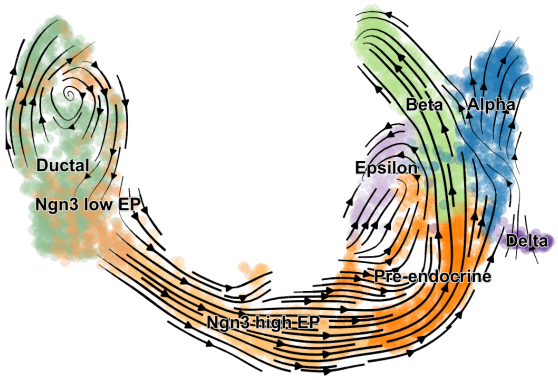
scVelo documentation

Release 0.2.5.dev3+g8d3a346.d20210826

Volker Bergen

Aug 26, 2021

MAIN



scVelo is a scalable toolkit for RNA velocity analysis in single cells, based on Bergen et al. (Nature Biotech, 2020).

RNA velocity enables the recovery of directed dynamic information by leveraging splicing kinetics. scVelo generalizes the concept of RNA velocity (La Manno et al., Nature, 2018) by relaxing previously made assumptions with a stochastic and a dynamical model that solves the full transcriptional dynamics. It thereby adapts RNA velocity to widely varying specifications such as non-stationary populations.

scVelo is compatible with scanpy and hosts efficient implementations of all RNA velocity models.

CHAPTER

ONE

SCVELO'S KEY APPLICATIONS

- estimate RNA velocity to study cellular dynamics.
- identify putative driver genes and regimes of regulatory changes.
- infer a latent time to reconstruct the temporal sequence of transcriptomic events.
- estimate reaction rates of transcription, splicing and degradation.
- use statistical tests, e.g., to detect different kinetics regimes.

scVelo has, for instance, recently been used to study immune response in COVID-19 patients and dynamic processes in human lung regeneration. Find out more in this list of application examples.

CHAPTER

TWO

LATEST NEWS

- Aug/2021: Perspectives paper out in MSB
- Feb/2021: scVelo goes multi-core
- Dec/2020: Cover of Nature Biotechnology
- Nov/2020: Talk at Single Cell Biology
- Oct/2020: Helmholtz Best Paper Award
- Oct/2020: Map cell fates with CellRank
- Sep/2020: Talk at Single Cell Omics
- Aug/2020: scVelo out in Nature Biotech

CHAPTER

THREE

REFERENCES

La Manno *et al.* (2018), RNA velocity of single cells, Nature.

Bergen *et al.* (2020), Generalizing RNA velocity to transient cell states through dynamical modeling, Nature Biotech.

Bergen *et al.* (2021), RNA velocity - current challenges and future perspectives, Molecular Systems Biology.

CHAPTER

FOUR

SUPPORT

Found a bug or would like to see a feature implemented? Feel free to submit an issue. Have a question or would like to start a new discussion? Head over to GitHub discussions. In either case, you can also always send us an email. Your help to improve scVelo is highly appreciated. For further information visit scvelo.org.

4.1 About scVelo

Measuring gene activity in individual cells requires destroying these cells to read out their content, making it challenging to study dynamic processes and to learn about cellular decision making. The introduction of RNA velocity by La Manno et al. (Nature, 2018) has enabled the recovery of directed dynamic information by leveraging the fact that newly transcribed, unspliced pre-mRNAs and mature, spliced mRNAs can be distinguished in common single-cell RNA-seq protocols, the former detectable by the presence of introns. This concept of measuring not only gene activity, but also their changes in individual cells (RNA velocity), has opened up new ways of studying cellular differentiation. The originally proposed framework obtains velocities as the deviation of the observed ratio of spliced and unspliced mRNA from an inferred steady state. Errors in velocity estimates arise if the central assumptions of a common splicing rate and the observation of the full splicing dynamics with steady-state mRNA levels are violated.

With scVelo, developed by Bergen et al. (Nature Biotechnology, 2020), these restrictions are addressed by solving the full transcriptional dynamics of splicing kinetics using a likelihood-based dynamical model. This generalizes RNA velocity to a wide variety of systems comprising transient cell states, which are common in development and in response to perturbations. Further, scVelo infers gene-specific rates of transcription, splicing and degradation, and recovers the latent time of the underlying cellular processes. This latent time represents the cell's internal clock and approximates the real time experienced by cells as they differentiate, based only on its transcriptional dynamics. Moreover, scVelo identifies regimes of regulatory changes such as stages of cell fate commitment and, therein, systematically detects putative driver genes.

4.1.1 RNA velocity models

With RNA velocity, inference of directional trajectories is explored by connecting measurements to the underlying mRNA splicing kinetics: Transcriptional induction for a particular gene results in an increase of (newly transcribed) precursor unspliced mRNAs while, conversely, repression or absence of transcription results in a decrease of unspliced mRNAs. Hence, by distinguishing unspliced from spliced mRNA, the change of mRNA abundance (RNA velocity) can be approximated. The combination of velocities across mRNAs can then be used to estimate the future state of an individual cell.

RNA velocity estimation can currently be tackled with three existing approaches:

- steady-state / deterministic model (using steady-state residuals)
- stochastic model (using second-order moments),
- dynamical model (using a likelihood-based framework).

The **steady-state / deterministic model**, as being used in velocyto, estimates velocities as follows: Under the assumption that transcriptional phases (induction and repression) last sufficiently long to reach a steady-state equilibrium (active and inactive), velocities are quantified as the deviation of the observed ratio from its steady-state ratio. The equilibrium mRNA levels are approximated with a linear regression on the presumed steady states in the lower and upper quantiles. This simplification makes two fundamental assumptions: a common splicing rate across genes and steady-state mRNA levels to be reflected in the data. It can lead to errors in velocity estimates and cellular states as the assumptions are often violated, in particular when a population comprises multiple heterogeneous subpopulation dynamics.

The **stochastic model** aims to better capture the steady states. By treating transcription, splicing and degradation as probabilistic events, the resulting Markov process is approximated by moment equations. By including second-order moments, it exploits not only the balance of unspliced to spliced mRNA levels but also their covariation. It has been demonstrated on the endocrine pancreas that stochasticity adds valuable information, overall yielding higher consistency than the deterministic model, while remaining as efficient in computation time.

The **dynamical model** (most powerful while computationally most expensive) solves the full dynamics of splicing kinetics for each gene. It thereby adapts RNA velocity to widely varying specifications such as non-stationary populations, as does not rely on the restrictions of a common splicing rate or steady states to be sampled.

The splicing dynamics

$$\frac{du(t)}{dt} = \alpha_k(t) - \beta u(t), \tag{4.1}$$

$$\frac{ds(t)}{dt} = \beta u(t) - \gamma s(t) \tag{4.2}$$

is solved in a likelihood-based expectation-maximization framework, by iteratively estimating the parameters of reaction rates and latent cell-specific variables, i.e. transcriptional state *k* and cell-internal latent time *t*.

It thereby aims to learn the unspliced/spliced phase trajectory. Four transcriptional states are modeled to account for all possible configurations of gene activity: two dynamic transient states (induction and repression) and two steady states (active and inactive) potentially reached after each dynamic transition.

In the expectation step, for a given model estimate of the unspliced/spliced phase trajectory, a latent time is assigned to an observed mRNA value by minimizing its distance to the phase trajectory. The transcriptional states are then assigned by associating a likelihood to the respective segments on the phase trajectory (induction, repression, active and inactive steady states). In the maximization step, the overall likelihood is then optimized by updating the parameters of reaction rates.

The model yields more consistent velocity estimates and better identification of transcriptional states. It further enables the systematic identification of dynamics-driving genes in a likelihood-based way, thereby finding the key drivers that govern cell fate transitions. Moreover, the dynamical model infers a universal cell-internal latent time shared across genes that enables relating genes and identifying regimes of transcriptional changes.

For best results and above-described additional insights, we recommend using the dynamical model. If runtime is of importance, the stochastic model is advised to be used as it very efficiently approximates the dynamical model, taking few minutes on 30k cells. The dynamical yet can take up to one hour, however, enhancing efficiency is in progress.

See Bergen et al. (2020) for a detailed exposition of the methods.

4.2 Installation

scVelo requires Python 3.6 or later. We recommend to use Miniconda.

4.2.1 PyPI

Install scVelo from PyPI using:

```
pip install -U scvelo
```

`-U` is short for `--upgrade`. If you get a `Permission denied` error, use `pip install -U scvelo --user` instead.

4.2.2 Development Version

To work with the latest development version, install from GitHub using:

```
pip install git+https://github.com/theislab/scvelo@develop
```

or:

```
git clone https://github.com/theislab/scvelo && cd scvelo
git checkout --track origin/develop
pip install -e .
```

`-e` is short for `--editable` and links the package to the original cloned location such that pulled changes are also reflected in the environment.

To contribute to scVelo, `cd` into the cloned directory and install the latest packages required for development together with the pre-commit hooks:

```
pip install -r requirements-dev.txt
pre-commit install
```

4.2.3 Dependencies

- anndata - annotated data object.
- scanpy - toolkit for single-cell analysis.
- numpy, scipy, pandas, scikit-learn, matplotlib.

Parts of scVelo (directed PAGA and Louvain modularity) require (optional):

```
pip install python-igraph louvain
```

Using fast neighbor search via hnswlib further requires (optional):

```
pip install pybind11 hnswlib
```

4.2.4 Jupyter Notebook

To run the tutorials in a notebook locally, please install:

```
conda install notebook
```

and run `jupyter notebook` in the terminal. If you get the error `Not a directory: 'xdg-settings'`, use `jupyter notebook --no-browser` instead and open the url manually (or use this bugfix).

If you run into issues, do not hesitate to approach us or raise a GitHub issue. scvelo - RNA velocity generalized through dynamical modeling

4.3 API

Import scVelo as:

```
import scvelo as scv
```

After reading the data (`scv.read`) or loading an in-built dataset (`scv.datasets.*`), the typical workflow consists of subsequent calls of preprocessing (`scv.pp.*`), analysis tools (`scv.tl.*`) and plotting (`scv.pl.*`). Further, several utilities (`scv.utils.*`) are provided to facilitate data analysis.

4.3.1 Read / Load

read(filename[, backed, sheet, ext, ...])	Read file and return AnnData object.
read_loom(filename, *[, sparse, cleanup, ...])	Read *.loom*-formatted hdf5 file.

scvelo.read

scvelo.**read**(*filename*, *backed=None*, *sheet=None*, *ext=None*, *delimiter=None*, *first_column_names=False*, *backup_url=None*, *cache=False*, *cache_compression=<Empty.token: 0>*, ***kwargs*)

Read file and return AnnData object.

To speed up reading, consider passing `cache=True`, which creates an hdf5 cache file.

Parameters

filename : Path, str If the filename has no file extension, it is interpreted as a key for generating a filename via `sc.settings.writedir / (filename + sc.settings.file_format_data)`. This is the same behavior as in `sc.read(filename, ...)`.

backed : _Genericalias[r, r+], None If `'r'`, load AnnData in `backed` mode instead of fully loading it into memory (*memory* mode). If you want to modify backed attributes of the AnnData object, you need to choose `'r+'`.

sheet : str, None Name of sheet/table in hdf5 or Excel file.

ext : str, None Extension that indicates the file type. If `None`, uses extension of filename.

delimiter : str, None Delimiter that separates data within text file. If `None`, will split at arbitrary number of white spaces, which is different from enforcing splitting at any single white space `' '`.

first_column_names : bool Assume the first column stores row names. This is only necessary if these are not strings: strings in the first column are automatically assumed to be row names.

backup_url : str, None Retrieve the file from an URL if not present on disk.

cache : bool If *False*, read from source, if *True*, read from fast 'h5ad' cache.

cache_compression : _Genericalias[gzip, lzf], None, Empty See the h5py dataset_compression. (Default: *settings.cache_compression*)

kwargs Parameters passed to `read_loom()`.

Return type `AnnData`

Returns An `AnnData` object

scvelo.read_loom

scvelo.**read_loom**(*filename*, *, *sparse=True*, *cleanup=False*, *X_name='spliced'*, *obs_names='CellID'*, *obsm_names=None*, *var_names='Gene'*, *varm_names=None*, *dtype='float32'*, *obsm_mapping=mappingproxy({})*, *varm_mapping=mappingproxy({})*, ***kwargs*)

Read *.loom*-formatted hdf5 file.

This reads the whole file into memory.

Beware that you have to explicitly state when you want to read the file as sparse data.

Parameters

filename : PathLike The filename.

sparse : bool Whether to read the data matrix as sparse.

cleanup : bool Whether to collapse all obs/var fields that only store one unique value into *.uns['loom-']*.

X_name : str Loompy key with which the data matrix `X` is initialized.

obs_names : str Loompy key where the observation/cell names are stored.

obsm_mapping : Mapping[str, Iterable[str]] Loompy keys which will be constructed into observation matrices

var_names : str Loompy key where the variable/gene names are stored.

varm_mapping : Mapping[str, Iterable[str]] Loompy keys which will be constructed into variable matrices

****kwargs** Arguments to loompy.connect

Example

```
pbmc = anndata.read_loom(
    "pbmc.loom",
    sparse=True,
    X_name="lognorm",
    obs_names="cell_names",
    var_names="gene_names",
    obsm_mapping={
        "X_umap": ["umap_1", "umap_2"]
```

(continues on next page)

(continued from previous page)

```
    }
)
```

Return type AnnData

4.3.2 Preprocessing (pp)

Basic preprocessing (gene selection and normalization)

pp.filter_genes(data[, min_counts, ...])	Filter genes based on number of cells or counts.
pp.filter_genes_dispersion(data[, flavor, ...])	Extract highly variable genes.
pp.normalize_per_cell(data[, ...])	Normalize each cell by total counts over all genes.
pp.log1p(data[, copy])	Logarithmize the data matrix.
pp.filter_and_normalize(data[, min_counts, ...])	Filtering, normalization and log transform

scvelo.pp.filter_genes

scvelo.pp.**filter_genes**(*data*, *min_counts=None*, *min_cells=None*, *max_counts=None*, *max_cells=None*, *min_counts_u=None*, *min_cells_u=None*, *max_counts_u=None*, *max_cells_u=None*, *min_shared_counts=None*, *min_shared_cells=None*, *retain_genes=None*, *copy=False*)

Filter genes based on number of cells or counts. Keep genes that have at least *min_counts* counts or are expressed in at least *min_cells* cells or have at most *max_counts* counts or are expressed in at most *max_cells* cells. Only provide one of the optional parameters *min_counts*, *min_cells*, *max_counts*, *max_cells* per call.

Parameters

data : AnnData, *np.ndarray*, *sp.spmatrix* The (annotated) data matrix of shape *n_obs* × *n_vars*. Rows correspond to cells and columns to genes.

min_counts : *int*, optional (default: *None*) Minimum number of counts required for a gene to pass filtering.

min_cells : *int*, optional (default: *None*) Minimum number of cells expressed required for a gene to pass filtering.

max_counts : *int*, optional (default: *None*) Maximum number of counts required for a gene to pass filtering.

max_cells : *int*, optional (default: *None*) Maximum number of cells expressed required for a gene to pass filtering.

min_counts_u : *int*, optional (default: *None*) Minimum number of unspliced counts required for a gene to pass filtering.

min_cells_u : *int*, optional (default: *None*) Minimum number of unspliced cells expressed required to pass filtering.

max_counts_u : *int*, optional (default: *None*) Maximum number of unspliced counts required for a gene to pass filtering.

max_cells_u : *int*, optional (default: *None*) Maximum number of unspliced cells expressed required to pass filtering.

min_shared_counts : *int*, optional (default: *None*) Minimum number of counts (both unspliced and spliced) required for a gene.

min_shared_cells : *int*, optional (default: *None*) Minimum number of cells required to be expressed (both unspliced and spliced).

retain_genes : *list*, optional (default: *None*) List of gene names to be retained independent of thresholds.

copy : *bool*, optional (default: *False*) Determines whether a copy is returned.

Returns Filters the object and adds *n_counts* to *adata.var*.

scvelo.pp.filter_genes_dispersion

scvelo.pp.**filter_genes_dispersion**(*data*, *flavor='seurat'*, *min_disp=None*, *max_disp=None*, *min_mean=None*, *max_mean=None*, *n_bins=20*, *n_top_genes=None*, *retain_genes=None*, *log=True*, *subset=True*, *copy=False*)

Extract highly variable genes.

Expects non-logarithmized data. The normalized dispersion is obtained by scaling with the mean and standard deviation of the dispersions for genes falling into a given bin for mean expression of genes. This means that for each bin of mean expression, highly variable genes are selected.

Parameters

data : AnnData, *np.ndarray*, *sp.sparse* The (annotated) data matrix of shape *n_obs* × *n_vars*. Rows correspond to cells and columns to genes.

flavor : *{'seurat', 'cell_ranger', 'svr'}, optional (default: 'seurat')* Choose the flavor for computing normalized dispersion. If choosing 'seurat', this expects non-logarithmized data - the logarithm of mean and dispersion is taken internally when *log* is at its default value *True*. For 'cell_ranger', this is usually called for logarithmized data - in this case you should set *log* to *False*. In their default workflows, Seurat passes the cutoffs whereas Cell Ranger passes *n_top_genes*.

min_mean=0.0125 : *float*, optional If *n_top_genes* unequals *None*, these cutoffs for the means and the normalized dispersions are ignored.

max_mean=3 : *float*, optional If *n_top_genes* unequals *None*, these cutoffs for the means and the normalized dispersions are ignored.

min_disp=0.5 : *float*, optional If *n_top_genes* unequals *None*, these cutoffs for the means and the normalized dispersions are ignored.

max_disp=`None` : *float*, optional If *n_top_genes* unequals *None*, these cutoffs for the means and the normalized dispersions are ignored.

n_bins : *int* (default: 20) Number of bins for binning the mean gene expression. Normalization is done with respect to each bin. If just a single gene falls into a bin, the normalized dispersion is artificially set to 1. You'll be informed about this if you set *settings.verbosity = 4*.

n_top_genes : *int* or *None* (default: *None*) Number of highly-variable genes to keep.

retain_genes : *list*, optional (default: *None*) List of gene names to be retained independent of thresholds.

log : *bool*, optional (default: *True*) Use the logarithm of the mean to variance ratio.

subset : *bool*, optional (default: *True*) Keep highly-variable genes only (if True) else write a bool array for highly-variable genes while keeping all genes.

copy : *bool*, optional (default: *False*) If an AnnData is passed, determines whether a copy is returned.

Returns If an AnnData *adata* is passed, returns or updates *adata* depending on *copy*. It filters the *adata* and adds the annotations

scvelo.pp.normalize_per_cell

scvelo.pp.**normalize_per_cell**(*data*, *counts_per_cell_after=None*, *counts_per_cell=None*, *key_n_counts=None*, *max_proportion_per_cell=None*, *use_initial_size=True*, *layers=None*, *enforce=None*, *copy=False*)

Normalize each cell by total counts over all genes.

Parameters

data : AnnData, *np.ndarray*, *sp.sparse* The (annotated) data matrix of shape *n_obs* × *n_vars*. Rows correspond to cells and columns to genes.

counts_per_cell_after : *float* or *None*, optional (default: *None*) If *None*, after normalization, each cell has a total count equal to the median of the *counts_per_cell* before normalization.

counts_per_cell : *np.array*, optional (default: *None*) Precomputed counts per cell.

key_n_counts : *str*, optional (default: *'n_counts'*) Name of the field in *adata.obs* where the total counts per cell are stored.

max_proportion_per_cell : *int* (default: *None*) Exclude genes counts that account for more than a specific proportion of cell size, e.g. 0.05.

use_initial_size : *bool* (default: *True*) Whether to use initial cell sizes oder actual cell sizes.

layers : *str* or *list* (default: *['spliced', 'unspliced']*) Keys for layers to be also considered for normalization.

copy : *bool*, optional (default: *False*) If an AnnData is passed, determines whether a copy is returned.

Returns Returns or updates *adata* with normalized counts.

scvelo.pp.log1p

scvelo.pp.**log1p**(*data*, *copy=False*)

Logarithmize the data matrix. Computes $X = \log(X + 1)$, where *log* denotes the natural logarithm. :param data: Annotated data matrix. :type data: AnnData :param copy: Return a copy of *adata* instead of updating it. :type copy: *bool* (default: *False*)

Returns Returns or updates *adata* depending on *copy*.

scvelo.pp.filter_and_normalize

scvelo.pp.**filter_and_normalize**(*data*, *min_counts=None*, *min_counts_u=None*, *min_cells=None*, *min_cells_u=None*, *min_shared_counts=None*, *min_shared_cells=None*, *n_top_genes=None*, *retain_genes=None*, *subset_highly_variable=True*, *flavor='seurat'*, *log=True*, *layers_normalize=None*, *copy=False*, ***kwargs*)

Filtering, normalization and log transform

Expects non-logarithmized data. If using logarithmized data, pass *log=False*.

Runs the following steps

```
scv.pp.filter_genes(adata)
scv.pp.normalize_per_cell(adata)
if n_top_genes is not None:
    scv.pp.filter_genes_dispersion(adata)
if log:
    scv.pp.log1p(adata)
```

Parameters

data : AnnData Annotated data matrix.

min_counts : *int* (default: *None*) Minimum number of counts required for a gene to pass filtering (spliced).

min_counts_u : *int* (default: *None*) Minimum number of counts required for a gene to pass filtering (unspliced).

min_cells : *int* (default: *None*) Minimum number of cells expressed required to pass filtering (spliced).

min_cells_u : *int* (default: *None*) Minimum number of cells expressed required to pass filtering (unspliced).

min_shared_counts : *int*, optional (default: *None*) Minimum number of counts (both unspliced and spliced) required for a gene.

min_shared_cells : *int*, optional (default: *None*) Minimum number of cells required to be expressed (both unspliced and spliced).

n_top_genes : *int* (default: *None*) Number of genes to keep.

retain_genes : *list*, optional (default: *None*) List of gene names to be retained independent of thresholds.

subset_highly_variable : *bool* (default: True) Whether to subset highly variable genes or to store in .var['highly_variable'].

flavor : *{'seurat', 'cell_ranger', 'svr'}, optional (default: 'seurat')* Choose the flavor for computing normalized dispersion. If choosing 'seurat', this expects non-logarithmized data.

log : *bool* (default: *True*) Take logarithm.

layers_normalize : list of *str* (default: None) List of layers to be normalized. If set to None, the layers {'X', 'spliced', 'unspliced'} are considered for normalization upon testing whether they have already been normalized (by checking type of entries: int -> unprocessed, float -> processed).

copy : *bool* (default: *False*) Return a copy of *adata* instead of updating it.

****kwargs** Keyword arguments passed to pp.normalize_per_cell (e.g. counts_per_cell).

Returns Returns or updates *adata* depending on *copy*.

Moments (across nearest neighbors in PCA space)

pp.pca(data[, n_comps, zero_center, ...])	Principal component analysis [Pedregosa11].
pp.neighbors(adata[, n_neighbors, n_pcs, ...])	Compute a neighborhood graph of observations.
pp.moments(data[, n_neighbors, n_pcs, mode, ...])	Computes moments for velocity estimation.

scvelo.pp.pca

scvelo.pp.**pca**(*data*, *n_comps=None*, *zero_center=True*, *svd_solver='arpack'*, *random_state=0*, *return_info=False*, *use_highly_variable=None*, *dtype='float32'*, *copy=False*, *chunked=False*, *chunk_size=None*)

Principal component analysis [Pedregosa11].

Computes PCA coordinates, loadings and variance decomposition. Uses the implementation of *scikit-learn* [Pedregosa11].

Changed in version 1.5.0: In previous versions, computing a PCA on a sparse matrix would make a dense copy of the array for mean centering. As of scanpy 1.5.0, mean centering is implicit. While results are extremely similar, they are not exactly the same. If you would like to reproduce the old results, pass a dense array.

Parameters

data : AnnData, ndarray, spmatrix The (annotated) data matrix of shape *n_obs* × *n_vars*. Rows correspond to cells and columns to genes.

n_comps : int, None Number of principal components to compute. Defaults to 50, or 1 - minimum dimension size of selected representation.

zero_center : bool, None If *True*, compute standard PCA from covariance matrix. If *False*, omit zero-centering variables (uses `TruncatedSVD`), which allows to handle sparse input efficiently. Passing *None* decides automatically based on sparseness of the data.

svd_solver : str

SVD solver to use:

***'arpack'* (the default)** for the ARPACK wrapper in SciPy (`svds()`)

'randomized' for the randomized algorithm due to Halko (2009).

'auto' chooses automatically depending on the size of the problem.

'lobpcg' An alternative SciPy solver.

Changed in version 1.4.5: Default value changed from *'auto'* to *'arpack'*.

Efficient computation of the principal components of a sparse matrix currently only works with the *'arpack'* or *'lobpcg'* solvers.

random_state : None, int, RandomState Change to use different initial states for the optimization.

return_info : bool Only relevant when not passing an `AnnData`: see "**Returns**".

use_highly_variable : bool, None Whether to use highly variable genes only, stored in *.var['highly_variable']*. By default uses them if they have been determined beforehand.

dtype : str Numpy data type string to which to convert the result.

copy : bool If an AnnData is passed, determines whether a copy is returned. Is ignored otherwise.

chunked : bool If *True*, perform an incremental PCA on segments of *chunk_size*. The incremental PCA automatically zero centers and ignores settings of *random_seed* and *svd_solver*. If *False*, perform a full PCA.

chunk_size : int, None Number of observations to include in each chunk. Required if *chunked=True* was passed.

