pegasus*), 54

T

tsne() (*in module pegasus*), 56

U

umap() (*in module pegasus*), 57

V

violin() (*in module pegasus*), 75
volcano() (*in module pegasus*), 82

W

write_output() (*in module pegasus*), 31
write_results_to_excel() (*in module pegasus*), 67