Return type AnnData, ndarray, spmatrix

Returns

- **X_pca** (spmatrix, ndarray) – If *data* is array-like and *return_info=False* was passed, this function only returns *X_pca*...
- **adata** (*anndata.AnnData*) – ... otherwise if *copy=True* it returns or else adds fields to *adata*:

  .obsm['X_pca'] PCA representation of data.

  .varm['PCs'] The principal components containing the loadings.

  .uns['pca']['variance_ratio'] Ratio of explained variance.

  .uns['pca']['variance'] Explained variance, equivalent to the eigenvalues of the covariance matrix.

scvelo.pp.neighbors

scvelo.pp.**neighbors**(*adata*, *n_neighbors=30*, *n_pcs=None*, *use_rep=None*, *use_highly_variable=True*, *knn=True*, *random_state=0*, *method='umap'*, *metric='euclidean'*, *metric_kwds=None*, *num_threads=- 1*, *copy=False*)

Compute a neighborhood graph of observations.

The neighbor graph methods (umap, hnsw, sklearn) only differ in runtime and yield the same result as scanpy [Wolf18]. Connectivities are computed with adaptive kernel width as proposed in Haghverdi et al. 2016 (doi:10.1038/nmeth.3971).

Parameters

adata Annotated data matrix.

n_neighbors The size of local neighborhood (in terms of number of neighboring data points) used for manifold approximation. Larger values result in more global views of the manifold, while smaller values result in more local data being preserved. In general values should be in the range 2 to 100. If *knn* is *True*, number of nearest neighbors to be searched. If *knn* is *False*, a Gaussian kernel width is set to the distance of the *n_neighbors* neighbor.

n_pcs : *int* or *None* (default: None) Number of principal components to use. If not specified, the full space is used of a pre-computed PCA, or 30 components are used when PCA is computed internally.

use_rep : *None*, *'X'* or any key for *.obsm* (default: None) Use the indicated representation. If *None*, the representation is chosen automatically: for .n_vars < 50, .X is used, otherwise 'X_pca' is used.

use_highly_variable : *bool* (default: True) Whether to use highly variable genes only, stored in .var['highly_variable'].

knn If *True*, use a hard threshold to restrict the number of neighbors to *n_neighbors*, that is, consider a knn graph. Otherwise, use a Gaussian Kernel to assign low weights to neighbors more distant than the *n_neighbors* nearest neighbor.

random_state A numpy random seed.

method : {{'umap', 'hnsw', 'sklearn'}} (default: *'umap'*) Method to compute neighbors, only differs in runtime. The 'hnsw' method is most efficient and requires to *pip install hnswlib*. Connectivities are computed with adaptive kernel.

metric A known metric's name or a callable that returns a distance.

metric_kwds Options for the metric.

num_threads Number of threads to be used (for runtime).

copy Return a copy instead of writing to adata.

Returns

- **connectivities** (*.obsp*) – Sparse weighted adjacency matrix of the neighborhood graph of data points. Weights should be interpreted as connectivities.
- **distances** (*.obsp*) – Sparse matrix of distances for each pair of neighbors.

scvelo.pp.moments

scvelo.pp.**moments**(*data*, *n_neighbors=30*, *n_pcs=None*, *mode='connectivities'*, *method='umap'*, *use_rep=None*, *use_highly_variable=True*, *copy=False*)

Computes moments for velocity estimation.

First-/second-order moments are computed for each cell across its nearest neighbors, where the neighbor graph is obtained from euclidean distances in PCA space.

Parameters

data : AnnData Annotated data matrix.

n_neighbors : *int* (default: 30) Number of neighbors to use.

n_pcs : *int* (default: None) Number of principal components to use. If not specified, the full space is used of a pre-computed PCA, or 30 components are used when PCA is computed internally.

mode : *'connectivities'* or *'distances'* (default: *'connectivities'*) Distance metric to use for moment computation.

method : {{'umap', 'hnsw', 'sklearn', *None*}} (default: *'umap'*) Method to compute neighbors, only differs in runtime. Connectivities are computed with adaptive kernel width as proposed in Haghverdi et al. 2016 (https://doi.org/10.1038/nmeth.3971).

use_rep : *None*, *'X'* or any key for *.obsm* (default: None) Use the indicated representation. If *None*, the representation is chosen automatically: for .n_vars < 50, .X is used, otherwise 'X_pca' is used.

use_highly_variable : *bool* (default: True) Whether to use highly variable genes only, stored in .var['highly_variable'].

copy : *bool* (default: *False*) Return a copy instead of writing to adata.

Returns

- **Ms** (*.layers*) – dense matrix with first order moments of spliced counts.

- **Mu** (*.layers*) – dense matrix with first order moments of unspliced counts.

4.3.3 Tools (tl)

Clustering and embedding (more at scanpy-docs)

tl.louvain(adata[, resolution, ...])	Cluster cells into subgroups [Blondel08] [Levine15] [Traag17].
tl.umap(adata[, min_dist, spread, ...])	Embed the neighborhood graph using UMAP [McInnes18].

scvelo.tl.louvain

scvelo.tl.**louvain**(*adata*, *resolution=None*, *random_state=0*, *restrict_to=None*, *key_added='louvain'*, *adjacency=None*, *flavor='vtraag'*, *directed=True*, *use_weights=False*, *partition_type=None*, *partition_kwargs=mappingproxy({})*, *neighbors_key=None*, *obsp=None*, *copy=False*)

Cluster cells into subgroups [Blondel08] [Levine15] [Traag17].

Cluster cells using the Louvain algorithm [Blondel08] in the implementation of [Traag17]. The Louvain algorithm has been proposed for single-cell analysis by [Levine15].

This requires having ran `neighbors()` or `bbknn()` first, or explicitly passing a `adjacency` matrix.

Parameters

adata : AnnData The annotated data matrix.

resolution : float, None For the default flavor (`'vtraag'`), you can provide a resolution (higher resolution means finding more and smaller clusters), which defaults to 1.0. See "Time as a resolution parameter" in [Lambiotte09].

random_state : None, int, RandomState Change the initialization of the optimization.

restrict_to : Tuple[str, Sequence[str]], None Restrict the clustering to the categories within the key for sample annotation, tuple needs to contain `(obs_key, list_of_categories)`.

key_added : str Key under which to add the cluster labels. (default: `'louvain'`)

adjacency : spmatrix, None Sparse adjacency matrix of the graph, defaults to neighbors connectivities.

flavor : _Genericalias[vtraag, igraph, rapids] Choose between to packages for computing the clustering. `'vtraag'` is much more powerful, and the default.

directed : bool Interpret the `adjacency` matrix as directed graph?

use_weights : bool Use weights from knn graph.

partition_type : Type[MutableVertexPartition], None Type of partition to use. Only a valid argument if `flavor` is `'vtraag'`.

partition_kwargs : Mapping[str, Any] Key word arguments to pass to partitioning, if `vtraag` method is being used.

neighbors_key : str, None Use neighbors connectivities as adjacency. If not specified, louvain looks .obsp['connectivities'] for connectivities (default storage place for

pp.neighbors). If specified, louvain looks .obsp[.uns[neighbors_key]['connectivities_key']] for connectivities.

obsp : str, None Use .obsp[obsp] as adjacency. You can't specify both *obsp* and *neighbors_key* at the same time.

copy : bool Copy adata or modify it inplace.

Return type AnnData, None

Returns

- None – By default (copy=False), updates adata with the following fields:

  adata.obs['louvain'] (pandas.Series, dtype category) Array of dim (number of samples) that stores the subgroup id ('0', '1', ...) for each cell.

- AnnData – When copy=True is set, a copy of adata with those fields is returned.

scvelo.tl.umap

scvelo.tl.**umap**(*adata*, *min_dist=0.5*, *spread=1.0*, *n_components=2*, *maxiter=None*, *alpha=1.0*, *gamma=1.0*, *negative_sample_rate=5*, *init_pos='spectral'*, *random_state=0*, *a=None*, *b=None*, *copy=False*, *method='umap'*, *neighbors_key=None*)

Embed the neighborhood graph using UMAP [McInnes18].

UMAP (Uniform Manifold Approximation and Projection) is a manifold learning technique suitable for visualizing high-dimensional data. Besides tending to be faster than tSNE, it optimizes the embedding such that it best reflects the topology of the data, which we represent throughout Scanpy using a neighborhood graph. tSNE, by contrast, optimizes the distribution of nearest-neighbor distances in the embedding such that these best match the distribution of distances in the high-dimensional space. We use the implementation of umap-learn [McInnes18]. For a few comparisons of UMAP with tSNE, see this preprint.

Parameters

adata : AnnData Annotated data matrix.

min_dist : float The effective minimum distance between embedded points. Smaller values will result in a more clustered/clumped embedding where nearby points on the manifold are drawn closer together, while larger values will result on a more even dispersal of points. The value should be set relative to the spread value, which determines the scale at which embedded points will be spread out. The default of in the *umap-learn* package is 0.1.

spread : float The effective scale of embedded points. In combination with *min_dist* this determines how clustered/clumped the embedded points are.

n_components : int The number of dimensions of the embedding.

maxiter : int, None The number of iterations (epochs) of the optimization. Called *n_epochs* in the original UMAP.

alpha : float The initial learning rate for the embedding optimization.

gamma : float Weighting applied to negative samples in low dimensional embedding optimization. Values higher than one will result in greater weight being given to negative samples.

negative_sample_rate : int The number of negative edge/1-simplex samples to use per positive edge/1-simplex sample in optimizing the low dimensional embedding.

init_pos : _Genericalias[paga, spectral, random], ndarray, None

How to initialize the low dimensional embedding. Called *init* in the original UMAP. Options are:

- Any key for *adata.obsm*.
- 'paga': positions from `paga()`.
- 'spectral': use a spectral embedding of the graph.
- 'random': assign initial embedding positions at random.
- A numpy array of initial embedding positions.

random_state : None, int, RandomState If *int*, *random_state* is the seed used by the random number generator; If *RandomState* or *Generator*, *random_state* is the random number generator; If *None*, the random number generator is the *RandomState* instance used by *np.random*.

a : float, None More specific parameters controlling the embedding. If *None* these values are set automatically as determined by *min_dist* and *spread*.

b : float, None More specific parameters controlling the embedding. If *None* these values are set automatically as determined by *min_dist* and *spread*.

copy : bool Return a copy instead of writing to adata.

method : _Genericalias[umap, rapids] Use the original 'umap' implementation, or 'rapids' (experimental, GPU only)

neighbors_key : str, None If not specified, umap looks .uns['neighbors'] for neighbors settings and .obsp['connectivities'] for connectivities (default storage places for pp.neighbors). If specified, umap looks .uns[neighbors_key] for neighbors settings and .obsp[.uns[neighbors_key]['connectivities_key']] for connectivities.

Return type AnnData, None

Returns

- Depending on *copy*, returns or updates *adata* with the following fields.
- ****X_umap**** (*adata.obsm* field) – UMAP coordinates of data.

Velocity estimation

tl.velocity(data[, vkey, mode, fit_offset, ...])	Estimates velocities in a gene-specific manner.
tl.velocity_graph(data[, vkey, xkey, tkey, ...])	Computes velocity graph based on cosine similarities.
tl.velocity_embedding(data[, basis, vkey, ...])	Projects the single cell velocities into any embedding.

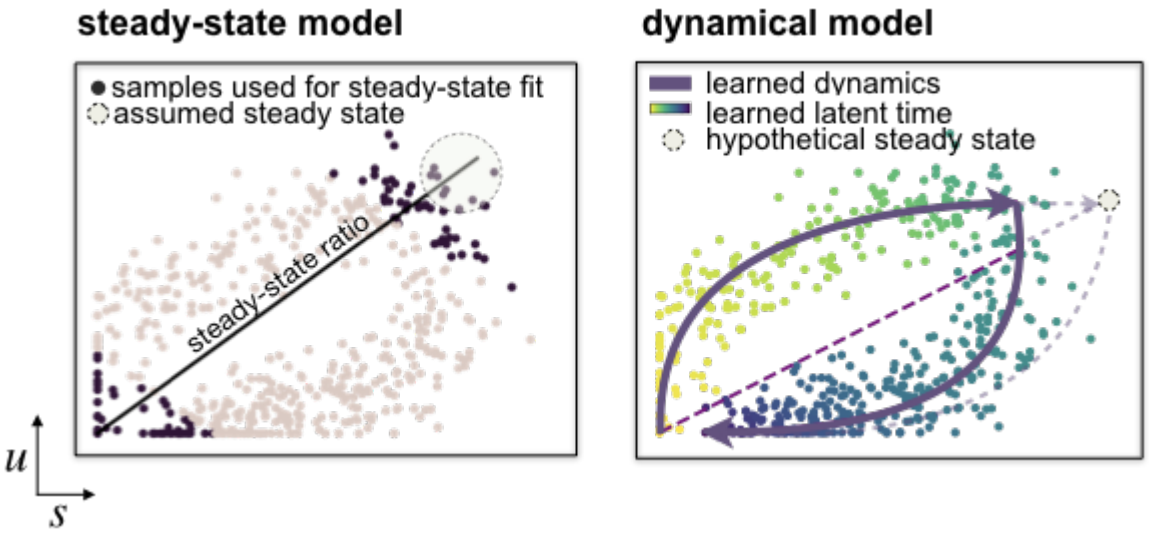
scvelo.tl.velocity

scvelo.tl.**velocity**(*data*, *vkey='velocity'*, *mode='stochastic'*, *fit_offset=False*, *fit_offset2=False*, *filter_genes=False*, *groups=None*, *groupby=None*, *groups_for_fit=None*, *constrain_ratio=None*, *use_raw=False*, *use_latent_time=None*, *perc=[5, 95]*, *min_r2=0.01*, *min_likelihood=0.001*, *r2_adjusted=None*, *use_highly_variable=True*, *diff_kinetics=None*, *copy=False*, ***kwargs*)

Estimates velocities in a gene-specific manner.

The steady-state model [Manno18] determines velocities by quantifying how observations deviate from a presumed steady-state equilibrium ratio of unspliced to spliced mRNA levels. This steady-state ratio is obtained by performing a linear regression restricting the input data to the extreme quantiles. By including second-order moments, the stochastic model [Bergen19] exploits not only the balance of unspliced to spliced mRNA lev-

els but also their covariation. By contrast, the likelihood-based dynamical model [Bergen19] solves the full splicing kinetics and generalizes RNA velocity estimation to transient systems. It is also capable of capturing non-observed steady states.



Parameters

data : AnnData Annotated data matrix.

vkey : *str* (default: *'velocity'*) Name under which to refer to the computed velocities for *velocity_graph* and *velocity_embedding*.

mode : *'deterministic'*, *'stochastic'* or *'dynamical'* (default: *'stochastic'*) Whether to run the estimation using the steady-state/deterministic, stochastic or dynamical model of transcriptional dynamics. The dynamical model requires to run *tl.recover_dynamics* first.

fit_offset : *bool* (default: *False*) Whether to fit with offset for first order moment dynamics.

fit_offset2 : *bool*, (default: *False*) Whether to fit with offset for second order moment dynamics.

filter_genes : *bool* (default: *True*) Whether to remove genes that are not used for further velocity analysis.

groups : *str*, *list* (default: *None*) Subset of groups, e.g. ['g1', 'g2', 'g3'], to which velocity analysis shall be restricted.

groupby : *str*, *list* or *np.ndarray* (default: *None*) Key of observations grouping to consider.

groups_for_fit : *str*, *list* or *np.ndarray* (default: *None*) Subset of groups, e.g. ['g1', 'g2', 'g3'], to which steady-state fitting shall be restricted.

constrain_ratio : *float* or tuple of type *float* or None: (default: *None*) Bounds for the steady-state ratio.

use_raw : *bool* (default: *False*) Whether to use raw data for estimation.

use_latent_time : *bool`or `None* (default: *None*) Whether to use latent time as a regularization for velocity estimation.

perc : *float* (default: *[5, 95]*) Percentile, e.g. 98, for extreme quantile fit.

min_r2 : *float* (default: 0.01) Minimum threshold for coefficient of determination

min_likelihood : *float* (default: *None*) Minimal likelihood for velocity genes to fit the model on.

r2_adjusted : *bool* (default: *None*) Whether to compute coefficient of determination on full data fit (adjusted) or extreme quantile fit (None)

use_highly_variable : *bool* (default: True) Whether to use highly variable genes only, stored in .var['highly_variable'].

copy : *bool* (default: *False*) Return a copy instead of writing to *adata*.

Returns

- **velocity** (*.layers*) – velocity vectors for each individual cell
- **velocity_genes, velocity_beta, velocity_gamma, velocity_r2** (*.var*) – parameters

scvelo.tl.velocity_graph

scvelo.tl.**velocity_graph**(*data*, *vkey='velocity'*, *xkey='Ms'*, *tkey=None*, *basis=None*, *n_neighbors=None*, *n_recurse_neighbors=None*, *random_neighbors_at_max=None*, *sqrt_transform=None*, *variance_stabilization=None*, *gene_subset=None*, *compute_uncertainties=None*, *approx=None*, *mode_neighbors='distances'*, *copy=False*, *n_jobs=None*, *backend='loky'*)

Computes velocity graph based on cosine similarities.

The cosine similarities are computed between velocities and potential cell state transitions, i.e. it measures how well a corresponding change in gene expression $\delta_{ij} = x_j - x_i$ matches the predicted change according to the velocity vector ν_i,

$$\pi_{ij} = \cos \angle(\delta_{ij}, \nu_i) = \frac{\delta_{ij}^T \nu_i}{\|\delta_{ij}\| \, \|\nu_i\|}.$$

Parameters

data : AnnData Annotated data matrix.

vkey : *str* (default: *'velocity'*) Name of velocity estimates to be used.

xkey : *str* (default: *'Ms'*) Layer key to extract count data from.

tkey : *str* (default: *None*) Observation key to extract time data from.

basis : *str* (default: *None*) Basis / Embedding to use.

n_neighbors : *int* or *None* (default: None) Use fixed number of neighbors or do recursive neighbor search (if *None*).

n_recurse_neighbors : *int* (default: *None*) Number of recursions for neighbors search. Defaults to 2 if mode_neighbors is 'distances', and 1 if mode_neighbors is 'connectivities'.

random_neighbors_at_max : *int* or *None* (default: *None*) If number of iterative neighbors for an individual cell is higher than this threshold, a random selection of such are chosen as reference neighbors.

sqrt_transform : *bool* (default: *False*) Whether to variance-transform the cell states changes and velocities before computing cosine similarities.

gene_subset : *list* of *str*, subset of adata.var_names or *None`(default: `None*) Subset of genes to compute velocity graph on exclusively.

compute_uncertainties : *bool* (default: *None*) Whether to compute uncertainties along with cosine correlation.

approx : *bool* or *None* (default: *None*) If True, first 30 pc's are used instead of the full count matrix

mode_neighbors : 'str' (default: *'distances'*) Determines the type of KNN graph used. Options are 'distances' or 'connectivities'. The latter yields a symmetric graph.

copy : *bool* (default: *False*) Return a copy instead of writing to adata.

n_jobs : *int* or *None* (default: *None*) Number of parallel jobs.

backend : *str* (default: "loky") Backend used for multiprocessing. See `joblib.Parallel` for valid options.

Returns **velocity_graph** (*.uns*) – sparse matrix with correlations of cell state transitions with velocities

scvelo.tl.velocity_embedding

scvelo.tl.**velocity_embedding**(*data*, *basis=None*, *vkey='velocity'*, *scale=10*, *self_transitions=True*, *use_negative_cosines=True*, *direct_pca_projection=None*, *retain_scale=False*, *autoscale=True*, *all_comps=True*, *T=None*, *copy=False*)

Projects the single cell velocities into any embedding.

Given normalized difference of the embedding positions :math: *tilde delta_{ij} = frac{x_j-x_i}{leftlVert x_j-x_i rightrVert}*. the projections are obtained as expected displacements with respect to the transition matrix $\tilde{\pi}_{ij}$ as

$$\tilde{\nu}_i = E_{\tilde{\pi}_{i\cdot}}[\tilde{\delta}_{i\cdot}] = \sum_{j \neq i} \left(\tilde{\pi}_{ij} - \frac{1}{n} \right) \tilde{} delta_{ij}.$$

Parameters

data : AnnData Annotated data matrix.

basis : *str* (default: *'tsne'*) Which embedding to use.

vkey : *str* (default: *'velocity'*) Name of velocity estimates to be used.

scale : *int* (default: 10) Scale parameter of gaussian kernel for transition matrix.

self_transitions : *bool* (default: *True*) Whether to allow self transitions, based on the confidences of transitioning to neighboring cells.

use_negative_cosines : *bool* (default: *True*) Whether to project cell-to-cell transitions with negative cosines into negative/opposite direction.

direct_pca_projection : *bool* (default: *None*) Whether to directly project the velocities into PCA space, thus skipping the velocity graph.

retain_scale : *bool* (default: *False*) Whether to retain scale from high dimensional space in embedding.

autoscale : *bool* (default: *True*) Whether to scale the embedded velocities by a scalar multiplier, which simply ensures that the arrows in the embedding are properly scaled.

all_comps : *bool* (default: *True*) Whether to compute the velocities on all embedding components.

T : *csr_matrix* (default: *None*) Allows the user to directly pass a transition matrix.

copy : *bool* (default: *False*) Return a copy instead of writing to *adata*.

Returns velocity_umap (*.obsm*) – coordinates of velocity projection on embedding (e.g., basis='umap')

Dynamical modeling

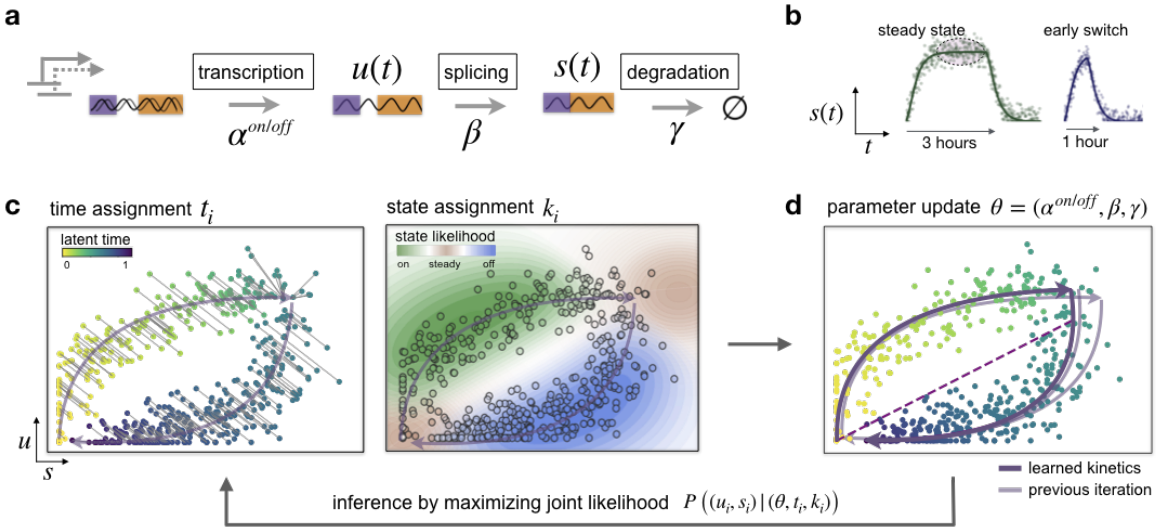
tl.recover_dynamics(data[, var_names, ...])	Recovers the full splicing kinetics of specified genes.
tl.differential_kinetic_test(data[, ...])	Test to detect cell types / lineages with different kinetics.

scvelo.tl.recover_dynamics

scvelo.tl.**recover_dynamics**(*data*, *var_names='velocity_genes'*, *n_top_genes=None*, *max_iter=10*, *assignment_mode='projection'*, *t_max=None*, *fit_time=True*, *fit_scaling=True*, *fit_steady_states=True*, *fit_connected_states=None*, *fit_basal_transcription=None*, *use_raw=False*, *load_pars=None*, *return_model=None*, *plot_results=False*, *steady_state_prior=None*, *add_key='fit'*, *copy=False*, *n_jobs=None*, *backend='loky'*, ***kwargs*)

Recovers the full splicing kinetics of specified genes.

The model infers transcription rates, splicing rates, degradation rates, as well as cell-specific latent time and transcriptional states, estimated iteratively by expectation-maximization.



Parameters

data : AnnData Annotated data matrix.

var_names : *str*, list of *str* (default: *'velocity_genes'*) Names of variables/genes to use for the fitting. If *var_names='velocity_genes'* but there is no column *'velocity_genes'* in *adata.var*, velocity genes are estimated using the steady state model.

n_top_genes : *int* or *None* (default: *None*) Number of top velocity genes to use for the dynamical model.

max_iter : *int* (default: *10*) Maximal iterations in the EM-Algorithm.

assignment_mode : *str* (default: *projection*) Determined how times are assigned to observations. If *projection*, observations are projected onto the model trajectory. Else uses an inverse approximating formula.

t_max : *float*, *False* or *None* (default: *None*) Total range for time assignments.

fit_scaling : *bool* or *float* or *None* (default: *True*) Whether to fit scaling between unspliced and spliced.

fit_time : *bool* or *float* or *None* (default: *True*) Whether to fit time or keep initially given time fixed.

fit_steady_states : *bool* or *None* (default: *True*) Whether to explicitly model and fit steady states (next to induction/repression)

fit_connected_states : *bool* or *None* (default: *None*) Restricts fitting to neighbors given by connectivities.

fit_basal_transcription : *bool* or *None* (default: *None*) Enables model to incorporate basal transcriptions.

use_raw : *bool* or *None* (default: *None*) if True, use .layers['sliced'], else use moments from .layers['Ms']

load_pars : *bool* or *None* (default: *None*) Load parameters from past fits.

return_model : *bool* or *None* (default: *True*) Whether to return the model as :DynamicsRecovery: object.

plot_results : *bool* or *None* (default: *False*) Plot results after parameter inference.

steady_state_prior : list of *bool* or *None* (default: *None*) Mask for indices used for steady state regression.

add_key : *str* (default: *'fit'*) Key to add to parameter names, e.g. 'fit_t' for fitted time.

copy : *bool* (default: *False*) Return a copy instead of writing to *adata*.

n_jobs : *int* or *None* (default: *None*) Number of parallel jobs.

backend : *str* (default: "loky") Backend used for multiprocessing. See `joblib.Parallel` for valid options.

Returns

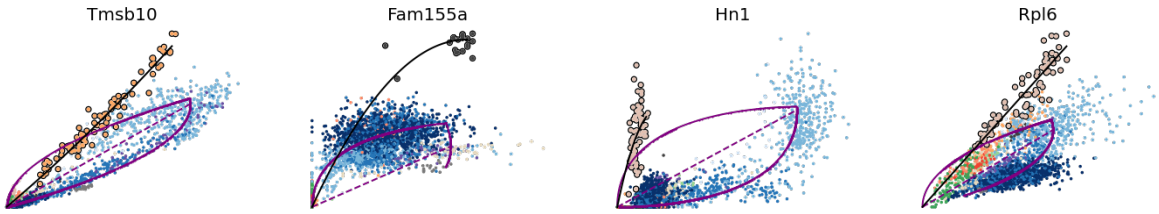
- **fit_alpha** (*.var*) – inferred transcription rates
- **fit_beta** (*.var*) – inferred splicing rates
- **fit_gamma** (*.var*) – inferred degradation rates
- **fit_t_** (*.var*) – inferred switching time points
- **fit_scaling** (*.var*) – internal variance scaling factor for un/spliced counts
- **fit_likelihood** (*.var*) – likelihood of model fit
- **fit_alignment_scaling** (*.var*) – scaling factor to align gene-wise latent times to a universal latent time

scvelo.tl.differential_kinetic_test

scvelo.tl.**differential_kinetic_test**(*data*, *var_names='velocity_genes'*, *groupby=None*, *use_raw=None*, *return_model=None*, *add_key='fit'*, *copy=None*, ***kwargs*)

Test to detect cell types / lineages with different kinetics.

Likelihood ratio test for differential kinetics to detect clusters/lineages that display kinetic behavior that cannot be sufficiently explained by a single model for the overall dynamics. Each cell type is tested whether an independent fit yields a significantly improved likelihood.



Parameters

data : AnnData Annotated data matrix.

var_names : *str*, list of *str* (default: *'velocity_genes'*) Names of variables/genes to use for the fitting.

groupby : *str* (default: *None*) Key of observations grouping to consider, e.g. *'clusters'*.

use_raw : *bool* (default: *False*) Whether to use raw data for estimation.

add_key : *str* (default: *'fit'*) Key to add to parameter names, e.g. 'fit_t' for fitted time.

copy : *bool* (default: *None*) Return a copy instead of writing to *adata*.

Returns

- **fit_pvals_kinetics** (*.varm*) – P-values of competing kinetic for each group and gene
- **fit_diff_kinetics** (*.var*) – Groups that have differential kinetics for each gene.

Dynamical genes

tl.rank_velocity_genes(data[, vkey, ...])	Rank genes for velocity characterizing groups.
tl.rank_dynamical_genes(data[, n_genes, ...])	Rank genes by likelihoods per cluster/regime.

scvelo.tl.rank_velocity_genes

scvelo.tl.**rank_velocity_genes**(*data*, *vkey='velocity'*, *n_genes=100*, *groupby=None*, *match_with=None*, *resolution=None*, *min_counts=None*, *min_r2=None*, *min_corr=None*, *min_dispersion=None*, *min_likelihood=None*, *copy=False*)

Rank genes for velocity characterizing groups.

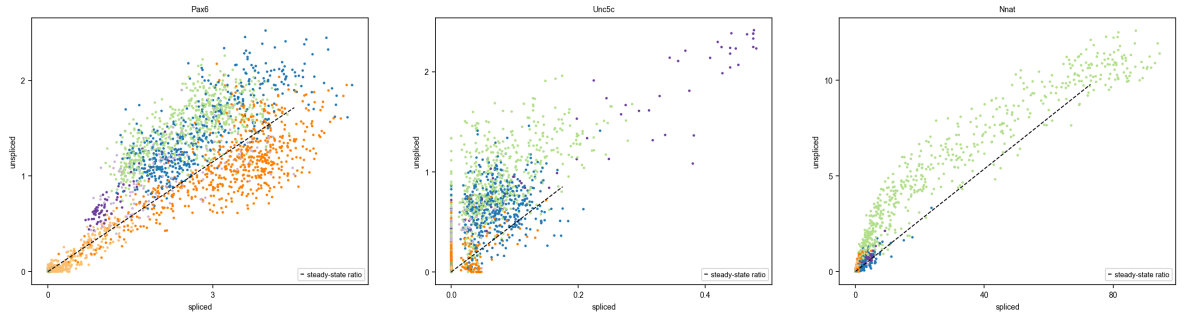
This applies a differential expression test (Welch t-test with overestimated variance to be conservative) on velocity expression, to find genes in a cluster that show dynamics that is transcriptionally regulated differently compared to all other clusters (e.g. induction in that cluster and homeostasis in remaining population). If no clusters are given, it priorly computes velocity clusters by applying louvain modularity on velocity expression.

```
scv.tl.rank_velocity_genes(adata, groupby='clusters')
```

(continues on next page)

(continued from previous page)

```
scv.pl.scatter(
    adata, basis=adata.uns['rank_velocity_genes']['names']['Beta'][:3]
)
pd.DataFrame(adata.uns['rank_velocity_genes']['names']).head()
```



	Ductal	Ngn3 low EP	Ngn3 high EP	Pre-endocrine	Beta	Alpha	Delta	Epsilon
0	Notch2	Hacd1	Pde1c	Pam	Pax6	Nlgn1	Zdbf2	Heg1
1	Sox5	Hspa8	Pclo	Baiap3	Unc5c	Zcchc16	Akr1c19	Tmcc3
2	Krt19	Fgf12	Ptprs	Sdk1	Nnat	Prune2	Ank2	Gpr179
3	Hspa8	Trp53cor1	Rap1gap2	Abcc8	Tmem108	Ndst4	Cpm	Ncoa7
4	Nr2f6	Stard10	Ttyh2	Gnas	Ptprt	Ksr2	Ptprt	Ica1

Parameters

data : AnnData Annotated data matrix.

vkey : *str* (default: *'velocity'*) Key of velocities computed in *tl.velocity*

n_genes : *int*, optional (default: 100) The number of genes that appear in the returned tables.

groupby : *str*, *list* or *np.ndarray* (default: *None*) Key of observations grouping to consider.

match_with : *str* or *None* (default: *None*) adata.obs key to separatively rank velocities on.

resolution : *str* or *None* (default: *None*) Resolution for louvain modularity.

min_counts : *float* (default: None) Minimum count of genes for consideration.

min_r2 : *float* (default: None) Minimum r2 value of genes for consideration.

min_corr : *float* (default: None) Minimum Spearmans correlation coefficient between spliced and unspliced.

min_dispersion : *float* (default: None) Minimum dispersion norm value of genes for consideration.

min_likelihood : *float* between *0* and *1* or *None* (default: *None*) Only rank velocity of genes with a likelihood higher than min_likelihood.

copy : *bool* (default: *False*) Return a copy instead of writing to data.

Returns

- **rank_velocity_genes** (*.uns*) – Structured array to be indexed by group id storing the gene names. Ordered according to scores.

- **velocity_score** (*.var*) – Storing the score for each gene for each group. Ordered according to scores.

scvelo.tl.rank_dynamical_genes

scvelo.tl.**rank_dynamical_genes**(*data*, *n_genes=100*, *groupby=None*, *copy=False*)

Rank genes by likelihoods per cluster/regime.

This ranks genes by their likelihood obtained from the dynamical model grouped by clusters specified in groupby.

Parameters

data : AnnData Annotated data matrix.

n_genes : *int*, optional (default: 100) The number of genes that appear in the returned tables.

groupby : *str*, *list* or *np.ndarray* (default: *None*) Key of observations grouping to consider.

copy : *bool* (default: *False*) Return a copy instead of writing to data.

Returns **rank_dynamical_genes** (*.uns*) – Structured array to be indexed by group id storing the gene names. Ordered according to scores.

Pseudotime and trajectory inference

tl.terminal_states(data[, vkey, modality, ...])	Computes terminal states (root and end points).
tl.velocity_pseudotime(adata[, vkey, ...])	Computes a pseudotime based on the velocity graph.
tl.latent_time(data[, vkey, min_likelihood, ...])	Computes a gene-shared latent time.
tl.paga(adata[, groups, vkey, ...])	PAGA graph with velocity-directed edges.

scvelo.tl.terminal_states

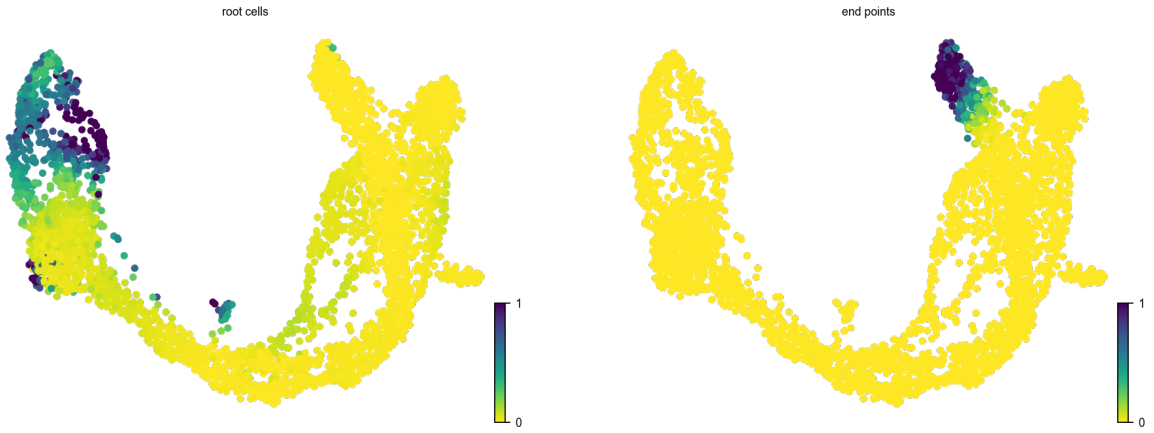
scvelo.tl.**terminal_states**(*data*, *vkey='velocity'*, *modality='Ms'*, *groupby=None*, *groups=None*, *self_transitions=False*, *eps=0.001*, *random_state=0*, *copy=False*, ***kwargs*)

Computes terminal states (root and end points).

The end points and root cells are obtained as stationary states of the velocity-inferred transition matrix and its transposed, respectively, which is given by left eigenvectors corresponding to an eigenvalue of 1, i.e.

$$^{\text{end}} = ^{\text{end}}\pi, \quad ^{\text{root}} = ^{\text{root}}\pi^{\mathrm{T}}.$$

```
scv.tl.terminal_states(adata)
scv.pl.scatter(adata, color=['root_cells', 'end_points'])
```



Alternatively, we recommend to use `cellrank.tl.terminal_states()` providing an improved/generalized approach of identifying terminal states.

Parameters

`data` : `AnnData` Annotated data matrix.

`vkey` : *str* (default: *'velocity'*) Name of velocity estimates to be used.

`modality` : *str* (default: *'Ms'*) Layer used to calculate terminal states.

`groupby` : *str*, *list* or *np.ndarray* (default: *None*) Key of observations grouping to consider. Only to be set, if each group is assumed to have a distinct lineage with an independent root and end point.

`groups` : *str*, *list* or *np.ndarray* (default: *None*) Groups selected to find terminal states on. Must be an element of .obs[groupby]. To be specified only for very distinct/disconnected clusters.

`self_transitions` : *bool* (default: *False*) Allow transitions from one node to itself.

`eps` : *float* (default: 1e-3) Tolerance for eigenvalue selection.

`random_state` : *int* or None (default: 0) Seed used by the random number generator. If *None*, use the *RandomState* instance by *np.random*.

`copy` : *bool* (default: *False*) Return a copy instead of writing to data.

`**kwargs` Passed to scvelo.tl.transition_matrix(), e.g. basis, weight_diffusion.

Returns

- **root_cells** (*.obs*) – sparse matrix with transition probabilities.
- **end_points** (*.obs*) – sparse matrix with transition probabilities.

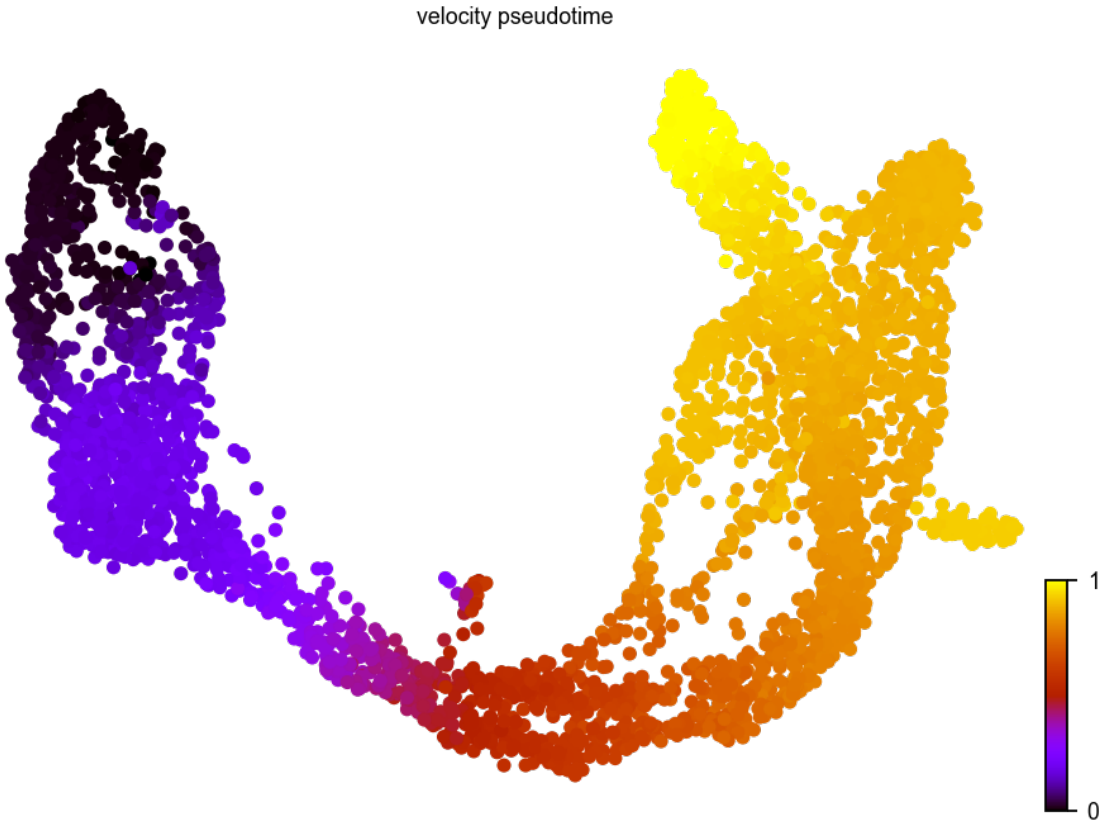
scvelo.tl.velocity_pseudotime

scvelo.tl.**velocity_pseudotime**(*adata*, *vkey='velocity'*, *groupby=None*, *groups=None*, *root_key=None*, *end_key=None*, *n_dcs=10*, *use_velocity_graph=True*, *save_diffmap=None*, *return_model=None*, ***kwargs*)

Computes a pseudotime based on the velocity graph.

Velocity pseudotime is a random-walk based distance measures on the velocity graph. After computing a distribution over root cells obtained from the velocity-inferred transition matrix, it measures the average number of steps it takes to reach a cell after start walking from one of the root cells. Contrarily to diffusion pseudotime, it implicitly infers the root cells and is based on the directed velocity graph instead of the similarity-based diffusion kernel.

```
scv.tl.velocity_pseudotime(adata)
scv.pl.scatter(adata, color='velocity_pseudotime', color_map='gnuplot')
```



Parameters

adata : AnnData Annotated data matrix

vkey : *str* (default: *'velocity'*) Name of velocity estimates to be used.

groupby : *str*, *list* or *np.ndarray* (default: *None*) Key of observations grouping to consider.

groups : *str*, *list* or *np.ndarray* (default: *None*) Groups selected to find terminal states on. Must be an element of adata.obs[groupby]. Only to be set, if each group is assumed to have a distinct lineage with an independent root and end point.

root_key : *int* (default: *None*) Index of root cell to be used. Computed from velocity-inferred transition matrix if not specified.

end_key : *int* (default: *None*) Index of end point to be used. Computed from velocity-inferred transition matrix if not specified.

n_dcs : *int* (default: 10) The number of diffusion components to use.

use_velocity_graph : *bool* (default: *True*) Whether to use the velocity graph. If False, it uses the similarity-based diffusion kernel.

save_diffmap : *bool* (default: *None*) Whether to store diffmap coordinates.

return_model : *bool* (default: *None*) Whether to return the vpt object for further inspection.

****kwargs** Further arguments to pass to VPT (e.g. min_group_size, allow_kendall_tau_shift).

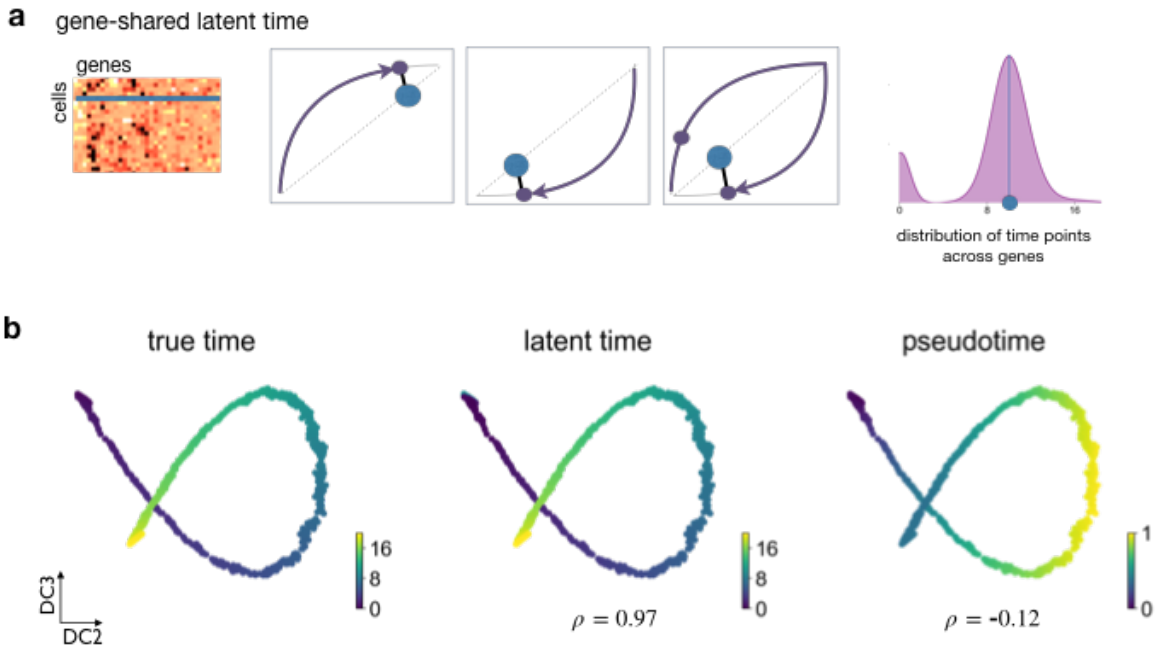
Returns **velocity_pseudotime** (*.obs*) – Velocity pseudotime obtained from velocity graph.

scvelo.tl.latent_time

scvelo.tl.**latent_time**(*data*, *vkey='velocity'*, *min_likelihood=0.1*, *min_confidence=0.75*, *min_corr_diffusion=None*, *weight_diffusion=None*, *root_key=None*, *end_key=None*, *t_max=None*, *copy=False*)

Computes a gene-shared latent time.

Gene-specific latent timepoints obtained from the dynamical model are coupled to a universal gene-shared latent time, which represents the cell's internal clock and is based only on its transcriptional dynamics.



Parameters

data : AnnData Annotated data matrix

vkey : *str* (default: *'velocity'*) Name of velocity estimates to be used.

min_likelihood : *float* between *0* and *1* or *None* (default: *.1*) Minimal likelihood fitness for genes to be included to the weighting.

min_confidence : *float* between *0* and *1* (default: *.75*) Parameter for local coherence selection.

min_corr_diffusion : *float* between *0* and *1* or *None* (default: *None*) Only select genes that correlate with velocity pseudotime obtained from diffusion random walk on velocity graph.

weight_diffusion : *float* or *None* (default: *None*) Weight applied to couple latent time with diffusion-based velocity pseudotime.

root_key : *str* or *None* (default: *'root_cells'*) Key (.uns, .obs) of root cell to be used. If not set, it obtains root cells from velocity-inferred transition matrix.

end_key : *str* or *None* (default: *None*) Key (.obs) of end points to be used.

t_max : *float* or *None* (default: *None*) Overall duration of differentiation process. If not set, a overall transcriptional timescale of 20 hours is used as prior.

copy : *bool* (default: *False*) Return a copy instead of writing to *adata*.

Returns **latent_time** (*.obs*) – latent time from learned dynamics for each cell

scvelo.tl.paga

scvelo.tl.**paga**(*adata*, *groups=None*, *vkey='velocity'*, *use_time_prior=True*, *root_key=None*, *end_key=None*, *threshold_root_end_prior=None*, *minimum_spanning_tree=True*, *copy=False*)

PAGA graph with velocity-directed edges.

Mapping out the coarse-grained connectivity structures of complex manifolds [Wolf19]. By quantifying the connectivity of partitions (groups, clusters) of the single-cell graph, partition-based graph abstraction (PAGA) generates a much simpler abstracted graph (*PAGA graph*) of partitions, in which edge weights represent confidence in the presence of connections.

Parameters

adata : AnnData An annotated data matrix.

groups : key for categorical in *adata.obs*, optional (default: 'louvain') You can pass your predefined groups by choosing any categorical annotation of observations (*adata.obs*).

vkey : *str* or *None* (default: *None*) Key for annotations of observations/cells or variables/genes.

use_time_prior : *str* or bool, optional (default: True) Obs key for pseudo-time values. If True, 'velocity_pseudotime' is used if available.

root_key : *str* or bool, optional (default: None) Obs key for root states.

end_key : *str* or bool, optional (default: None) Obs key for end states.

threshold_root_end_prior : *float* (default: 0.9) Threshold for root and final states priors, to be in the range of [0,1]. Values above the threshold will be considered as terminal and included as prior.

minimum_spanning_tree : *bool, optional (default: True)* Whether to prune the tree such that a path from A-to-B is removed if another more confident path exists.

copy : *bool*, optional (default: *False*) Copy *adata* before computation and return a copy. Otherwise, perform computation inplace and return *None*.

Returns

- **connectivities** (*.uns*) – The full adjacency matrix of the abstracted graph, weights correspond to confidence in the connectivities of partitions.
- **connectivities_tree** (*.uns*) – The adjacency matrix of the tree-like subgraph that best explains the topology.
- **transitions_confidence** (*.uns*) – The adjacency matrix of the abstracted directed graph, weights correspond to confidence in the transitions between partitions.

Further tools

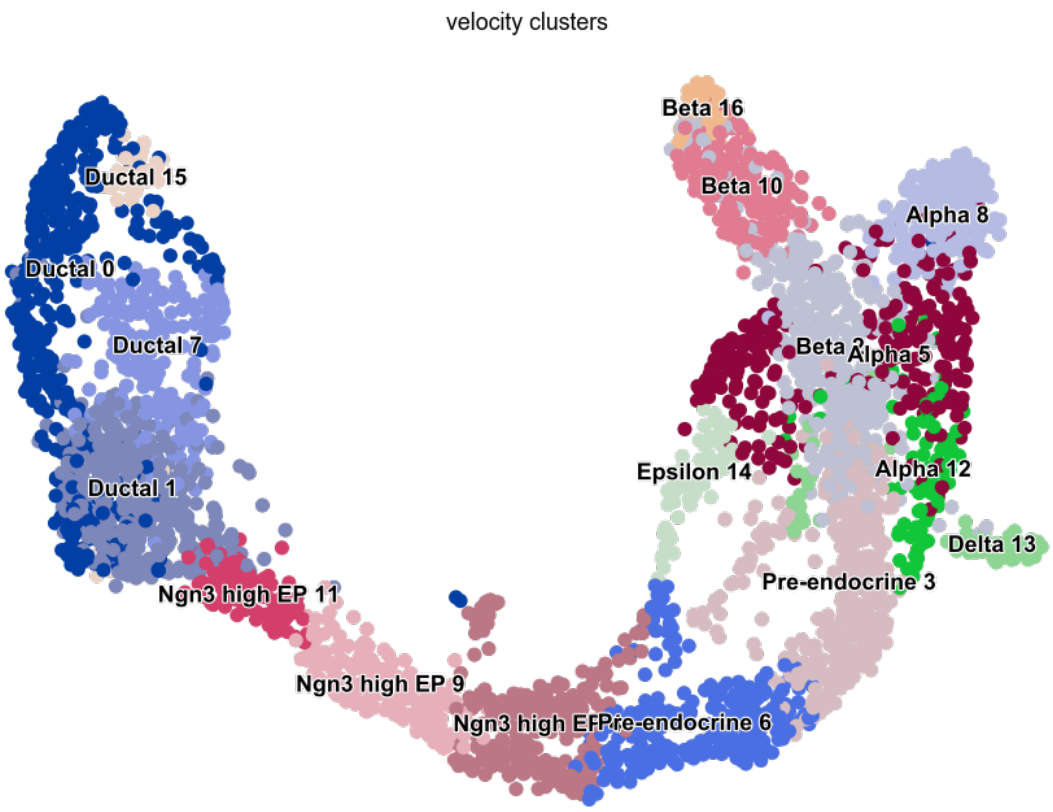
tl.velocity_clusters(data[, vkey, ...])	Computes velocity clusters via louvain on velocities.
tl.velocity_confidence(data[, vkey, copy])	Computes confidences of velocities.
tl.score_genes_cell_cycle(adata[, s_genes, ...])	Score cell cycle genes.

scvelo.tl.velocity_clusters

scvelo.tl.**velocity_clusters**(*data*, *vkey='velocity'*, *match_with='clusters'*, *sort_by='velocity_pseudotime'*, *resolution=None*, *min_likelihood=None*, *copy=False*)

Computes velocity clusters via louvain on velocities.

```
scv.tl.velocity_clusters(adata)
scv.pl.scatter(adata, color='velocity_clusters')
```



Parameters

data : AnnData Annotated data matrix.

vkey : *str* (default: *'velocity'*) Key of velocities computed in *tl.velocity*

match_with : *str* (default: *'clusters'*) The number of genes that appear in the returned tables.

match_with Match the names of the velocity clusters with the names of this key (.obs).

sort_by : *str* or *None* (default: *'dpt_pseudotime'*) Sort velocity clusters by this key (.obs).

resolution : *float* (default: 0.7) Resolution for louvain modularity.

min_likelihood : *float* between *0* and *1* or *None* (default: *None*) Only rank velocity of genes with a likelihood higher than min_likelihood.

copy : *bool* (default: *False*) Return a copy instead of writing to data.

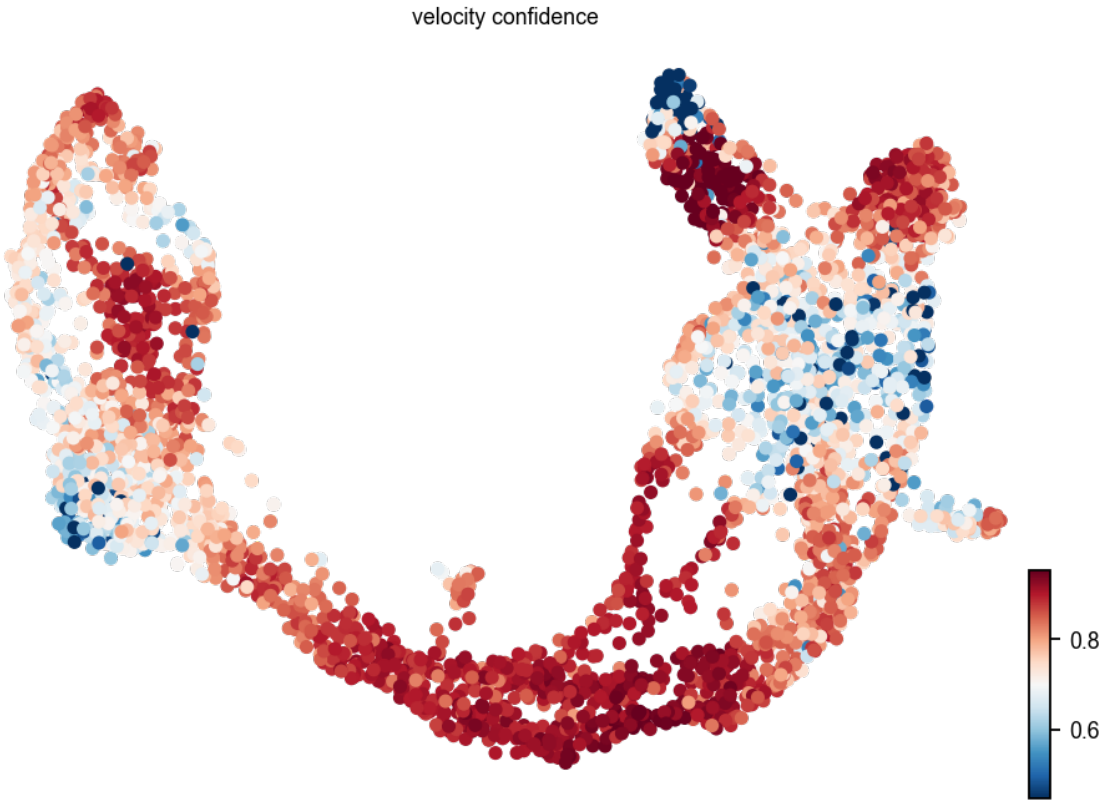
Returns **velocity_clusters** (*.obs*) – Clusters obtained from applying louvain modularity on velocity expression.

scvelo.tl.velocity_confidence

scvelo.tl.**velocity_confidence**(*data*, *vkey='velocity'*, *copy=False*)

Computes confidences of velocities.

```
scv.tl.velocity_confidence(adata)
scv.pl.scatter(adata, color='velocity_confidence', perc=[2,98])
```



Parameters

data : AnnData Annotated data matrix.

vkey : *str* (default: *'velocity'*) Name of velocity estimates to be used.

copy : *bool* (default: *False*) Return a copy instead of writing to adata.

Returns

- **velocity_length** (*.obs*) – Length of the velocity vectors for each individual cell
- **velocity_confidence** (*.obs*) – Confidence for each cell

scvelo.tl.score_genes_cell_cycle

scvelo.tl.**score_genes_cell_cycle**(*adata*, *s_genes=None*, *g2m_genes=None*, *copy=False*, ***kwargs*)

Score cell cycle genes.

Calculates scores and assigns a cell cycle phase (G1, S, G2M) using the list of cell cycle genes defined in Tirosh et al, 2015 (https://doi.org/10.1126/science.aad0501).

Parameters

adata The annotated data matrix.

s_genes List of genes associated with S phase.

g2m_genes List of genes associated with G2M phase.

copy Copy *adata* or modify it inplace.

****kwargs** Are passed to `score_genes()`. *ctrl_size* is not possible, as it's set as *min(len(s_genes), len(g2m_genes))*.

Returns

- **S_score** (*adata.obs*, dtype *object*) – The score for S phase for each cell.
- **G2M_score** (*adata.obs*, dtype *object*) – The score for G2M phase for each cell.
- **phase** (*adata.obs*, dtype *object*) – The cell cycle phase (*S*, *G2M* or *G1*) for each cell.

4.3.4 Plotting (pl)

Base scatter plot

pl.scatter([adata, basis, x, y, vkey, . . .])	Scatter plot along observations or variables axes.

scvelo.pl.scatter

scvelo.pl.**scatter**(*adata=None*, *basis=None*, *x=None*, *y=None*, *vkey=None*, *color=None*, *use_raw=None*, *layer=None*, *color_map=None*, *colorbar=None*, *palette=None*, *size=None*, *alpha=None*, *linewidth=None*, *linecolor=None*, *perc=None*, *groups=None*, *sort_order=True*, *components=None*, *projection=None*, *legend_loc=None*, *legend_loc_lines=None*, *legend_fontsize=None*, *legend_fontweight=None*, *legend_fontoutline=None*, *legend_align_text=None*, *xlabel=None*, *ylabel=None*, *title=None*, *fontsize=None*, *figsize=None*, *xlim=None*, *ylim=None*, *add_density=None*, *add_assignments=None*, *add_linfit=None*, *add_polyfit=None*, *add_rug=None*, *add_text=None*, *add_text_pos=None*, *add_margin=None*, *add_outline=None*, *outline_width=None*, *outline_color=None*, *n_convolve=None*, *smooth=None*, *normalize_data=None*, *rescale_color=None*, *color_gradients=None*, *dpi=None*, *frameon=None*, *zorder=None*, *ncols=None*, *nrows=None*, *wspace=None*, *hspace=None*, *show=None*, *save=None*, *ax=None*, ***kwargs*)

Scatter plot along observations or variables axes.

Parameters

adata : AnnData Annotated data matrix.

x : *str*, *np.ndarray* or *None* (default: *None*) x coordinate

y : *str*, *np.ndarray* or *None* (default: *None*) y coordinate

basis : *str* or list of *str* (default: *None*) Key for embedding. If not specified, use 'umap', 'tsne' or 'pca' (ordered by preference).

vkey : *str* or list of *str* (default: *None*) Key for velocity / steady-state ratio to be visualized.

color : *str*, list of *str* or *None* (default: *None*) Key for annotations of observations/cells or variables/genes

use_raw : *bool* (default: *None*) Use *raw* attribute of *adata* if present.

layer : *str*, list of *str* or *None* (default: *None*) Specify the layer for *color*.

color_map : *str* (default: *matplotlib.rcParams['image.cmap']*) String denoting matplotlib color map.

colorbar : *bool* (default: *False*) Whether to show colorbar.

palette : list of *str* (default: *None*) Colors to use for plotting groups (categorical annotation).

size : *float* (default: 5) Point size.

alpha : *float* (default: 1) Set blending - 0 transparent to 1 opaque.

linewidth : *float* (default: 1) Scaling factor for the width of occurring lines.

linecolor : *str* ir list of *str* (default: 'k') Color of lines from velocity fits, linear fits and polynomial fits

perc : tuple, e.g. [2,98] (default: *None*) Specify percentile for continuous coloring.

groups : *str* or list of *str* (default: *all groups*) Restrict to a few categories in categorical observation annotation. Multiple categories can be passed as list with ['cluster_1', 'cluster_3'], or as string with 'cluster_1, cluster_3'.

sort_order : *bool* (default: *True*) For continuous annotations used as color parameter, plot data points with higher values on top of others.

components : *str* or list of *str* (default: '1,2') For instance, ['1,2', '2,3'].

projection : *{'2d', '3d'} (default: '2d')* Projection of plot.

legend_loc : *str (default: 'none')* Location of legend, either 'on data', 'right margin' or valid keywords for matplotlib.legend.

legend_fontsize : *int* (default: *None*) Legend font size.

legend_fontweight : {'normal', 'bold', ...} (default: *None*) Legend font weight. A numeric value in range 0-1000 or a string. Defaults to 'bold' if *legend_loc = 'on data'*, otherwise to 'normal'. Available are *['light', 'normal', 'medium', 'semibold', 'bold', 'heavy', 'black']*.

legend_fontoutline : float (default: *None*) Line width of the legend font outline in pt. Draws a white outline using the path effect `withStroke`.

legend_align_text : bool or str (default: *None*) Aligns the positions of the legend texts. Set the axis along which the best alignment should be determined. This can be 'y' or True (vertically), 'x' (horizontally), or 'xy' (best alignment in both directions).

right_margin : *float* or list of *float* (default: *None*) Adjust the width of the space right of each plotting panel.

left_margin : *float* or list of *float* (default: *None*) Adjust the width of the space left of each plotting panel.

xlabel : ***str*** **(default:** ***None*****)** Label of x-axis.

ylabel : ***str*** **(default:** ***None*****)** Label of y-axis.

title : ***str*** **(default:** ***None*****)** Provide title for panels either as, e.g. *["title1", "title2", ...]*.

fontsize : ***float*** **(default:** ***None*****)** Label font size.

figsize : tuple (default: ***(7,5)*****)** Figure size.

xlim : tuple, e.g. [0,1] or ***None*** **(default:** ***None*****)** Restrict x-limits of the axis.

ylim : tuple, e.g. [0,1] or ***None*** **(default:** ***None*****)** Restrict y-limits of the axis.

add_density : ***bool*** **or** ***str*** **or** ***None*** **(default:** ***None*****)** Whether to show density of values along x and y axes. Color of the density plot can also be passed as *str*.

add_assignments : ***bool*** **or** ***str*** **or** ***None*** **(default:** ***None*****)** Whether to add assignments to the model curve. Color of the assignments can also be passed as *str*.

add_linfit : ***bool*** **or** ***str*** **or** ***None*** **(default:** ***None*****)** Whether to add linear regression fit to the data points. Color of the line can also be passed as *str*. Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_polyfit : ***bool*** **or** ***str*** **or** ***int*** **or** ***None*** **(default:** ***None*****)** Whether to add polynomial fit to the data points. Color of the polyfit plot can also be passed as *str*. The degree of the polynomial fit can be passed as *int* (default is 2 for quadratic fit). Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_rug : ***str*** **or** ***None*** **(default:** ***None*****)** If categorical observation annotation (e.g. 'clusters') is given, a rugplot is attached to the x-axis showing the data membership to each of the categories.

add_text : ***str*** **(default:** ***None*****)** Text to be added to the plot, passed as *str*.

add_text_pos : ***tuple*****, e.g. [0.05, 0.95] (defaut:** ***[0.05, 0.95]*****)** Text position. Default is *[0.05, 0.95]*, positioning the text at top right corner.

add_margin : ***float*** **(default:** ***None*****)** A value between [-1, 1] to add (positive) and reduce (negative) figure margins.

add_outline : ***bool*** **or** ***str*** **(default:** ***False*****)** Whether to show an outline around scatter plot dots. Alternatively a string of cluster names can be passed, e.g. 'cluster_1, clusters_3'.

outline_width : tuple type ***scalar*** **or** ***None*** **(default:** ***(0.3, 0.05)*****)** Width of the inner and outer outline

outline_color : tuple of type ***str*** **or** ***None*** **(default:** ***('black', 'white')*****)** Inner and outer matplotlib color of the outline

n_convolve : ***int*** **or** ***None*** **(default:** ***None*****)** If *int* is given, data is smoothed by convolution along the x-axis with kernel size *n_convolve*.

smooth : ***bool*** **or** ***int*** **(default:** ***None*****)** Whether to convolve/average the color values over the nearest neighbors. If *int*, it specifies number of neighbors.

normalize_data : ***bool*** **(default:** ***None*****)** Whether to rescale values for x, y to [0,1].

rescale_color : ***tuple*** **(default:** ***None*****)** Boundaries for color rescaling, e.g. [0, 1], setting min/max values of the colorbar.

color_gradients : ***str*** **or** ***np.ndarray*** **(default:** ***None*****)** Key for *.obsm* or array with color gradients by categories.

dpi : *int* (default: 80) Figure dpi.

frameon : *bool* (default: *True*) Draw a frame around the scatter plot.

ncols : *int* (default: *None*) Number of panels per row.

nrows : *int* (default: *None*) Number of panels per column.

wspace : *float* (default: None) Adjust the width of the space between multiple panels.

hspace : *float* (default: None) Adjust the height of the space between multiple panels.

show : *bool*, optional (default: *None*) Show the plot, do not return axis.

save : *bool* or *str*, optional (default: *None*) If *True* or a *str*, save the figure. A string is appended to the default filename. Infer the filetype if ending on {'.pdf', '.png', '.svg'}.

ax : *matplotlib.Axes*, optional (default: *None*) A matplotlib axes object. Only works if plotting a single component.

Returns If *show==False* a *matplotlib.Axis*

Velocity embeddings

pl.velocity_embedding(adata[, basis, vkey, ...])	Scatter plot of velocities on the embedding.
pl.velocity_embedding_grid(adata[, basis, ...])	Scatter plot of velocities on a grid.
pl.velocity_embedding_stream(adata[, basis, ...])	Stream plot of velocities on the embedding.

scvelo.pl.velocity_embedding

scvelo.pl.**velocity_embedding**(*adata*, *basis=None*, *vkey='velocity'*, *density=None*, *arrow_size=None*, *arrow_length=None*, *scale=None*, *X=None*, *V=None*, *recompute=None*, *color=None*, *use_raw=None*, *layer=None*, *color_map=None*, *colorbar=True*, *palette=None*, *size=None*, *alpha=0.2*, *perc=None*, *sort_order=True*, *groups=None*, *components=None*, *projection='2d'*, *legend_loc='none'*, *legend_fontsize=None*, *legend_fontweight=None*, *xlabel=None*, *ylabel=None*, *title=None*, *fontsize=None*, *figsize=None*, *dpi=None*, *frameon=None*, *show=None*, *save=None*, *ax=None*, *ncols=None*, ***kwargs*)

Scatter plot of velocities on the embedding.

Parameters

adata : AnnData Annotated data matrix.

density : *float* (default: 1) Amount of velocities to show - 0 none to 1 all

arrow_size : *float* or triple *headlength, headwidth, headaxislength* (default: 1) Size of arrows.

arrow_length : *float* (default: 1) Length of arrows.

scale : *float* (default: 1) Length of velocities in the embedding.

basis : *str* or list of *str* (default: *None*) Key for embedding. If not specified, use 'umap', 'tsne' or 'pca' (ordered by preference).

vkey : *str* or list of *str* (default: *None*) Key for velocity / steady-state ratio to be visualized.

color : *str*, list of *str* or *None* (default: *None*) Key for annotations of observations/cells or variables/genes

use_raw : *bool* (default: *None*) Use *raw* attribute of *adata* if present.

layer : *str*, list of *str* or *None* (default: *None*) Specify the layer for *color*.

color_map : *str* (default: *matplotlib.rcParams['image.cmap']*) String denoting matplotlib color map.

colorbar : *bool* (default: *False*) Whether to show colorbar.

palette : list of *str* (default: *None*) Colors to use for plotting groups (categorical annotation).

size : *float* (default: 5) Point size.

alpha : *float* (default: 1) Set blending - 0 transparent to 1 opaque.

linewidth : *float* (default: 1) Scaling factor for the width of occurring lines.

linecolor : *str* ir list of *str* (default: 'k') Color of lines from velocity fits, linear fits and polynomial fits

perc : tuple, e.g. [2,98] (default: *None*) Specify percentile for continuous coloring.

groups : *str* or list of *str* (default: *all groups*) Restrict to a few categories in categorical observation annotation. Multiple categories can be passed as list with ['cluster_1', 'cluster_3'], or as string with 'cluster_1, cluster_3'.

sort_order : *bool* (default: *True*) For continuous annotations used as color parameter, plot data points with higher values on top of others.

components : *str* or list of *str* (default: '1,2') For instance, ['1,2', '2,3'].

projection : *{'2d', '3d'} (default: '2d')* Projection of plot.

legend_loc : *str (default: 'none')* Location of legend, either 'on data', 'right margin' or valid keywords for matplotlib.legend.

legend_fontsize : *int* (default: *None*) Legend font size.

legend_fontweight : {'normal', 'bold', ...} (default: *None*) Legend font weight. A numeric value in range 0-1000 or a string. Defaults to 'bold' if *legend_loc = 'on data'*, otherwise to 'normal'. Available are *['light', 'normal', 'medium', 'semibold', 'bold', 'heavy', 'black']*.

legend_fontoutline : float (default: *None*) Line width of the legend font outline in pt. Draws a white outline using the path effect `withStroke`.

legend_align_text : bool or str (default: *None*) Aligns the positions of the legend texts. Set the axis along which the best alignment should be determined. This can be 'y' or True (vertically), 'x' (horizontally), or 'xy' (best alignment in both directions).

right_margin : *float* or list of *float* (default: *None*) Adjust the width of the space right of each plotting panel.

left_margin : *float* or list of *float* (default: *None*) Adjust the width of the space left of each plotting panel.

xlabel : *str* (default: *None*) Label of x-axis.

ylabel : *str* (default: *None*) Label of y-axis.

title : *str* (default: *None*) Provide title for panels either as, e.g. *["title1", "title2", ...]*.

fontsize : *float* (default: *None*) Label font size.

figsize : tuple (default: *(7,5)*) Figure size.

xlim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict x-limits of the axis.

ylim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict y-limits of the axis.

add_density : *bool* or *str* or *None* (default: *None*) Whether to show density of values along x and y axes. Color of the density plot can also be passed as *str*.

add_assignments : *bool* or *str* or *None* (default: *None*) Whether to add assignments to the model curve. Color of the assignments can also be passed as *str*.

add_linfit : *bool* or *str* or *None* (default: *None*) Whether to add linear regression fit to the data points. Color of the line can also be passed as *str*. Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_polyfit : *bool* or *str* or *int* or *None* (default: *None*) Whether to add polynomial fit to the data points. Color of the polyfit plot can also be passed as *str*. The degree of the polynomial fit can be passed as *int* (default is 2 for quadratic fit). Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_rug : *str* or *None* (default: *None*) If categorical observation annotation (e.g. 'clusters') is given, a rugplot is attached to the x-axis showing the data membership to each of the categories.

add_text : *str* (default: *None*) Text to be added to the plot, passed as *str*.

add_text_pos : *tuple*, e.g. [0.05, 0.95] (defaut: *[0.05, 0.95]*) Text position. Default is *[0.05, 0.95]*, positioning the text at top right corner.

add_margin : *float* (default: *None*) A value between [-1, 1] to add (positive) and reduce (negative) figure margins.

add_outline : *bool* or *str* (default: *False*) Whether to show an outline around scatter plot dots. Alternatively a string of cluster names can be passed, e.g. 'cluster_1, clusters_3'.

outline_width : tuple type *scalar* or *None* (default: *(0.3, 0.05)*) Width of the inner and outer outline

outline_color : tuple of type *str* or *None* (default: *('black', 'white')*) Inner and outer matplotlib color of the outline

n_convolve : *int* or *None* (default: *None*) If *int* is given, data is smoothed by convolution along the x-axis with kernel size *n_convolve*.

smooth : *bool* or *int* (default: *None*) Whether to convolve/average the color values over the nearest neighbors. If *int*, it specifies number of neighbors.

normalize_data : *bool* (default: *None*) Whether to rescale values for x, y to [0,1].

rescale_color : *tuple* (default: *None*) Boundaries for color rescaling, e.g. [0, 1], setting min/max values of the colorbar.

color_gradients : *str* or *np.ndarray* (default: *None*) Key for *.obsm* or array with color gradients by categories.

dpi : *int* (default: 80) Figure dpi.

frameon : *bool* (default: *True*) Draw a frame around the scatter plot.

ncols : *int* (default: *None*) Number of panels per row.

nrows : *int* (default: *None*) Number of panels per column.

wspace : *float* (default: None) Adjust the width of the space between multiple panels.

hspace : *float* (default: None) Adjust the height of the space between multiple panels.

show : *bool*, optional (default: *None*) Show the plot, do not return axis.

save : *bool* or *str*, optional (default: *None*) If *True* or a *str*, save the figure. A string is appended to the default filename. Infer the filetype if ending on {'.pdf', '.png', '.svg'}.

ax : *matplotlib.Axes*, optional (default: *None*) A matplotlib axes object. Only works if plotting a single component.

Returns *matplotlib.Axis* if *show==False*

scvelo.pl.velocity_embedding_grid

scvelo.pl.**velocity_embedding_grid**(*adata*, *basis=None*, *vkey='velocity'*, *density=None*, *smooth=None*, *min_mass=None*, *arrow_size=None*, *arrow_length=None*, *arrow_color=None*, *scale=None*, *autoscale=True*, *n_neighbors=None*, *recompute=None*, *X=None*, *V=None*, *X_grid=None*, *V_grid=None*, *principal_curve=False*, *color=None*, *use_raw=None*, *layer=None*, *color_map=None*, *colorbar=True*, *palette=None*, *size=None*, *alpha=0.2*, *perc=None*, *sort_order=True*, *groups=None*, *components=None*, *projection='2d'*, *legend_loc='none'*, *legend_fontsize=None*, *legend_fontweight=None*, *xlabel=None*, *ylabel=None*, *title=None*, *fontsize=None*, *figsize=None*, *dpi=None*, *frameon=None*, *show=None*, *save=None*, *ax=None*, *ncols=None*, ***kwargs*)

Scatter plot of velocities on a grid.

Parameters

adata : AnnData Annotated data matrix.

density : *float* (default: 1) Amount of velocities to show - 0 none to 1 all

arrow_size : *float* or triple *headlength, headwidth, headaxislength* (default: 1) Size of arrows.

arrow_length : *float* (default: 1) Length of arrows.

scale : *float* (default: 1) Length of velocities in the embedding.

min_mass : *float* or *None* (default: *None*) Minimum threshold for mass to be shown. It can range between 0 (all velocities) and 100 (large velocities).

smooth : *bool* or *int* (default: *None*) Multiplication factor for scale in Gaussian kernel around grid point.

n_neighbors : *int* (default: None) Number of neighbors to consider around grid point.

X : *np.ndarray* (default: None) embedding grid point coordinates

V : *np.ndarray* (default: None) embedding grid velocity coordinates

basis : *str* or list of *str* (default: *None*) Key for embedding. If not specified, use 'umap', 'tsne' or 'pca' (ordered by preference).

vkey : *str* or list of *str* (default: *None*) Key for velocity / steady-state ratio to be visualized.

color : *str*, list of *str* or *None* (default: *None*) Key for annotations of observations/cells or variables/genes

use_raw : *bool* (default: *None*) Use *raw* attribute of *adata* if present.

layer : *str*, list of *str* or *None* (default: *None*) Specify the layer for *color*.

color_map : *str* (default: *matplotlib.rcParams['image.cmap']*) String denoting matplotlib color map.

colorbar : *bool* (default: *False*) Whether to show colorbar.

palette : list of *str* (default: *None*) Colors to use for plotting groups (categorical annotation).

size : *float* (default: 5) Point size.

alpha : *float* (default: 1) Set blending - 0 transparent to 1 opaque.

linewidth : *float* (default: 1) Scaling factor for the width of occurring lines.

linecolor : *str* ir list of *str* (default: 'k') Color of lines from velocity fits, linear fits and polynomial fits

perc : tuple, e.g. [2,98] (default: *None*) Specify percentile for continuous coloring.

groups : *str* or list of *str* (default: *all groups*) Restrict to a few categories in categorical observation annotation. Multiple categories can be passed as list with ['cluster_1', 'cluster_3'], or as string with 'cluster_1, cluster_3'.

sort_order : *bool* (default: *True*) For continuous annotations used as color parameter, plot data points with higher values on top of others.

components : *str* or list of *str* (default: '1,2') For instance, ['1,2', '2,3'].

projection : *{'2d', '3d'} (default: '2d')* Projection of plot.

legend_loc : *str (default: 'none')* Location of legend, either 'on data', 'right margin' or valid keywords for matplotlib.legend.

legend_fontsize : *int* (default: *None*) Legend font size.

legend_fontweight : {'normal', 'bold', ...} (default: *None*) Legend font weight. A numeric value in range 0-1000 or a string. Defaults to 'bold' if *legend_loc = 'on data'*, otherwise to 'normal'. Available are *['light', 'normal', 'medium', 'semibold', 'bold', 'heavy', 'black']*.

legend_fontoutline : float (default: *None*) Line width of the legend font outline in pt. Draws a white outline using the path effect `withStroke`.

legend_align_text : bool or str (default: *None*) Aligns the positions of the legend texts. Set the axis along which the best alignment should be determined. This can be 'y' or True (vertically), 'x' (horizontally), or 'xy' (best alignment in both directions).

right_margin : *float* or list of *float* (default: *None*) Adjust the width of the space right of each plotting panel.

left_margin : *float* or list of *float* (default: *None*) Adjust the width of the space left of each plotting panel.

xlabel : *str* (default: *None*) Label of x-axis.

ylabel : *str* (default: *None*) Label of y-axis.

title : *str* (default: *None*) Provide title for panels either as, e.g. *["title1", "title2", ...]*.

fontsize : *float* (default: *None*) Label font size.

figsize : tuple (default: *(7,5)*) Figure size.

xlim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict x-limits of the axis.

ylim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict y-limits of the axis.

add_density : *bool* or *str* or *None* (default: *None*) Whether to show density of values along x and y axes. Color of the density plot can also be passed as *str*.

add_assignments : *bool* or *str* or *None* (default: *None*) Whether to add assignments to the model curve. Color of the assignments can also be passed as *str*.

add_linfit : *bool* or *str* or *None* (default: *None*) Whether to add linear regression fit to the data points. Color of the line can also be passed as *str*. Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_polyfit : *bool* or *str* or *int* or *None* (default: *None*) Whether to add polynomial fit to the data points. Color of the polyfit plot can also be passed as *str*. The degree of the polynomial fit can be passed as *int* (default is 2 for quadratic fit). Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_rug : *str* or *None* (default: *None*) If categorical observation annotation (e.g. 'clusters') is given, a rugplot is attached to the x-axis showing the data membership to each of the categories.

add_text : *str* (default: *None*) Text to be added to the plot, passed as *str*.

add_text_pos : *tuple*, e.g. [0.05, 0.95] (defaut: *[0.05, 0.95]*) Text position. Default is *[0.05, 0.95]*, positioning the text at top right corner.

add_margin : *float* (default: *None*) A value between [-1, 1] to add (positive) and reduce (negative) figure margins.

add_outline : *bool* or *str* (default: *False*) Whether to show an outline around scatter plot dots. Alternatively a string of cluster names can be passed, e.g. 'cluster_1, clusters_3'.

outline_width : tuple type *scalar* or *None* (default: *(0.3, 0.05)*) Width of the inner and outer outline

outline_color : tuple of type *str* or *None* (default: *('black', 'white')*) Inner and outer matplotlib color of the outline

n_convolve : *int* or *None* (default: *None*) If *int* is given, data is smoothed by convolution along the x-axis with kernel size *n_convolve*.

smooth Whether to convolve/average the color values over the nearest neighbors. If *int*, it specifies number of neighbors.

normalize_data : *bool* (default: *None*) Whether to rescale values for x, y to [0,1].

rescale_color : *tuple* (default: *None*) Boundaries for color rescaling, e.g. [0, 1], setting min/max values of the colorbar.

color_gradients : *str* or *np.ndarray* (default: *None*) Key for *.obsm* or array with color gradients by categories.

dpi : *int* (default: 80) Figure dpi.

frameon : *bool* (default: *True*) Draw a frame around the scatter plot.

ncols : *int* (default: *None*) Number of panels per row.

nrows : *int* (default: *None*) Number of panels per column.

wspace : *float* (default: None) Adjust the width of the space between multiple panels.

hspace : *float* (default: None) Adjust the height of the space between multiple panels.

show : *bool*, optional (default: *None*) Show the plot, do not return axis.

save : *bool* or *str*, optional (default: *None*) If *True* or a *str*, save the figure. A string is appended to the default filename. Infer the filetype if ending on {'.pdf', '.png', '.svg'}.

ax : *matplotlib.Axes*, optional (default: *None*) A matplotlib axes object. Only works if plotting a single component.

Returns *matplotlib.Axis* if *show==False*

scvelo.pl.velocity_embedding_stream

scvelo.pl.**velocity_embedding_stream**(*adata*, *basis=None*, *vkey='velocity'*, *density=2*, *smooth=None*, *min_mass=None*, *cutoff_perc=None*, *arrow_color=None*, *arrow_size=1*, *arrow_style='-|>'*, *max_length=4*, *integration_direction='both'*, *linewidth=None*, *n_neighbors=None*, *recompute=None*, *color=None*, *use_raw=None*, *layer=None*, *color_map=None*, *colorbar=True*, *palette=None*, *size=None*, *alpha=0.3*, *perc=None*, *X=None*, *V=None*, *X_grid=None*, *V_grid=None*, *sort_order=True*, *groups=None*, *components=None*, *legend_loc='on data'*, *legend_fontsize=None*, *legend_fontweight=None*, *xlabel=None*, *ylabel=None*, *title=None*, *fontsize=None*, *figsize=None*, *dpi=None*, *frameon=None*, *show=None*, *save=None*, *ax=None*, *ncols=None*, ***kwargs*)

Stream plot of velocities on the embedding.

Parameters

adata : AnnData Annotated data matrix.

density : *float* (default: 2) Controls the closeness of streamlines. When density = 2 (default), the domain is divided into a 60x60 grid, whereas density linearly scales this grid. Each cell in the grid can have, at most, one traversing streamline. For different densities in each direction, use a tuple (density_x, density_y).

smooth : *bool* or *int* (default: *None*) Multiplication factor for scale in Gaussian kernel around grid point.

min_mass : *float* (default: 1) Minimum threshold for mass to be shown. It can range between 0 (all velocities) and 5 (large velocities only).

cutoff_perc : *float* (default: *None*) If set, mask small velocities below a percentile threshold (between 0 and 100).

linewidth : *float* (default: 1) Line width for streamplot.

arrow_color : *str* or 2D array (default: 'k') The streamline color. If given an array, it must have the same shape as u and v.

arrow_size : *float* (default: 1) Scaling factor for the arrow size.

arrow_style : *str* (default: '-|>') Arrow style specification, '-|>' or '->'.

max_length : *float* (default: 4) Maximum length of streamline in axes coordinates.

integration_direction : *str* (default: 'both') Integrate the streamline in 'forward', 'backward' or 'both' directions.

n_neighbors : *int* (default: None) Number of neighbors to consider around grid point.

X : *np.ndarray* (default: None) Embedding coordinates. Using *adata.obsm['X_umap']* per default.

V : *np.ndarray* (default: None) Embedding velocity coordinates. Using *adata.obsm['velocity_umap']* per default.

basis : *str* or list of *str* (default: *None*) Key for embedding. If not specified, use 'umap', 'tsne' or 'pca' (ordered by preference).

vkey : *str* or list of *str* (default: *None*) Key for velocity / steady-state ratio to be visualized.

color : *str*, list of *str* or *None* (default: *None*) Key for annotations of observations/cells or variables/genes

use_raw : *bool* (default: *None*) Use *raw* attribute of *adata* if present.

layer : *str*, list of *str* or *None* (default: *None*) Specify the layer for *color*.

color_map : *str* (default: *matplotlib.rcParams['image.cmap']*) String denoting matplotlib color map.

colorbar : *bool* (default: *False*) Whether to show colorbar.

palette : list of *str* (default: *None*) Colors to use for plotting groups (categorical annotation).

size : *float* (default: 5) Point size.

alpha : *float* (default: 1) Set blending - 0 transparent to 1 opaque.

linewidth Scaling factor for the width of occurring lines.

linecolor : *str* ir list of *str* (default: 'k') Color of lines from velocity fits, linear fits and polynomial fits

perc : tuple, e.g. [2,98] (default: *None*) Specify percentile for continuous coloring.

groups : *str* or list of *str* (default: *all groups*) Restrict to a few categories in categorical observation annotation. Multiple categories can be passed as list with ['cluster_1', 'cluster_3'], or as string with 'cluster_1, cluster_3'.

sort_order : *bool* (default: *True*) For continuous annotations used as color parameter, plot data points with higher values on top of others.

components : *str* or list of *str* (default: '1,2') For instance, ['1,2', '2,3'].

projection : *{'2d', '3d'} (default: '2d')* Projection of plot.

legend_loc : *str (default: 'none')* Location of legend, either 'on data', 'right margin' or valid keywords for matplotlib.legend.

legend_fontsize : *int* (default: *None*) Legend font size.

legend_fontweight : {'normal', 'bold', ...} (default: *None*) Legend font weight. A numeric value in range 0-1000 or a string. Defaults to 'bold' if *legend_loc = 'on data'*, otherwise to 'normal'. Available are *['light', 'normal', 'medium', 'semibold', 'bold', 'heavy', 'black']*.

legend_fontoutline : float (default: *None*) Line width of the legend font outline in pt. Draws a white outline using the path effect `withStroke`.

legend_align_text : bool or str (default: *None*) Aligns the positions of the legend texts. Set the axis along which the best alignment should be determined. This can be 'y' or True (vertically), 'x' (horizontally), or 'xy' (best alignment in both directions).

right_margin : *float* or list of *float* (default: *None*) Adjust the width of the space right of each plotting panel.

left_margin : *float* or list of *float* (default: *None*) Adjust the width of the space left of each plotting panel.

xlabel : *str* (default: *None*) Label of x-axis.

ylabel : *str* (default: *None*) Label of y-axis.

title : *str* (default: *None*) Provide title for panels either as, e.g. *["title1", "title2", ...]*.

fontsize : *float* (default: *None*) Label font size.

figsize : tuple (default: *(7,5)*) Figure size.

xlim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict x-limits of the axis.

ylim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict y-limits of the axis.

add_density : *bool* or *str* or *None* (default: *None*) Whether to show density of values along x and y axes. Color of the density plot can also be passed as *str*.

add_assignments : *bool* or *str* or *None* (default: *None*) Whether to add assignments to the model curve. Color of the assignments can also be passed as *str*.

add_linfit : *bool* or *str* or *None* (default: *None*) Whether to add linear regression fit to the data points. Color of the line can also be passed as *str*. Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_polyfit : *bool* or *str* or *int* or *None* (default: *None*) Whether to add polynomial fit to the data points. Color of the polyfit plot can also be passed as *str*. The degree of the polynomial fit can be passed as *int* (default is 2 for quadratic fit). Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_rug : *str* or *None* (default: *None*) If categorical observation annotation (e.g. 'clusters') is given, a rugplot is attached to the x-axis showing the data membership to each of the categories.

add_text : *str* (default: *None*) Text to be added to the plot, passed as *str*.

add_text_pos : *tuple*, e.g. [0.05, 0.95] (defaut: *[0.05, 0.95]*) Text position. Default is *[0.05, 0.95]*, positioning the text at top right corner.

add_margin : *float* (default: *None*) A value between [-1, 1] to add (positive) and reduce (negative) figure margins.

add_outline : *bool* or *str* (default: *False*) Whether to show an outline around scatter plot dots. Alternatively a string of cluster names can be passed, e.g. 'cluster_1, clusters_3'.

outline_width : tuple type *scalar* or *None* (default: *(0.3, 0.05)*) Width of the inner and outer outline

outline_color : tuple of type *str* or *None* (default: *('black', 'white')*) Inner and outer matplotlib color of the outline

n_convolve : *int* or *None* (default: *None*) If *int* is given, data is smoothed by convolution along the x-axis with kernel size *n_convolve*.

smooth Whether to convolve/average the color values over the nearest neighbors. If *int*, it specifies number of neighbors.

normalize_data : *bool* (default: *None*) Whether to rescale values for x, y to [0,1].

rescale_color : *tuple* (default: *None*) Boundaries for color rescaling, e.g. [0, 1], setting min/max values of the colorbar.

color_gradients : *str* or *np.ndarray* (default: *None*) Key for *.obsm* or array with color gradients by categories.

dpi : *int* (default: 80) Figure dpi.

frameon : *bool* (default: *True*) Draw a frame around the scatter plot.

ncols : *int* (default: *None*) Number of panels per row.

nrows : *int* (default: *None*) Number of panels per column.

wspace : *float* (default: None) Adjust the width of the space between multiple panels.

hspace : *float* (default: None) Adjust the height of the space between multiple panels.

show : *bool*, optional (default: *None*) Show the plot, do not return axis.

save : *bool* or *str*, optional (default: *None*) If *True* or a *str*, save the figure. A string is appended to the default filename. Infer the filetype if ending on {'.pdf', '.png', '.svg'}.

ax : *matplotlib.Axes*, optional (default: *None*) A matplotlib axes object. Only works if plotting a single component.

Returns *matplotlib.Axis* if *show==False*

Velocity graph

pl.velocity(adata[, var_names, basis, vkey, ...])	Phase and velocity plot for set of genes.
pl.velocity_graph(adata[, basis, vkey, ...])	Plot of the velocity graph.
pl.paga(adata[, basis, vkey, color, layer, ...])	Plot PAGA graph with velocity-directed edges.

scvelo.pl.velocity

scvelo.pl.**velocity**(*adata*, *var_names=None*, *basis=None*, *vkey='velocity'*, *mode=None*, *fits=None*, *layers='all'*, *color=None*, *color_map=None*, *colorbar=True*, *perc=[2, 98]*, *alpha=0.5*, *size=None*, *groupby=None*, *groups=None*, *legend_loc='none'*, *legend_fontsize=8*, *use_raw=False*, *fontsize=None*, *figsize=None*, *dpi=None*, *show=None*, *save=None*, *ax=None*, *ncols=None*, ***kwargs*)

Phase and velocity plot for set of genes.

The phase plot shows spliced against unspliced expressions with steady-state fit. Further the embedding is shown colored by velocity and expression.

Parameters

adata : AnnData Annotated data matrix.

var_names : *str* or list of *str* (default: *None*) Which variables to show.

basis : *str* (default: *'umap'*) Key for embedding coordinates.

mode : *'stochastic'* or *None* (default: *None*) Whether to show show covariability phase portrait.

fits : *str* or list of *str* (default: *['velocity', 'dynamics']*) Which steady-state estimates to show.

layers : *str* or list of *str* (default: *'all'*) Which layers to show.

color : *str*, list of *str* or *None* (default: *None*) Key for annotations of observations/cells or variables/genes

color_map : *str* or tuple (default: *['RdYlGn', 'gnuplot_r']*) String denoting matplotlib color map. If tuple is given, first and latter color map correspond to velocity and expression, respectively.

perc : tuple, e.g. [2,98] (default: *[2,98]*) Specify percentile for continuous coloring.

groups : *str*, *list* (default: *None*) Subset of groups, e.g. ['g1', 'g2'], to which the plot shall be restricted.

groupby : *str*, *list* or *np.ndarray* (default: *None*) Key of observations grouping to consider.

legend_loc : *str (default: 'none')* Location of legend, either 'on data', 'right margin' or valid keywords for matplotlib.legend.

size : *float* (default: 5) Point size.

alpha : *float* (default: 1) Set blending - 0 transparent to 1 opaque.

fontsize : *float* (default: *None*) Label font size.

figsize : tuple (default: *(7,5)*) Figure size.

dpi : *int* (default: 80) Figure dpi.

show : *bool*, optional (default: *None*) Show the plot, do not return axis.

save : *bool* or *str*, optional (default: *None*) If *True* or a *str*, save the figure. A string is appended to the default filename. Infer the filetype if ending on {'.pdf', '.png', '.svg'}.

ax : *matplotlib.Axes*, optional (default: *None*) A matplotlib axes object. Only works if plotting a single component.

ncols : *int* or *None* (default: *None*) Number of columns to arange multiplots into.

scvelo.pl.velocity_graph

scvelo.pl.**velocity_graph**(*adata*, *basis=None*, *vkey='velocity'*, *which_graph=None*, *n_neighbors=10*, *arrows=None*, *arrowsize=3*, *alpha=0.8*, *perc=None*, *threshold=None*, *edge_width=0.2*, *edge_color='grey'*, *edges_on_top=None*, *color=None*, *layer=None*, *size=None*, *groups=None*, *components=None*, *title=None*, *dpi=None*, *show=None*, *save=None*, *ax=None*, ***kwargs*)

Plot of the velocity graph.

Parameters

adata : AnnData Annotated data matrix.

which_graph : *'velocity_graph'* or *'connectivities'* (default: *None*) Whether to show transitions from velocity graph or neighbor connectivities.

n_neighbors : *int* (default: 10) Number of neighbors to be included for generating connectivity / velocity graph.

arrows : *bool* (default: *None*) Whether to display arrows instead of edges. Recommended to be used only on a cluster by setting groups parameter.

arrowsize : *int* (default: 3) Size of the arrow heads.

basis : *str* or list of *str* (default: *None*) Key for embedding. If not specified, use 'umap', 'tsne' or 'pca' (ordered by preference).

vkey : *str* or list of *str* (default: *None*) Key for velocity / steady-state ratio to be visualized.

color : *str*, list of *str* or *None* (default: *None*) Key for annotations of observations/cells or variables/genes

use_raw : *bool* (default: *None*) Use *raw* attribute of *adata* if present.

layer : *str*, list of *str* or *None* (default: *None*) Specify the layer for *color*.

color_map : *str* (default: *matplotlib.rcParams['image.cmap']*) String denoting matplotlib color map.

colorbar : *bool* (default: *False*) Whether to show colorbar.

palette : list of *str* (default: *None*) Colors to use for plotting groups (categorical annotation).

size : *float* (default: 5) Point size.

alpha : *float* (default: 1) Set blending - 0 transparent to 1 opaque.

linewidth : *float* (default: 1) Scaling factor for the width of occurring lines.

linecolor : *str* ir list of *str* (default: 'k') Color of lines from velocity fits, linear fits and polynomial fits

perc : tuple, e.g. [2,98] (default: *None*) Specify percentile for continuous coloring.

groups : *str* or list of *str* (default: *all groups*) Restrict to a few categories in categorical observation annotation. Multiple categories can be passed as list with ['cluster_1', 'cluster_3'], or as string with 'cluster_1, cluster_3'.

sort_order : *bool* (default: *True*) For continuous annotations used as color parameter, plot data points with higher values on top of others.

components : *str* or list of *str* (default: '1,2') For instance, ['1,2', '2,3'].

projection : *{'2d', '3d'} (default: '2d')* Projection of plot.

legend_loc : *str (default: 'none')* Location of legend, either 'on data', 'right margin' or valid keywords for matplotlib.legend.

legend_fontsize : *int* (default: *None*) Legend font size.

legend_fontweight : {'normal', 'bold', ...} (default: *None*) Legend font weight. A numeric value in range 0-1000 or a string. Defaults to 'bold' if *legend_loc = 'on data'*, otherwise to 'normal'. Available are *['light', 'normal', 'medium', 'semibold', 'bold', 'heavy', 'black']*.

legend_fontoutline : float (default: *None*) Line width of the legend font outline in pt. Draws a white outline using the path effect `withStroke`.

legend_align_text : bool or str (default: *None*) Aligns the positions of the legend texts. Set the axis along which the best alignment should be determined. This can be 'y' or True (vertically), 'x' (horizontally), or 'xy' (best alignment in both directions).

right_margin : *float* or list of *float* (default: *None*) Adjust the width of the space right of each plotting panel.

left_margin : *float* or list of *float* (default: *None*) Adjust the width of the space left of each plotting panel.

xlabel : ***str*** **(default:** ***None*****)** Label of x-axis.

ylabel : ***str*** **(default:** ***None*****)** Label of y-axis.

title : ***str*** **(default:** ***None*****)** Provide title for panels either as, e.g. *["title1", "title2", ...]*.

fontsize : ***float*** **(default:** ***None*****)** Label font size.

figsize : tuple (default: ***(7,5)*****)** Figure size.

xlim : tuple, e.g. [0,1] or ***None*** **(default:** ***None*****)** Restrict x-limits of the axis.

ylim : tuple, e.g. [0,1] or ***None*** **(default:** ***None*****)** Restrict y-limits of the axis.

add_density : ***bool*** **or** ***str*** **or** ***None*** **(default:** ***None*****)** Whether to show density of values along x and y axes. Color of the density plot can also be passed as *str*.

add_assignments : ***bool*** **or** ***str*** **or** ***None*** **(default:** ***None*****)** Whether to add assignments to the model curve. Color of the assignments can also be passed as *str*.

add_linfit : ***bool*** **or** ***str*** **or** ***None*** **(default:** ***None*****)** Whether to add linear regression fit to the data points. Color of the line can also be passed as *str*. Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_polyfit : ***bool*** **or** ***str*** **or** ***int*** **or** ***None*** **(default:** ***None*****)** Whether to add polynomial fit to the data points. Color of the polyfit plot can also be passed as *str*. The degree of the polynomial fit can be passed as *int* (default is 2 for quadratic fit). Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_rug : ***str*** **or** ***None*** **(default:** ***None*****)** If categorical observation annotation (e.g. 'clusters') is given, a rugplot is attached to the x-axis showing the data membership to each of the categories.

add_text : ***str*** **(default:** ***None*****)** Text to be added to the plot, passed as *str*.

add_text_pos : ***tuple*****, e.g. [0.05, 0.95] (defaut:** ***[0.05, 0.95]*****)** Text position. Default is *[0.05, 0.95]*, positioning the text at top right corner.

add_margin : ***float*** **(default:** ***None*****)** A value between [-1, 1] to add (positive) and reduce (negative) figure margins.

add_outline : ***bool*** **or** ***str*** **(default:** ***False*****)** Whether to show an outline around scatter plot dots. Alternatively a string of cluster names can be passed, e.g. 'cluster_1, clusters_3'.

outline_width : tuple type ***scalar*** **or** ***None*** **(default:** ***(0.3, 0.05)*****)** Width of the inner and outer outline

outline_color : tuple of type ***str*** **or** ***None*** **(default:** ***('black', 'white')*****)** Inner and outer matplotlib color of the outline

n_convolve : ***int*** **or** ***None*** **(default:** ***None*****)** If *int* is given, data is smoothed by convolution along the x-axis with kernel size *n_convolve*.

smooth : ***bool*** **or** ***int*** **(default:** ***None*****)** Whether to convolve/average the color values over the nearest neighbors. If *int*, it specifies number of neighbors.

normalize_data : ***bool*** **(default:** ***None*****)** Whether to rescale values for x, y to [0,1].

rescale_color : ***tuple*** **(default:** ***None*****)** Boundaries for color rescaling, e.g. [0, 1], setting min/max values of the colorbar.

color_gradients : ***str*** **or** ***np.ndarray*** **(default:** ***None*****)** Key for *.obsm* or array with color gradients by categories.

dpi : *int* (default: 80) Figure dpi.

frameon : *bool* (default: *True*) Draw a frame around the scatter plot.

ncols : *int* (default: *None*) Number of panels per row.

nrows : *int* (default: *None*) Number of panels per column.

wspace : *float* (default: None) Adjust the width of the space between multiple panels.

hspace : *float* (default: None) Adjust the height of the space between multiple panels.

show : *bool*, optional (default: *None*) Show the plot, do not return axis.

save : *bool* or *str*, optional (default: *None*) If *True* or a *str*, save the figure. A string is appended to the default filename. Infer the filetype if ending on {'.pdf', '.png', '.svg'}.

ax : *matplotlib.Axes*, optional (default: *None*) A matplotlib axes object. Only works if plotting a single component.

Returns *matplotlib.Axis* if *show==False*

scvelo.pl.paga

scvelo.pl.**paga**(*adata*, *basis=None*, *vkey='velocity'*, *color=None*, *layer=None*, *title=None*, *threshold=None*, *layout=None*, *layout_kwds=None*, *init_pos=None*, *root=0*, *labels=None*, *single_component=False*, *dashed_edges='connectivities'*, *solid_edges='transitions_confidence'*, *transitions='transitions_confidence'*, *node_size_scale=1*, *node_size_power=0.5*, *edge_width_scale=0.4*, *min_edge_width=None*, *max_edge_width=2*, *arrowsize=15*, *random_state=0*, *pos=None*, *node_colors=None*, *normalize_to_color=False*, *cmap=None*, *cax=None*, *cb_kwds=None*, *add_pos=True*, *export_to_gexf=False*, *plot=True*, *use_raw=None*, *size=None*, *groups=None*, *components=None*, *figsize=None*, *dpi=None*, *show=None*, *save=None*, *ax=None*, *ncols=None*, *scatter_flag=None*, ***kwargs*)

Plot PAGA graph with velocity-directed edges.

PAGA graph with connectivities (dashed) and transitions (solid/arrows).

Parameters

adata Annotated data matrix.

threshold Do not draw edges for weights below this threshold. Set to 0 if you want all edges. Discarding low-connectivity edges helps in getting a much clearer picture of the graph.

color : *str*, list of *str* or *None* (default: *None*) Gene name or *obs* annotation defining the node colors. Also plots the degree of the abstracted graph when passing 'degree_dashed' or 'degree_solid'.

labels The node labels. If *None*, this defaults to the group labels stored in the categorical for which `paga()` has been computed.

pos Two-column array-like storing the x and y coordinates for drawing. Otherwise, path to a *.gdf* file that has been exported from Gephi or a similar graph visualization software.

layout Plotting layout that computes positions. *'fa'* stands for ForceAtlas2, *'fr'* stands for Fruchterman-Reingold, *'rt'* stands for Reingold-Tilford, *'eq_tree'* stands for eqally spaced tree. All but *'fa'* and *'eq_tree'* are igraph layouts. All other igraph layouts are also permitted.

layout_kwds Keywords for the layout.

init_pos Two-column array storing the x and y coordinates for initializing the layout.

random_state For layouts with random initialization like *'fr'*, change this to use different intial states for the optimization. If *None*, the initial state is not reproducible.

root If choosing a tree layout, this is the index of the root node or a list of root node indices. If this is a non-empty vector then the supplied node IDs are used as the roots of the trees (or a single tree if the graph is connected). If this is *None* or an empty list, the root vertices are automatically calculated based on topological sorting.

transitions Key for *.uns['paga']* that specifies the matrix that – for instance *'transitions_confidence'* – that specifies the matrix that stores the arrows.

solid_edges Key for *.uns['paga']* that specifies the matrix that stores the edges to be drawn solid black.

dashed_edges Key for *.uns['paga']* that specifies the matrix that stores the edges to be drawn dashed grey. If *None*, no dashed edges are drawn.

single_component Restrict to largest connected component.

fontsize : *float* (default: *None*) Font size for node labels.

fontoutline Width of the white outline around fonts.

text_kwds Keywords for `text()`.

node_size_scale Increase or decrease the size of the nodes.

node_size_power The power with which groups sizes influence the radius of the nodes.

edge_width_scale Edge with scale in units of *rcParams['lines.linewidth']*.

min_edge_width Min width of solid edges.

max_edge_width Max width of solid and dashed edges.

arrowsize For directed graphs, choose the size of the arrow head head's length and width. See :py:class: *matplotlib.patches.FancyArrowPatch* for attribute *mutation_scale* for more info.

export_to_gexf Export to gexf format to be read by graph visualization programs such as Gephi.

normalize_to_color Whether to normalize categorical plots to *color* or the underlying grouping.

cmap The color map.

cax A matplotlib axes object for a potential colorbar.

cb_kwds Keyword arguments for `ColorbarBase`, e.g., *ticks*.

add_pos Add the positions to *adata.uns['paga']*.

title : *str* (default: *None*) Provide a title.

frameon : *bool* (default: *True*) Draw a frame around the PAGA graph.

plot If *False*, do not create the figure, simply compute the layout.

save : *bool* or *str*, optional (default: *None*) Save figure under default (if set True) or specified filename (if set *str*). Infer the filetype if ending on '.pdf', '.png', or '.svg'.

ax : *matplotlib.Axes*, optional (default: *None*) A matplotlib axes object.

basis : *str* or list of *str* (default: *None*) Key for embedding. If not specified, use 'umap', 'tsne' or 'pca' (ordered by preference).

vkey : *str* or list of *str* (default: *None*) Key for velocity / steady-state ratio to be visualized.

color Key for annotations of observations/cells or variables/genes

use_raw : *bool* (default: *None*) Use *raw* attribute of *adata* if present.

layer : *str*, list of *str* or *None* (default: *None*) Specify the layer for *color*.

color_map : *str* (default: *matplotlib.rcParams['image.cmap']*) String denoting matplotlib color map.

colorbar : *bool* (default: *False*) Whether to show colorbar.

palette : list of *str* (default: *None*) Colors to use for plotting groups (categorical annotation).

size : *float* (default: 5) Point size.

alpha : *float* (default: 1) Set blending - 0 transparent to 1 opaque.

linewidth : *float* (default: 1) Scaling factor for the width of occurring lines.

linecolor : *str* ir list of *str* (default: 'k') Color of lines from velocity fits, linear fits and polynomial fits

perc : tuple, e.g. [2,98] (default: *None*) Specify percentile for continuous coloring.

groups : *str* or list of *str* (default: *all groups*) Restrict to a few categories in categorical observation annotation. Multiple categories can be passed as list with ['cluster_1', 'cluster_3'], or as string with 'cluster_1, cluster_3'.

sort_order : *bool* (default: *True*) For continuous annotations used as color parameter, plot data points with higher values on top of others.

components : *str* or list of *str* (default: '1,2') For instance, ['1,2', '2,3'].

projection : *{'2d', '3d'} (default: '2d')* Projection of plot.

legend_loc : *str (default: 'none')* Location of legend, either 'on data', 'right margin' or valid keywords for matplotlib.legend.

legend_fontsize : *int* (default: *None*) Legend font size.

legend_fontweight : {'normal', 'bold', ...} (default: *None*) Legend font weight. A numeric value in range 0-1000 or a string. Defaults to 'bold' if *legend_loc = 'on data'*, otherwise to 'normal'. Available are *['light', 'normal', 'medium', 'semibold', 'bold', 'heavy', 'black']*.

legend_fontoutline : float (default: *None*) Line width of the legend font outline in pt. Draws a white outline using the path effect `withStroke`.

legend_align_text : bool or str (default: *None*) Aligns the positions of the legend texts. Set the axis along which the best alignment should be determined. This can be 'y' or True (vertically), 'x' (horizontally), or 'xy' (best alignment in both directions).

right_margin : *float* or list of *float* (default: *None*) Adjust the width of the space right of each plotting panel.

left_margin : *float* or list of *float* (default: *None*) Adjust the width of the space left of each plotting panel.

xlabel : *str* (default: *None*) Label of x-axis.

ylabel : *str* (default: *None*) Label of y-axis.

title Provide title for panels either as, e.g. *["title1", "title2", ...]*.

fontsize Label font size.

figsize : tuple (default: *(7,5)*) Figure size.

xlim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict x-limits of the axis.

ylim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict y-limits of the axis.

add_density : *bool* or *str* or *None* (default: *None*) Whether to show density of values along x and y axes. Color of the density plot can also be passed as *str*.

add_assignments : *bool* or *str* or *None* (default: *None*) Whether to add assignments to the model curve. Color of the assignments can also be passed as *str*.

add_linfit : *bool* or *str* or *None* (default: *None*) Whether to add linear regression fit to the data points. Color of the line can also be passed as *str*. Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_polyfit : *bool* or *str* or *int* or *None* (default: *None*) Whether to add polynomial fit to the data points. Color of the polyfit plot can also be passed as *str*. The degree of the polynomial fit can be passed as *int* (default is 2 for quadratic fit). Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_rug : *str* or *None* (default: *None*) If categorical observation annotation (e.g. 'clusters') is given, a rugplot is attached to the x-axis showing the data membership to each of the categories.

add_text : *str* (default: *None*) Text to be added to the plot, passed as *str*.

add_text_pos : *tuple*, e.g. [0.05, 0.95] (defaut: *[0.05, 0.95]*) Text position. Default is *[0.05, 0.95]*, positioning the text at top right corner.

add_margin : *float* (default: *None*) A value between [-1, 1] to add (positive) and reduce (negative) figure margins.

add_outline : *bool* or *str* (default: *False*) Whether to show an outline around scatter plot dots. Alternatively a string of cluster names can be passed, e.g. 'cluster_1, clusters_3'.

outline_width : tuple type *scalar* or *None* (default: *(0.3, 0.05)*) Width of the inner and outer outline

outline_color : tuple of type *str* or *None* (default: *('black', 'white')*) Inner and outer matplotlib color of the outline

n_convolve : *int* or *None* (default: *None*) If *int* is given, data is smoothed by convolution along the x-axis with kernel size *n_convolve*.

smooth : *bool* or *int* (default: *None*) Whether to convolve/average the color values over the nearest neighbors. If *int*, it specifies number of neighbors.

normalize_data : *bool* (default: *None*) Whether to rescale values for x, y to [0,1].

rescale_color : *tuple* (default: *None*) Boundaries for color rescaling, e.g. [0, 1], setting min/max values of the colorbar.

color_gradients : *str* or *np.ndarray* (default: *None*) Key for *.obsm* or array with color gradients by categories.

dpi : *int* (default: 80) Figure dpi.

frameon Draw a frame around the scatter plot.

ncols : *int* (default: *None*) Number of panels per row.

nrows : *int* (default: *None*) Number of panels per column.

wspace : *float* (default: None) Adjust the width of the space between multiple panels.

hspace : *float* (default: None) Adjust the height of the space between multiple panels.

show : *bool*, optional (default: *None*) Show the plot, do not return axis.

save If *True* or a *str*, save the figure. A string is appended to the default filename. Infer the filetype if ending on {'.pdf', '.png', '.svg'}.

ax A matplotlib axes object. Only works if plotting a single component.

Returns

- If *show==False*, one or more `Axes` objects.
- Adds *'pos'* to *adata.uns['paga']* if *add_pos* is *True*.

Further plotting

pl.proportions(adata[, groupby, layers, ...])	Plot pie chart of spliced/unspliced proprtions.
pl.heatmap(adata, var_names[, sortby, ...])	Plot time series for genes as heatmap.
pl.hist(arrays[, alpha, bins, color, ...])	Plot a histogram.

scvelo.pl.proportions

scvelo.pl.**proportions**(*adata*, *groupby='clusters'*, *layers=None*, *highlight='unspliced'*, *add_labels_pie=True*, *add_labels_bar=True*, *fontsize=8*, *figsize=(10, 2)*, *dpi=100*, *use_raw=True*, *show=True*, *save=None*)

Plot pie chart of spliced/unspliced proprtions.

Parameters

adata : AnnData Annotated data matrix.

groupby : *str* (default: 'clusters') Key of observations grouping to consider.

layers : list of *str`(default: `['spliced#, 'unspliced', 'ambiguous']*) Specify the layers of count matrices for computing proportions.

highlight : *str* (default: 'unspliced') Which proportions to highlight in pie chart.

add_labels_pie : *bool* (default: True) Whether to add percentage labels in pie chart.

add_labels_bar : *bool* (default: True) Whether to add percentage labels in bar chart.

fontsize : *float* (default: 8) Label font size.

figsize : tuple (default: *(10,2)*) Figure size.

dpi : *int* (default: 80) Figure dpi.

use_raw : *bool* (default: *True*) Use initial cell sizes before normalization and filtering.

show : *bool* (default: True) Show the plot, do not return axis.

save : *bool* or *str*, optional (default: *None*) If *True* or a *str*, save the figure. A string is appended to the default filename. Infer the filetype if ending on {'.pdf', '.png', '.svg'}.

Returns *Plots the proportions of abundances as pie chart.*

scvelo.pl.heatmap

scvelo.pl.**heatmap**(*adata*, *var_names*, *sortby='latent_time'*, *layer='Ms'*, *color_map='viridis'*, *col_color=None*, *palette='viridis'*, *n_convolve=30*, *standard_scale=0*, *sort=True*, *colorbar=None*, *col_cluster=False*, *row_cluster=False*, *context=None*, *font_scale=None*, *figsize=(8, 4)*, *show=None*, *save=None*, ***kwargs*)

Plot time series for genes as heatmap.

Parameters

adata : AnnData Annotated data matrix.

var_names : *str*, list of *str* Names of variables to use for the plot.

sortby : *str* (default: *'latent_time'*) Observation key to extract time data from.

layer : *str* (default: *'Ms'*) Layer key to extract count data from.

color_map : *str* (default: *'viridis'*) String denoting matplotlib color map.

col_color : *str* or list of *str* (default: *None*) String denoting matplotlib color map to use along the columns.

palette : list of *str* (default: *'viridis'*) Colors to use for plotting groups (categorical annotation).

n_convolve : *int* or *None* (default: *30*) If *int* is given, data is smoothed by convolution along the x-axis with kernel size n_convolve.

standard_scale : *int* or *None* (default: *0*) Either 0 (rows) or 1 (columns). Whether or not to standardize that dimension (each row or column), subtract minimum and divide each by its maximum.

sort : *bool* (default: *True*) Wether to sort the expression values given by xkey.

colorbar : *bool* or *None* (default: *None*) Whether to show colorbar.

{row : *bool* or *None* If True, cluster the {rows, columns}.

col}_cluster : *bool* or *None* If True, cluster the {rows, columns}.

context : *None*, or one of {paper, notebook, talk, poster} A dictionary of parameters or the name of a preconfigured set.

font_scale : *float, optional* Scaling factor to scale the size of the font elements.

figsize : tuple (default: *(8,4)*) Figure size.

show : *bool*, optional (default: *None*) Show the plot, do not return axis.

save : *bool* or *str*, optional (default: *None*) If *True* or a *str*, save the figure. A string is appended to the default filename. Infer the filetype if ending on {'.pdf', '.png', '.svg'}.

kwargs Arguments passed to seaborns clustermap, e.g., set *yticklabels=True* to display all gene names in all rows.

Returns If *show==False* a *matplotlib.Axis*

scvelo.pl.hist

scvelo.pl.**hist**(*arrays*, *alpha=0.5*, *bins=50*, *color=None*, *colors=None*, *labels=None*, *hist=None*, *kde=None*, *bw_method=None*, *xlabel=None*, *ylabel=None*, *xlim=None*, *ylim=None*, *cutoff=None*, *xscale=None*, *yscale=None*, *xticks=None*, *yticks=None*, *fontsize=None*, *legend_fontsize=None*, *figsize=None*, *normed=None*, *perc=None*, *exclude_zeros=None*, *axvline=None*, *axhline=None*, *pdf=None*, *ax=None*, *dpi=None*, *show=True*, ***kwargs*)

Plot a histogram.

Parameters

arrays : : *list* or *array* (default *['royalblue', 'white', 'forestgreen']*) List of colors, either as names or rgb values.

alpha : *list*, *np.ndarray* or *None* (default: *None*) Alpha of the colors. Must be same length as colors.

bins : *int* or *sequence* (default: *50*) If an integer is given, `bins + 1` bin edges are calculated and returned, consistent with *numpy.histogram*. If *bins* is a sequence, gives bin edges, including left edge of first bin and right edge of last bin. In this case, *bins* is returned unmodified.

colors : : *list* or *array* (default *['royalblue', 'white', 'forestgreen']*) List of colors, either as names or rgb values.

labels : str or None (default: *None*) String, or sequence of strings to label clusters.

hist : *bool* or *None* (default: *None*) Whether to show histogram.

kde : *bool* or *None* (default: *None*) Whether to use kernel density estimation on data.

bw_method : *str*, *scalar* or *callable*, (default: *None*) The method used to calculate the estimator bandwidth. This can be 'scott', 'silverman', a scalar constant or a callable. If a scalar, this will be used directly as *kde.factor*. If a callable, it should take a *gaussian_kde* instance as only parameter and return a scalar. If None (default), nothing happens; the current *kde.covariance_factor* method is kept.

xlabel : *str* (default: *None*) Label of x-axis.

ylabel : *str* (default: *None*) Label of y-axis.

xlim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict x-limits of the axis.

ylim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict y-limits of the axis.

cutoff : tuple, e.g. [0,1] or *float* or *None* (default: *None*) Bins will be cut off below and above the cutoff values.

xscale : *log* or *None* (default: *None*) Scale of the x-axis.

yscale : *log* or *None* (default: *None*) Scale of the y-axis.

fontsize : *float* (default: *None*) Label font size.

legend_fontsize : *int* (default: *None*) Legend font size.

figsize : tuple (default: *(7,5)*) Figure size.

normed : *bool* or *None* (default: *None*) Whether to normalize data.

perc : tuple, e.g. [2,98] (default: *None*) Specify percentile for continuous coloring.

exclude_zeros : *bool* or *None* (default: *None*) Whether to exclude zeros in data for the kde and hist plot.

axvline : *float* or *None* (default: *None*) Plot a vertical line at the specified x-value.

float or None (default: *axhline* Plot a horizontal line at the specified y-value.

pdf : *str* or *None* (default: *None*) probability density function to be fitted, e.g., 'norm', 't', 'chi', 'beta', 'gamma', 'laplace' etc.

ax : *matplotlib.Axes*, optional (default: *None*) A matplotlib axes object. Only works if plotting a single component.

dpi : *int* (default: 80) Figure dpi.

show : *bool*, optional (default: *None*) Show the plot, do not return axis.

Returns If *show==False* a *matplotlib.Axis*

4.3.5 Datasets

datasets.pancreas()	Pancreatic endocrinogenesis.
datasets.dentategyrus([adjusted])	Dentate Gyrus neurogenesis.
datasets.forebrain()	Developing human forebrain.
datasets.dentategyrus_lamanno()	Dentate Gyrus neurogenesis.
datasets.gastrulation()	Mouse gastrulation.
datasets.gastrulation_e75()	Mouse gastrulation subset to E7.5.
datasets.gastrulation_erythroid()	Mouse gastrulation subset to erythroid lineage.
datasets.bonemarrow()	Human bone marrow.
datasets.pbmc68k()	Peripheral blood mononuclear cells.
datasets.simulation([n_obs, n_vars, alpha, ...])	Simulation of mRNA splicing kinetics.

scvelo.datasets.pancreas

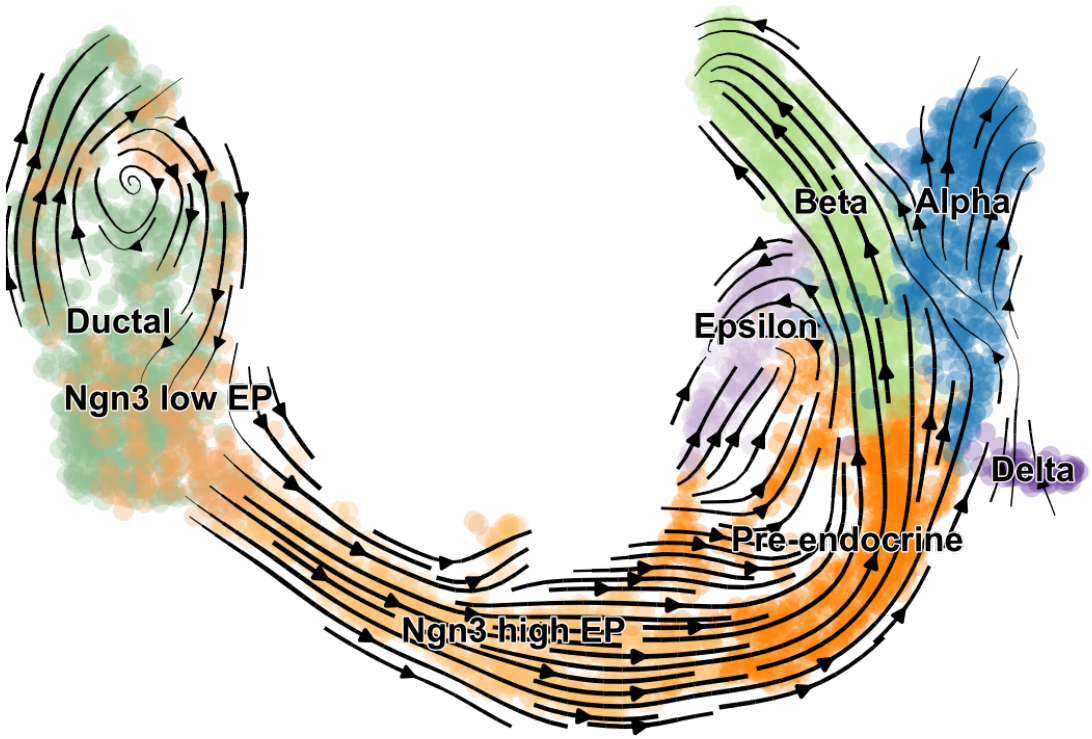
scvelo.datasets.**pancreas**()

Pancreatic endocrinogenesis.

Data from **`Bastidas-Ponce et al. (2019) <https://doi.org/10.1242/dev.173849 >`_**.

Pancreatic epithelial and Ngn3-Venus fusion (NVF) cells during secondary transition with transcriptome profiles sampled from embryonic day 15.5.

Endocrine cells are derived from endocrine progenitors located in the pancreatic epithelium. Endocrine commitment terminates in four major fates: glucagon- producing -cells, insulin-producing -cells, somatostatin-producing -cells and ghrelin-producing -cells.



Returns Returns *adata* object

scvelo.datasets.dentategyrus

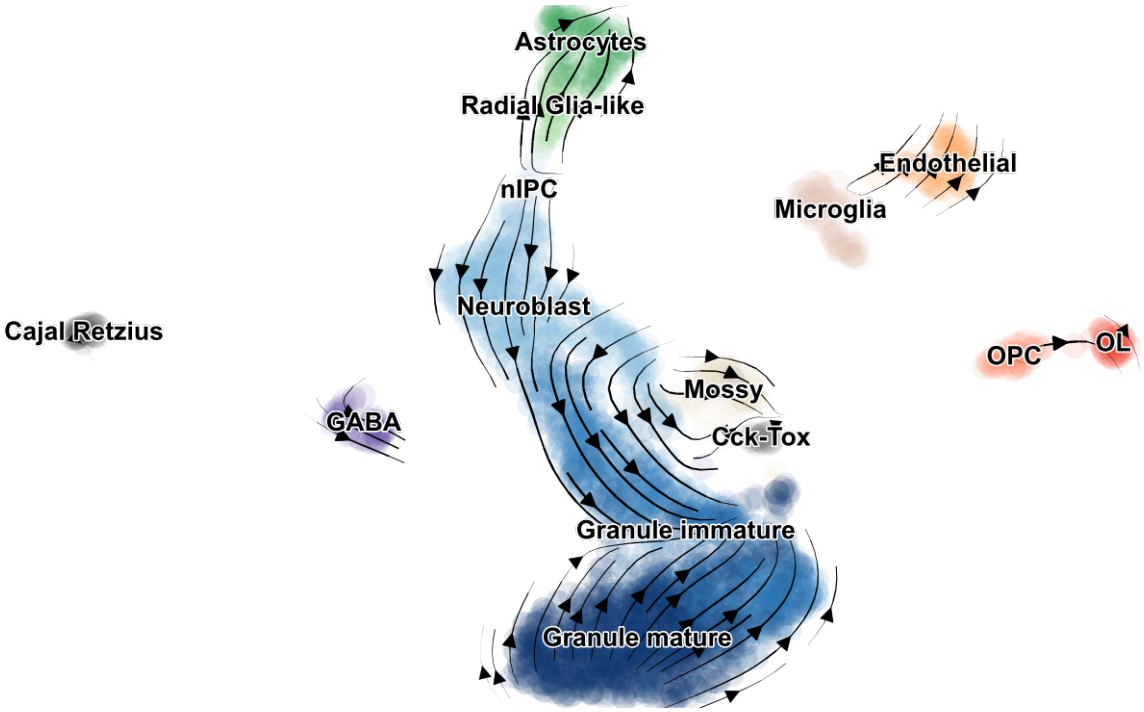
scvelo.datasets.**dentategyrus**(*adjusted=True*)

Dentate Gyrus neurogenesis.

Data from **`Hochgerner et al. (2018) <https://doi.org/10.1038/s41593-017-0056-2 >`_**.

Dentate gyrus (DG) is part of the hippocampus involved in learning, episodic memory formation and spatial coding. The experiment from the developing DG comprises two time points (P12 and P35) measured using droplet-based scRNA-seq (10x Genomics Chromium).

The dominating structure is the granule cell lineage, in which neuroblasts develop into granule cells. Simultaneously, the remaining population forms distinct cell types that are fully differentiated (e.g. Cajal-Retzius cells) or cell types that form a sub-lineage (e.g. GABA cells).



Returns Returns *adata* object

scvelo.datasets.forebrain

scvelo.datasets.**forebrain**()

Developing human forebrain.

From `La Manno et al. (2018) <https://doi.org/10.1038/s41586-018-0414-6>`_.

Forebrain tissue of a human week 10 embryo, focusing on glutamatergic neuronal lineage, obtained from elective routine abortions (10 weeks post-conception).

Returns Returns *adata* object

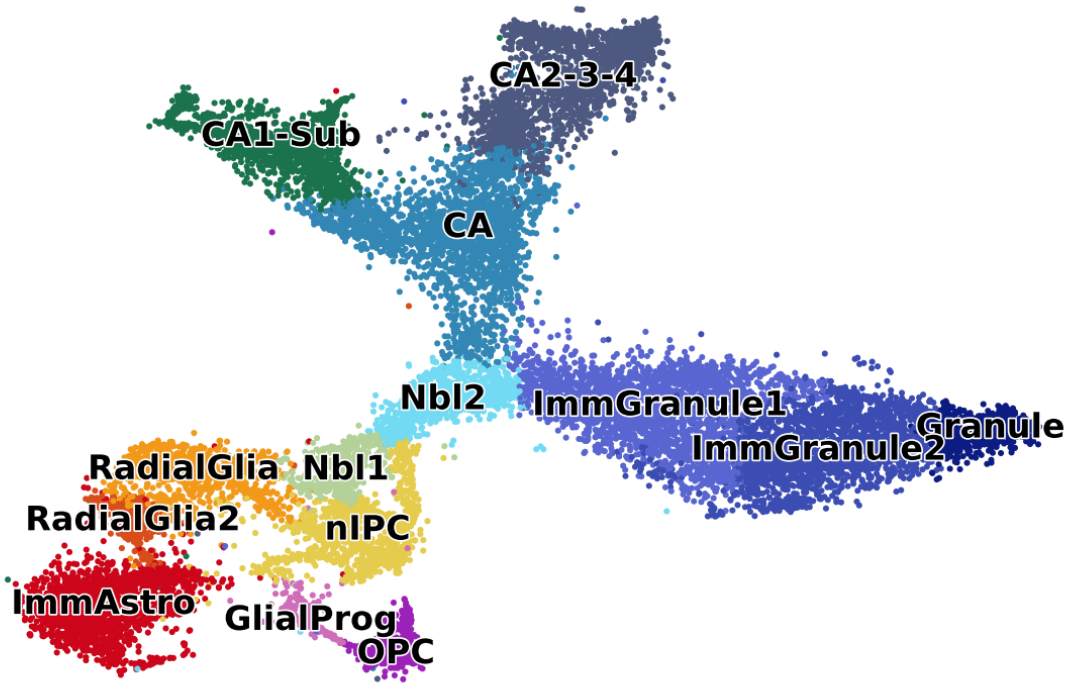
scvelo.datasets.dentategyrus_lamanno

scvelo.datasets.**dentategyrus_lamanno**()

Dentate Gyrus neurogenesis.

From `La Manno et al. (2018) <https://doi.org/10.1038/s41586-018-0414-6>`_.

The experiment from the developing mouse hippocampus comprises two time points (P0 and P5) and reveals the complex manifold with multiple branching lineages towards astrocytes, oligodendrocyte precursors (OPCs), granule neurons and pyramidal neurons.



Returns Returns *adata* object

scvelo.datasets.gastrulation

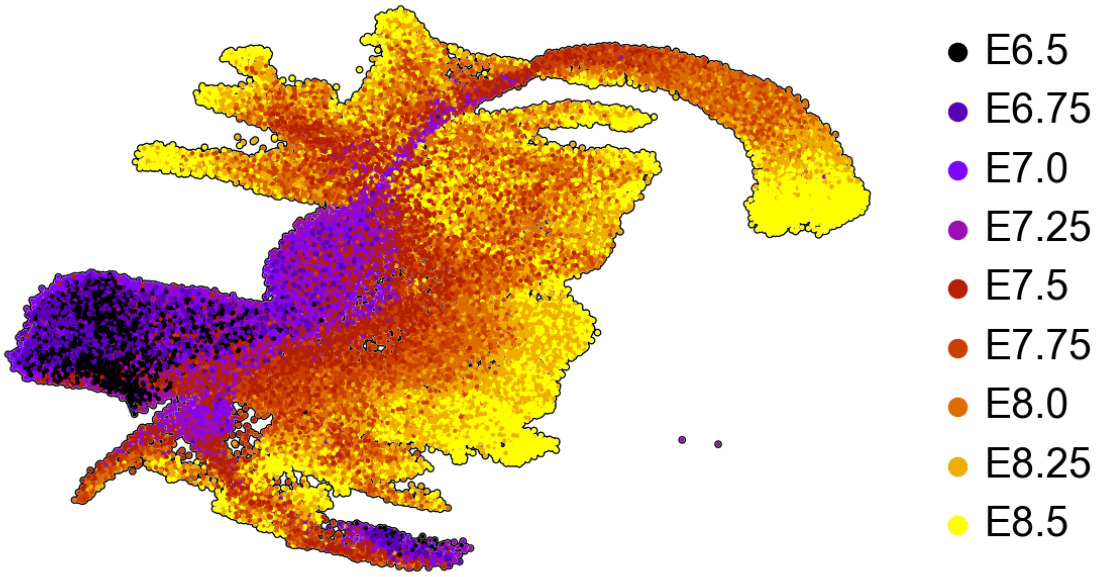
scvelo.datasets.**gastrulation**()

Mouse gastrulation.

Data from **`Pijuan-Sala et al. (2019) <https://doi.org/10.1038/s41586-019-0933-9 >`_**.

Gastrulation represents a key developmental event during which embryonic pluripotent cells diversify into lineage-specific precursors that will generate the adult organism.

This data contains the erythrocyte lineage from Pijuan-Sala et al. (2019). The experiment reveals the molecular map of mouse gastrulation and early organogenesis. It comprises transcriptional profiles of 116,312 single cells from mouse embryos collected at nine sequential time points ranging from 6.5 to 8.5 days post-fertilization. It served to explore the complex events involved in the convergence of visceral and primitive streak-derived endoderm.



Returns Returns *adata* object

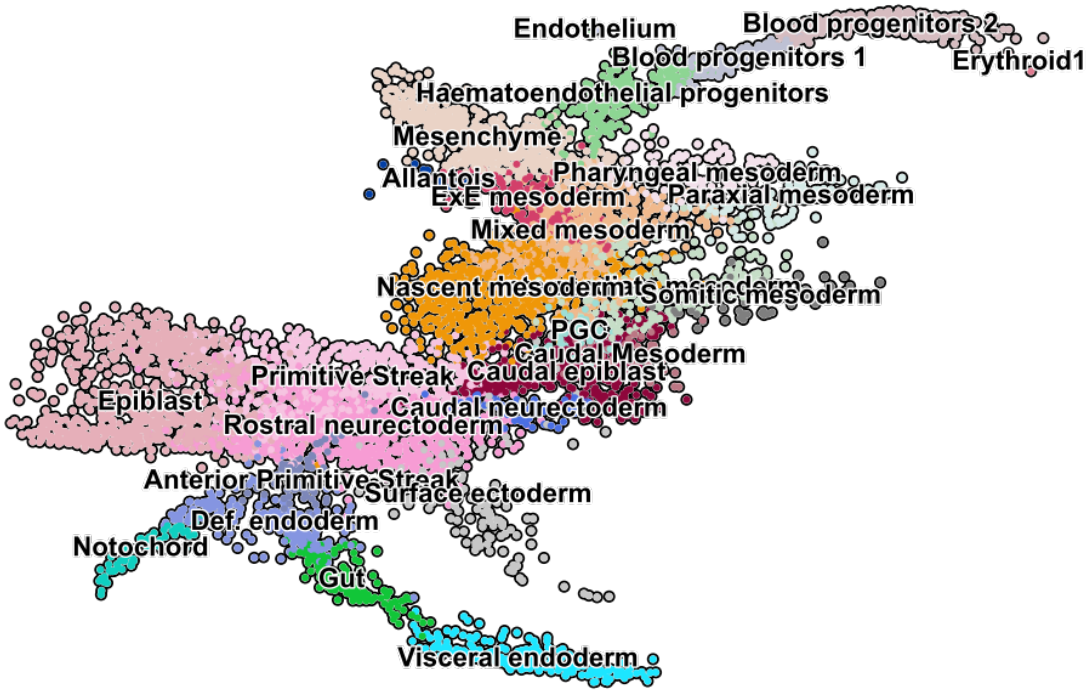
scvelo.datasets.gastrulation_e75

scvelo.datasets.**gastrulation_e75**()

Mouse gastrulation subset to E7.5.

Data from **`Pijuan-Sala et al. (2019) <https://doi.org/10.1038/s41586-019-0933-9 >`_**.

Gastrulation represents a key developmental event during which embryonic pluripotent cells diversify into lineage-specific precursors that will generate the adult organism.



Returns Returns *adata* object

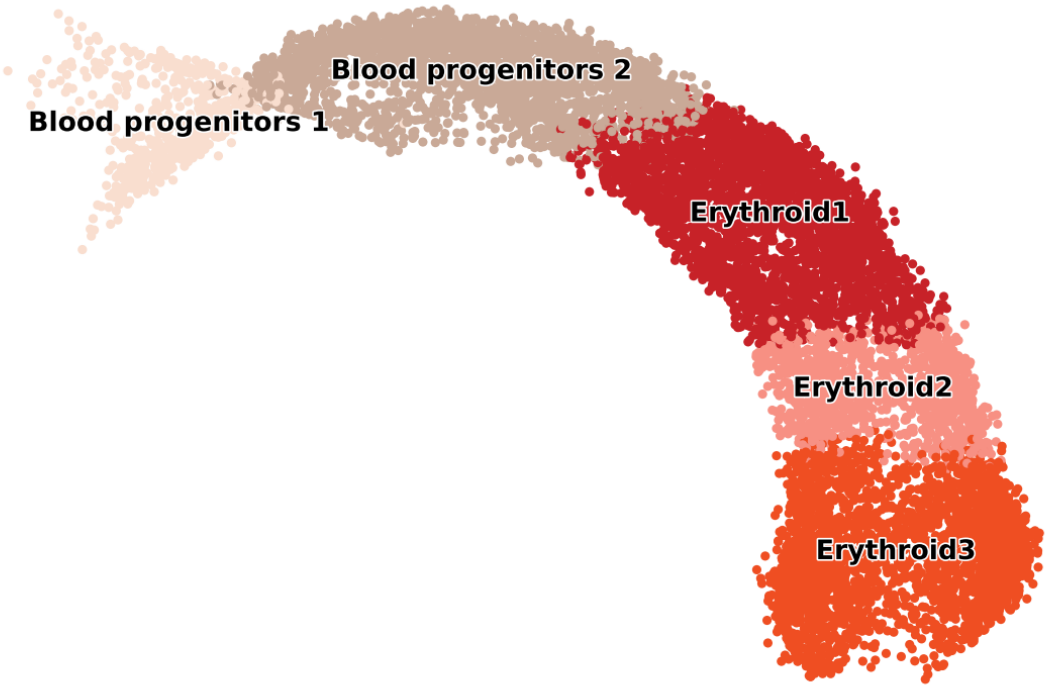
scvelo.datasets.gastrulation_erythroid

scvelo.datasets.**gastrulation_erythroid**()

Mouse gastrulation subset to erythroid lineage.

Data from **`Pijuan-Sala et al. (2019) <https://doi.org/10.1038/s41586-019-0933-9 >`_**.

Gastrulation represents a key developmental event during which embryonic pluripotent cells diversify into lineage-specific precursors that will generate the adult organism.



Returns Returns *adata* object

scvelo.datasets.bonemarrow

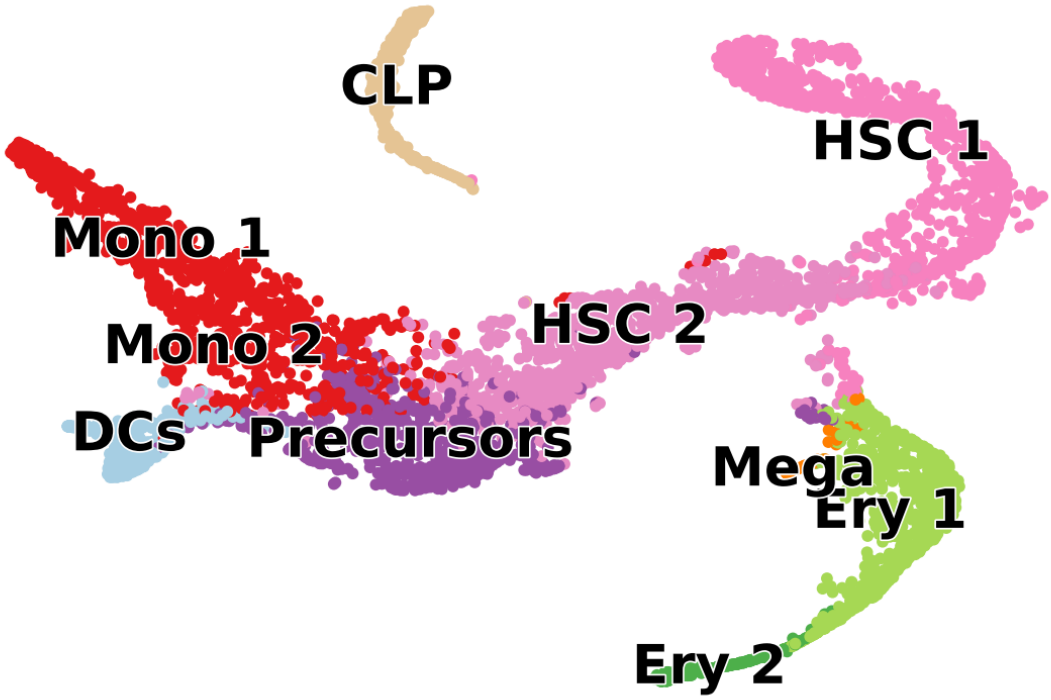
scvelo.datasets.**bonemarrow**()

Human bone marrow.

Data from `Setty et al. (2019) <https://doi.org/10.1242/dev.173849 >`_.

The bone marrow is the primary site of new blood cell production or haematopoiesis. It is composed of hematopoietic cells, marrow adipose tissue, and supportive stromal cells.

This dataset served to detect important landmarks of hematopoietic differentiation, to identify key transcription factors that drive lineage fate choice and to closely track when cells lose plasticity.



Returns Returns *adata* object

scvelo.datasets.pbmc68k

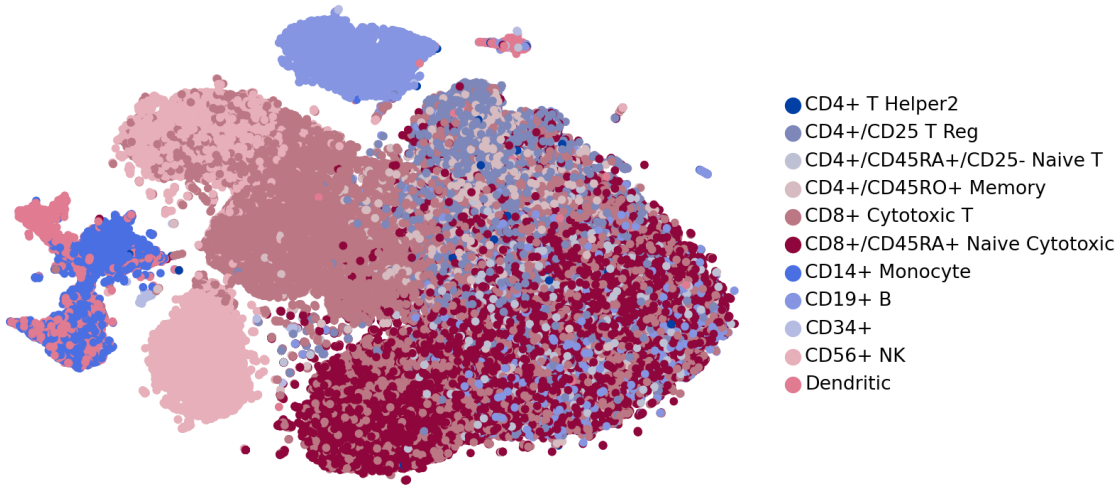
scvelo.datasets.**pbmc68k**()

Peripheral blood mononuclear cells.

Data from **`Zheng et al. (2017) <https://doi.org/10.1038/ncomms14049 >`_**.

This experiment contains 68k peripheral blood mononuclear cells (PBMC) measured using 10X.

PBMCs are a diverse mixture of highly specialized immune cells. They originate from hematopoietic stem cells (HSCs) that reside in the bone marrow and give rise to all blood cells of the immune system (hematopoiesis). HSCs give rise to myeloid (monocytes, macrophages, granulocytes, megakaryocytes, dendritic cells, erythrocytes) and lymphoid (T cells, B cells, NK cells) lineages.



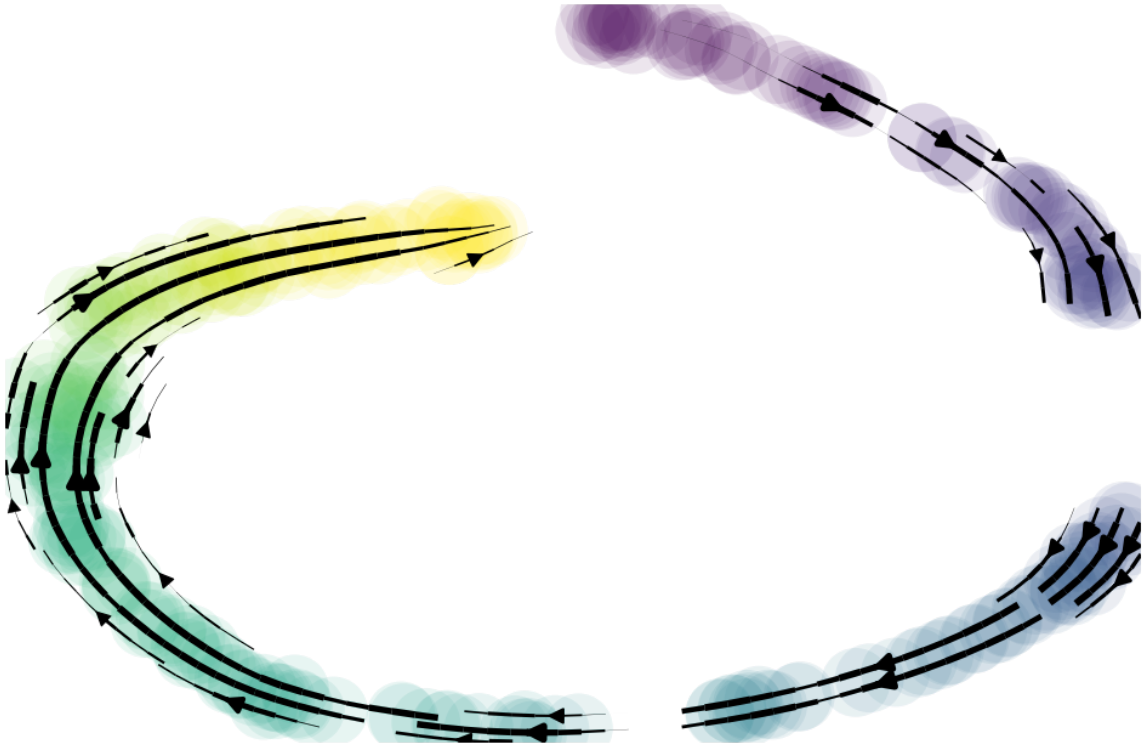
Returns Returns *adata* object

scvelo.datasets.simulation

scvelo.datasets.**simulation**(*n_obs=300*, *n_vars=None*, *alpha=None*, *beta=None*, *gamma=None*, *alpha_=None*, *t_max=None*, *noise_model='normal'*, *noise_level=1*, *switches=None*, *random_seed=0*)

Simulation of mRNA splicing kinetics.

Simulated mRNA metabolism with transcription, splicing and degradation. The parameters for each reaction are randomly sampled from a log-normal distribution and time events follow the Poisson law. The total time spent in a transcriptional state is varied between two and ten hours.



Returns Returns *adata* object

4.3.6 Utils

Get data by key

get_df(data[, keys, layer, index, columns, ...])	Get dataframe for a specified adata key.

scvelo.get_df

scvelo.**get_df**(*data*, *keys=None*, *layer=None*, *index=None*, *columns=None*, *sort_values=None*, *dropna='all'*, *precision=None*)

Get dataframe for a specified adata key.

Return values for specified key (in obs, var, obsm, varm, obsp, varp, uns, or layers) as a dataframe.

Parameters

data : AnnData AnnData object or a numpy array to get values from.

keys : str, List[str], None Keys from *.var_names*, *.obs_names*, *.var*, *.obs*, *.obsm*, *.varm*, *.obsp*, *.varp*, *.uns*, or *.layers*.

layer : str, None Layer of *adata* to use as expression values.

index : List[~T], None List to set as index.

columns : List[~T], None List to set as columns names.

sort_values : bool, None Wether to sort values by first column (sort_values=True) or a specified column.

dropna : _Genericalias[all, any] Drop columns/rows that contain NaNs in all ('all') or in any entry ('any').

precision : int, None Set precision for pandas dataframe.

Return type DataFrame

Returns pd.DataFrame – A dataframe.

Get gene info

utils.gene_info(name[, fields])	Retrieve gene information from biothings client.

scvelo.utils.gene_info

scvelo.utils.**gene_info**(*name*, *fields='name,symbol,refseq,generif,ensembl'*)

Retrieve gene information from biothings client.

Data preparation

utils.cleanup(data[, clean, keep, copy])	Delete not needed attributes.
utils.clean_obs_names(data[, base, ...])	Clean up the obs_names.
utils.merge(adata, ldata[, copy])	Merge two annotated data matrices.
utils.show_proportions(adata[, layers, use_raw])	Proportions of abundances of modalities in layers.

scvelo.utils.cleanup

scvelo.utils.**cleanup**(*data*, *clean='layers'*, *keep=None*, *copy=False*)

Delete not needed attributes.

Parameters

data : AnnData Annotated data matrix.

clean : _Genericalias[layers, obs, var, uns], List[_Genericalias[layers, obs, var, uns]] Which attributes to consider for freeing memory.

keep : str, List[str], None Which attributes to keep.

copy : bool Return a copy instead of writing to adata.

Return type AnnData, None

Returns *Optional[AnnData]* – Returns or updates *adata* with selection of attributes kept.

scvelo.utils.clean_obs_names

scvelo.utils.**clean_obs_names**(*data*, *base='[AGTCBDHKMNRSVWY]'*, *ID_length=12*, *copy=False*)

Clean up the obs_names.

For example an obs_name 'sample1_AGTCdate' is changed to 'AGTC' of the sample 'sample1_date'. The sample name is then saved in obs['sample_batch']. The genetic codes are identified according to according to https://www.neb.com/tools-and-resources/usage-guidelines/the-genetic-code.

Parameters

data : AnnData Annotated data matrix.

base : str Genetic code letters to be identified.

ID_length : int Length of the Genetic Codes in the samples.

copy : bool Return a copy instead of writing to adata.

Return type AnnData, None

Returns

Optional[AnnData] – Returns or updates *adata* with the attributes obs_names: list

updated names of the observations

sample_batch: *.obs* names of the identified sample batches

scvelo.utils.merge

scvelo.utils.**merge**(*adata*, *ldata*, *copy=True*)

Merge two annotated data matrices.

Parameters

adata : AnnData Annotated data matrix (reference data set).

ldata : AnnData Annotated data matrix (to be merged into adata).

copy : bool Boolean flag to manipulate original AnnData or a copy of it.

Return type AnnData, None

Returns Optional[anndata.AnnData] – Returns a AnnData object

scvelo.utils.show_proportions

scvelo.utils.**show_proportions**(*adata*, *layers=None*, *use_raw=True*)

Proportions of abundances of modalities in layers.

The proportions are printed.

Parameters

adata : AnnData Annotated data matrix.

layers : str, None Layers to consider.

use_raw : bool Use initial sizes, i.e., raw data, to determine proportions.

Return type None

Returns *None*

Getters

utils.get_moments(adata[, layer, ...])	Computes moments for a specified layer.
utils.get_transition_matrix(adata[, vkey, ...])	Computes cell-to-cell transition probabilities
utils.get_cell_transitions(adata[, ...])	Simulate cell transitions
utils.get_extrapolated_state(adata[, vkey, ...])	Get extrapolated cell state.

scvelo.utils.get_moments

scvelo.utils.**get_moments**(*adata*, *layer=None*, *second_order=None*, *centered=True*, *mode='connectivities'*)

Computes moments for a specified layer.

First and second order moments. If centered, that corresponds to means and variances across nearest neighbors.

Parameters

adata : *AnnData* Annotated data matrix.

layer : *str* (default: *None*) Key of layer with abundances to consider for moment computation.

second_order : *bool* (default: *None*) Whether to compute second order moments from abundances.

centered : *bool* (default: *True*) Whether to compute centered (=variance) or uncentered second order moments.

mode : *'connectivities'* or *'distances'* (default: *'connectivities'*) Distance metric to use for moment computation.

Returns **Mx** (*first or second order moments*)

scvelo.utils.get_transition_matrix

scvelo.utils.**get_transition_matrix**(*adata*, *vkey='velocity'*, *basis=None*, *backward=False*, *self_transitions=True*, *scale=10*, *perc=None*, *threshold=None*, *use_negative_cosines=False*, *weight_diffusion=0*, *scale_diffusion=1*, *weight_indirect_neighbors=None*, *n_neighbors=None*, *vgraph=None*, *basis_constraint=None*)

Computes cell-to-cell transition probabilities

$$\tilde{\pi}_{ij} = \frac{1}{z_i} \exp(\pi_{ij}/\sigma),$$

from the velocity graph π_{ij}, with row-normalization z_i and kernel width σ (scale parameter $\lambda = \sigma^{-1}$).

Alternatively, use `cellrank.tl.transition_matrix()` to account for uncertainty in the velocity estimates.

Parameters

adata : `AnnData` Annotated data matrix.

vkey : *str* (default: *'velocity'*) Name of velocity estimates to be used.

basis : *str* or *None* (default: *None*) Restrict transition to embedding if specified

backward : *bool* (default: *False*) Whether to use the transition matrix to push forward (*False*) or to pull backward (*True*)

self_transitions : *bool* (default: *True*) Allow transitions from one node to itself.

scale : *float* (default: 10) Scale parameter of gaussian kernel.

perc : *float* between *0* and *100* or *None* (default: *None*) Determines threshold of transitions to include.

use_negative_cosines : *bool* (default: *False*) If True, negatively similar transitions are taken into account.

weight_diffusion : *float* (default: 0) Relative weight to be given to diffusion kernel (Brownian motion)

scale_diffusion : *float* (default: 1) Scale of diffusion kernel.

weight_indirect_neighbors : *float* between *0* and *1* or *None* (default: *None*) Weight to be assigned to indirect neighbors (i.e. neighbors of higher degrees).

n_neighbors : *int* (default: None) Number of nearest neighbors to consider around each cell.

vgraph : csr matrix or *None* (default: *None*) Velocity graph representation to use instead of adata.uns[f'{vkey}_graph'].

Returns *Returns sparse matrix with transition probabilities.*

scvelo.utils.get_cell_transitions

scvelo.utils.**get_cell_transitions**(*adata*, *starting_cell=0*, *basis=None*, *n_steps=100*, *n_neighbors=30*, *backward=False*, *random_state=None*, ***kwargs*)

Simulate cell transitions

Parameters

adata : AnnData Annotated data matrix.

starting_cell : *int* (default: *0*) Index (*int*) or name (`obs_names) of starting cell.

n_steps : *int* (default: *100*) Number of transitions/steps to be simulated.

backward : *bool* (default: *False*) Whether to use the transition matrix to push forward (*False*) or to pull backward (*True*)

random_state : *int* or *None* (default: *None*) Set to *int* for reproducibility, otherwise *None* for a random seed.

****kwargs** To be passed to tl.transition_matrix.

Returns

- *Returns embedding coordinates (if basis is specified),*
- *otherwise return indices of simulated cell transitions.*

scvelo.utils.get_extrapolated_state

scvelo.utils.**get_extrapolated_state**(*adata*, *vkey='velocity'*, *dt=1*, *use_raw=None*, *dropna=True*)

Get extrapolated cell state.

Converters

utils.convert_to_ensembl([gene_names])	Retrieve ensembl IDs from a list of gene names.
utils.convert_to_gene_names([ensembl_names])	Retrieve gene names from ensembl IDs.

scvelo.utils.convert_to_ensembl

scvelo.utils.**convert_to_ensembl**(*gene_names=None*)
Retrieve ensembl IDs from a list of gene names.

scvelo.utils.convert_to_gene_names

scvelo.utils.**convert_to_gene_names**(*ensembl_names=None*)
Retrieve gene names from ensembl IDs.

Least squares and correlation

utils.leastsq(x, y[, fit_offset, perc, ...])	Solves least squares X*b=Y for b.
utils.vcorrcoef(X, y[, mode, axis])	Pearsons/Spearmans correlation coefficients.
utils.test_bimodality(x[, bins, kde, plot])	Test for bimodal distribution.

scvelo.utils.leastsq

scvelo.utils.**leastsq**(*x*, *y*, *fit_offset=False*, *perc=None*, *constraint_positive_offset=True*)
Solves least squares X*b=Y for b.

scvelo.utils.vcorrcoef

scvelo.utils.**vcorrcoef**(*X*, *y*, *mode='pearsons'*, *axis=- 1*)
Pearsons/Spearmans correlation coefficients.

Use Pearsons / Spearmans to test for linear / monotonic relationship.

Parameters

X : *np.ndarray* Data vector or matrix

y : *np.ndarray* Data vector or matrix

mode : 'pearsons' or 'spearmans' (default: *'pearsons'*) Which correlation metric to use.

scvelo.utils.test_bimodality

scvelo.utils.**test_bimodality**(*x*, *bins=30*, *kde=True*, *plot=False*)
Test for bimodal distribution.

4.3.7 Settings

set_figure_params([style, dpi, dpi_save, ...])	Set resolution/size, styling and format of figures.

scvelo.set_figure_params

scvelo.**set_figure_params**(*style='scvelo'*, *dpi=100*, *dpi_save=150*, *frameon=None*, *vector_friendly=True*, *transparent=True*, *fontsize=12*, *figsize=None*, *color_map=None*, *facecolor=None*, *format='pdf'*, *ipython_format='png2x'*)

Set resolution/size, styling and format of figures.

Parameters

style : *str* (default: *None*) Init default values for `matplotlib.rcParams` suited for *scvelo* or *scanpy*. Use *None* for the default matplotlib values.

dpi : *int* (default: *None*) Resolution of rendered figures - affects the size of figures in notebooks.

dpi_save : *int* (default: *None*) Resolution of saved figures. This should typically be higher to achieve publication quality.

frameon : *bool* (default: *None*) Add frames and axes labels to scatter plots.

vector_friendly : *bool* (default: *True*) Plot scatter plots using *png* backend even when exporting as *pdf* or *svg*.

transparent : *bool* (default: *True*) Save figures with transparent back ground. Sets *rcParams['savefig.transparent']*.

fontsize : *int* (default: 14) Set the fontsize for several *rcParams* entries.

figsize : *[float, float]* (default: *None*) Width and height for default figure size.

color_map : *str* (default: *None*) Convenience method for setting the default color map.

facecolor : *str* (default: *None*) Sets backgrounds *rcParams['figure.facecolor']* and *rcParams['axes.facecolor']* to *facecolor*.

format : *{'png', 'pdf', 'svg', etc.} (default: 'pdf')* This sets the default format for saving figures: *file_format_figs*.

ipython_format : list of *str* (default: 'png2x') Only concerns the notebook/IPython environment; see *IPython.core.display.set_matplotlib_formats* for more details.

4.4 Release Notes

4.4.1 Version 0.2.4 Aug 26, 2021

Perspectives:

- Landing page and two notebooks accompanying the perspectives manuscript at MSB.
- New datasets: Gastrulation, bone marrow, and PBMCs.

New capabilities:

- Added vignettes accompanying the NBT manuscript.
- Kinetic simulations with time-dependent rates.
- New arguments for *tl.velocity_embedding_stream* (PR 492).
- Introduced automated code formatting *flake8* and *isort* (PR 360, PR 374).
- *tl.velocity_graph* parallelized (PR 392).

- *legend_align_text* parameter in *pl.scatter* for smart placing of labels without overlapping.
- Save option for *pl.proportions*.

Bugfixes:

- Pinned *sphinx<4.0* and *nbsphinx<0.8.7*.
- Fix IPython import at CLI.

4.4.2 Version 0.2.3 Feb 13, 2021

- *tl.recover_dynamics*: Multicore implementation thanks to M Klein, Y Schaelte, P Weiler
- CI now runs on GitHub Actions

New utility functions:

- *utils.gene_info*: Retrieve gene information from biothings client.
- *utils.convert_to_ensembl* and *utils.convert_to_gene_names*: Converting ensembl IDs into gene names and vice versa.

4.4.3 Version 0.2.2 July 22, 2020

- *tl.paga*: PAGA graph with velocity-directed edges.
- Black code style

4.4.4 Version 0.2.1 May 28, 2020

Bugfixes:

- Correct identification of root cells in *tl.latent_time* thanks to M Lange
- Correct usage of latent_time prior in *tl.paga* thanks to G Lubatti

4.4.5 Version 0.2.0 May 12, 2020

New vignettes:

- RNA velocity basics
- Dynamical Modeling
- Differential Kinetics

Tools:

- *tl.differential_kinetic_test*: introduced a statistical test to detect different kinetic regimes.
- *tl.rank_dynamical_genes*: introduced a gene ranking by cluster-wise likelihoods.
- *tl.paga*: introduced directed PAGA graph

Plotting:

- *pl.scatter* enhancements: linear and polynomical fits, gradient coloring
- *pl.proportions*: Pie and bar chart of spliced/unspliced proprtions.

- *GridSpec*: multiplot environment.

4.4.6 Version 0.1.20 Sep 5, 2019

Tools:

- *tl.recover_dynamics*: introduced a dynamical model inferring the full splicing kinetics, thereby identifying all kinetic rates of transcription, splicing and degradation.
- *tl.recover_latent_time*: infers a shared latent time across all genes based on the learned splicing dynamics.

Plotting:

- *pl.scatter* enhancements: multiplots, rugplot, linear and polynomial fits, density plots, etc.
- *pl.heatmap*: heatmap / clustermap of genes along time coordinate sorted by expression along dynamics.

Preprocessing:

- New attributes in pp.filter_genes: *min_shared_counts* and *min_shared_genes*.
- Added fast neighbor search method: Hierarchical Navigable Small World graphs (HNSW)

4.4.7 Version 0.1.14 Dec 7, 2018

Plotting:

- New attriutes *arrow_length* and *arrow_size* for flexible adjustment of embedded velocities.
- *pl.velocity_graph*: Scatter plot of embedding with cell-to-cell transition connectivities.
- *pl.velocity_embedding_stream*: Streamplot visualization of velocities.
- Improve visualization of embedded single cell velocities (autosize, colors etc.)

Tools:

- *tl.cell_fate*: compute cell-specific terminal state likelihood
- New attribute *approx=True* in *tl.velocity_graph* to enable approximate graph computation by performing cosine correlations on PCA space.

Preprocessing:

- Automatically detect whether data is already preprocessed.

4.4.8 Version 0.1.11 Oct 27, 2018

Plotting:

- *settings.set_figure_params()*: adjust matplotlib defaults for beautified plots
- improved default point and arrow sizes; improved quiver autoscale
- enable direct plotting of

Tools:

- *tl.velocity_confidence*: Added two confidence measures 'velocity_confidence' and 'velocity_confidence_transition'.
- *tl.rank_velocity_genes*: Added functionality to rank genes for velocity characterizing groups using a t-test.

- New attribute *perc* in *tl.velocity* enables extreme quantile fit, e.g. set *perc=95*.
- New attribute *groups* in *tl.velocity* enables velocity estimation only on a subset of the data.
- Improved *tl.transition_matrix* by incorporating self-loops via *self_transitions=True* and state changes that have negative correlation with velocity (opposite direction) via *use_negative_cosines=True*

Utils:

- *utils.merge* to merge to AnnData objects such as already existing AnnData and newly generated Loom File.

4.4.9 Version 0.1.8 Sep 12, 2018

Plotting:

- support saving plots as pdf, png etc.
- support multiple colors and layers
- quiver autoscaling for velocity plots
- attributes added: figsize and dpi

Preprocessing:

- filter_and_normalize() instead of recipe_velocity()
- normalization of layers is done automatically when computing moments

Tools:

- terminal_states: computes root and end points via eigenvalue decomposition thanks to M Lange

4.4.10 Version 0.1.5 Sep 4, 2018

- Support writing loom files
- Support both dense and sparse layers
- Plotting bugfixes
- Added pp.recipe_velocity()

4.4.11 Version 0.1.2 Aug 21, 2018

First alpha release of scvelo.

4.5 References

4.6 Getting Started

Here, you will be briefly guided through the basics of how to use scVelo. Once you are set, the following tutorials go straight into analysis of RNA velocity, latent time, driver identification and many more.

First of all, the input data for scVelo are two count matrices of pre-mature (unspliced) and mature (spliced) abundances, which can be obtained from standard sequencing protocols, using the velocyto or loompy/kallisto counting pipeline.

4.6.1 scVelo workflow at a glance

Import scvelo as:

```
import scvelo as scv
```

For beautified visualization you can change the matplotlib settings to our defaults with:

```
scv.set_figure_params()
```

Read your data

Read your data file (loom, h5ad, csv, . . .) using:

```
adata = scv.read(filename, cache=True)
```

which stores the data matrix (`adata.X`), annotation of cells / observations (`adata.obs`) and genes / variables (`adata.var`), unstructured annotation such as graphs (`adata.uns`) and additional data layers where spliced and unspliced counts are stored (`adata.layers`) .

If you already have an existing preprocessed adata object you can simply merge the spliced/unspliced counts via:

```
ldata = scv.read(filename.loom, cache=True)
adata = scv.utils.merge(adata, ldata)
```

If you do not have a datasets yet, you can still play around using one of the in-built datasets, e.g.:

```
adata = scv.datasets.pancreas()
```

The typical workflow consists of subsequent calls of preprocessing (`scv.pp.*`), analysis tools (`scv.tl.*`) and plotting (`scv.pl.*`).

Basic preprocessing

After basic preprocessing (gene selection and normalization), we compute the first- and second-order moments (means and uncentered variances) for velocity estimation:

```
scv.pp.filter_and_normalize(adata, **params)
scv.pp.moments(adata, **params)
```

Velocity Tools

The core of the software is the efficient and robust estimation of velocities, obtained with:

```
scv.tl.velocity(adata, mode='stochastic', **params)
```

The velocities are vectors in gene expression space obtained by solving a stochastic model of transcriptional dynamics. The solution to the deterministic model is obtained by setting `mode='deterministic'`.

The solution to the dynamical model is obtained by setting `mode='dynamical'`, which requires to run `scv.tl.recover_dynamics(adata, **params)` beforehand.

The velocities are stored in `adata.layers` just like the count matrices.

The velocities are projected into a lower-dimensional embedding by translating them into likely cell transitions. That is, for each velocity vector we find the likely cell transitions that are in accordance with that direction. The probabilities of one cell transitioning into another cell are computed using cosine correlation (between the potential cell transition and the velocity vector) and are stored in a matrix denoted as velocity graph:

```
scv.tl.velocity_graph(adata, **params)
```

Visualization

Finally, the velocities can be projected and visualized in any embedding (e.g. UMAP) on single cell level, as gridlines, or as streamlines:

```
scv.pl.velocity_embedding(adata, basis='umap', **params)
scv.pl.velocity_embedding_grid(adata, basis='umap', **params)
scv.pl.velocity_embedding_stream(adata, basis='umap', **params)
```

For every tool module there is a plotting counterpart, which allows you to examine your results in detail, e.g.:

```
scv.pl.velocity(adata, var_names=['gene_A', 'gene_B'], **params)
scv.pl.velocity_graph(adata, **params)
```

4.7 RNA Velocity Basics

Here you will learn the basics of RNA velocity analysis.

For illustration, it is applied to endocrine development in the pancreas, with lineage commitment to four major fates: , , and -cells. See here for more details. It can be applied to your own data along the same lines.

The notebook is also available at Google Colab and nbviewer.

[ ]:
```
# update to the latest version, if not done yet.
!pip install scvelo --upgrade --quiet
```

[1]:
```
import scvelo as scv
scv.logging.print_version()
```

```
Running scvelo 0.2.0 (python 3.8.2) on 2020-05-16 12:55.
```

[2]:
```
scv.settings.verbosity = 3  # show errors(0), warnings(1), info(2), hints(3)
scv.settings.presenter_view = True  # set max width size for presenter view
scv.set_figure_params('scvelo')  # for beautified visualization
```

4.7.1 Load the Data

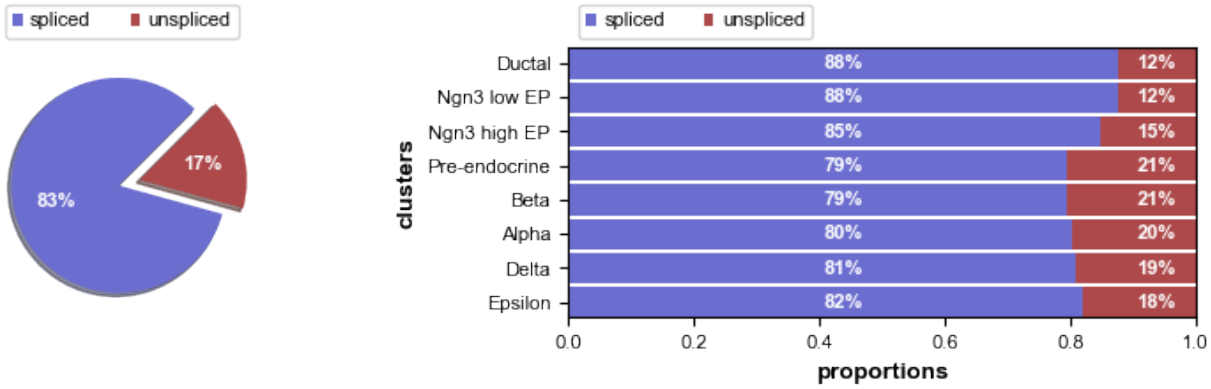
The analysis is based on the in-built pancreas data. To run velocity analysis on your own data, read your file (loom, h5ad, csv ...) to an AnnData object with `adata = scv.read('path/file.loom', cache=True)`. If you want to merge your loom file into an already existing AnnData object, use `scv.utils.merge(adata, adata_loom)`.

[3]:
```
adata = scv.datasets.pancreas()
adata
```

[3]:
```
AnnData object with n_obs × n_vars = 3696 × 27998
    obs: 'clusters_coarse', 'clusters', 'S_score', 'G2M_score'
    var: 'highly_variable_genes'
    uns: 'clusters_coarse_colors', 'clusters_colors', 'day_colors', 'neighbors', 'pca'
    obsm: 'X_pca', 'X_umap'
    layers: 'spliced', 'unspliced'
```

scVelo is based on `adata`, an object that stores a data matrix `adata.X`, annotation of observations `adata.obs`, variables `adata.var`, and unstructured annotations `adata.uns`. Names of observations and variables can be accessed via `adata.obs_names` and `adata.var_names`, respectively. AnnData objects can be sliced like dataframes, for example, `adata_subset = adata[:, list_of_gene_names]`. For more details, see the anndata docs.

[4]:
```
scv.pl.proportions(adata)
```



Here, the proportions of spliced/unspliced counts are displayed. Depending on the protocol used (Drop-Seq, Smart-Seq), we typically have between 10%-25% of unspliced molecules containing intronic sequences. We also advice you to examine the variations on cluster level to verify consistency in splicing efficiency. Here, we find variations as expected, with slightly lower unspliced proportions at cycling ductal cells, then higher proportion at cell fate commitment in Ngn3-high and Pre-endocrine cells where many genes start to be transcribed.

4.7.2 Preprocess the Data

Preprocessing requisites consist of **gene selection** by detection (with a minimum number of counts) and high variability (dispersion), **normalizing** every cell by its total size and **logarithmizing** X. Filtering and normalization is applied in the same vein to spliced/unspliced counts and X. Logarithmizing is only applied to X. If X is already preprocessed from former analysis, it will not be touched.

All of this is summarized in a single function `scv.pp.filter_and_normalize`, which essentially runs the following:

```
scv.pp.filter_genes(adata, min_shared_counts=20)
scv.pp.normalize_per_cell(adata)
scv.pp.filter_genes_dispersion(adata, n_top_genes=2000)
scv.pp.log1p(adata)
```

Further, we need the first and second order moments (means and uncentered variances) computed among nearest neighbors in PCA space, summarized in `scv.pp.moments`, which internally computes `scv.pp.pca` and `scv.pp.neighbors`. First order is needed for deterministic velocity estimation, while stochastic estimation also requires second order moments.

[5]:
```
scv.pp.filter_and_normalize(adata, min_shared_counts=20, n_top_genes=2000)
scv.pp.moments(adata, n_pcs=30, n_neighbors=30)
```

```
Filtered out 20801 genes that are detected in less than 20 counts (shared).
Normalized count data: X, spliced, unspliced.
Logarithmized X.
computing neighbors
    finished (0:00:03) --> added
    'distances' and 'connectivities', weighted adjacency matrices (adata.obsp)
computing moments based on connectivities
    finished (0:00:00) --> added
    'Ms' and 'Mu', moments of spliced/unspliced abundances (adata.layers)
```

Further preprocessing (such as batch effect correction) may be used to remove unwanted sources of variability. See the best practices for further details. Note, that any additional preprocessing step only affects X and is not applied to spliced/unspliced counts.

4.7.3 Estimate RNA velocity

Velocities are vectors in gene expression space and represent the direction and speed of movement of the individual cells. The velocities are obtained by modeling transcriptional dynamics of splicing kinetics, either stochastically (default) or deterministically (by setting `mode='deterministic'`). For each gene, a steady-state-ratio of premature (unspliced) and mature (spliced) mRNA counts is fitted, which constitutes a constant transcriptional state. Velocities are then obtained as residuals from this ratio. Positive velocity indicates that a gene is up-regulated, which occurs for cells that show higher abundance of unspliced mRNA for that gene than expected in steady state. Conversely, negative velocity indicates that a gene is down-regulated.

The solution to the full dynamical model is obtained by setting `mode='dynamical'`, which requires to run `scv.tl.recover_dynamics(adata)` beforehand. We will elaborate more on the dynamical model in the next tutorial.

[6]:
```
scv.tl.velocity(adata)
```

```
computing velocities
    finished (0:00:01) --> added
    'velocity', velocity vectors for each individual cell (adata.layers)
```

The computed velocities are stored in `adata.layers` just like the count matrices.

The combination of velocities across genes can then be used to estimate the future state of an individual cell. In order to project the velocities into a lower-dimensional embedding, transition probabilities of cell-to-cell transitions are estimated. That is, for each velocity vector we find the likely cell transitions that are accordance with that direction. The transition probabilities are computed using cosine correlation between the potential cell-to-cell transitions and the velocity vector, and are stored in a matrix denoted as velocity graph. The resulting velocity graph has dimension $n_{obs} \times n_{obs}$ and summarizes the possible cell state changes that are well explained through the velocity vectors (for runtime speedup it can also be computed on reduced PCA space by setting `approx=True`).

[7]:
```
scv.tl.velocity_graph(adata)
```

```
computing velocity graph
    finished (0:00:10) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
```

For a variety of applications, the velocity graph can be converted to a transition matrix by applying a Gaussian kernel to transform the cosine correlations into actual transition probabilities. You can access the Markov transition matrix via `scv.utils.get_transition_matrix`.

As mentioned, it is internally used to project the velocities into a low-dimensional embedding by applying the mean transition with respect to the transition probabilities, obtained with `scv.tl.velocity_embedding`. Further, we can trace cells along the Markov chain to their origins and potential fates, thereby getting root cells and end points within a trajectory, obtained via `scv.tl.terminal_states`.

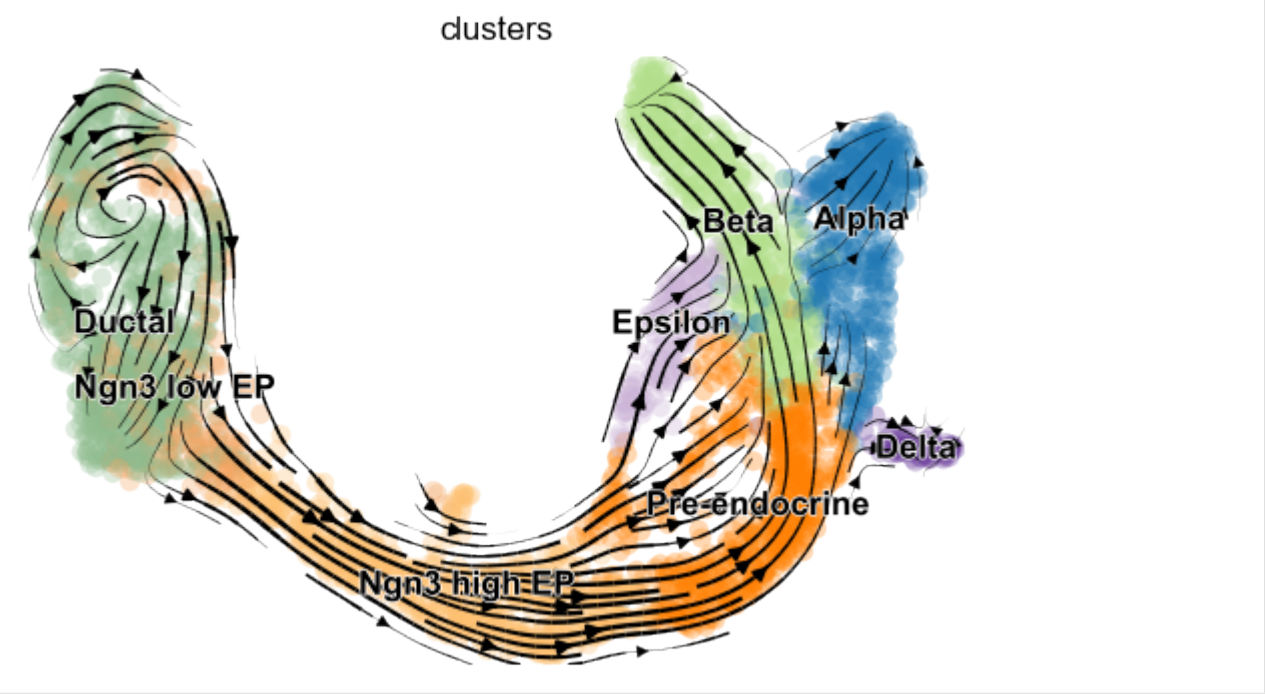
4.7.4 Project the velocities

Finally, the velocities are projected onto any embedding, specified by `basis`, and visualized in one of these ways: - on cellular level with `scv.pl.velocity_embedding`, - as gridlines with `scv.pl.velocity_embedding_grid`, - or as streamlines with `scv.pl.velocity_embedding_stream`.

Note, that the data has an already pre-computed UMAP embedding, and annotated clusters. When applying to your own data, these can be obtained with `scv.tl.umap` and `scv.tl.louvain`. For more details, see the scanpy tutorial. Further, all plotting functions are defaulted to using `basis='umap'` and `color='clusters'`, which you can set accordingly.

[8]:
```
scv.pl.velocity_embedding_stream(adata, basis='umap')
```

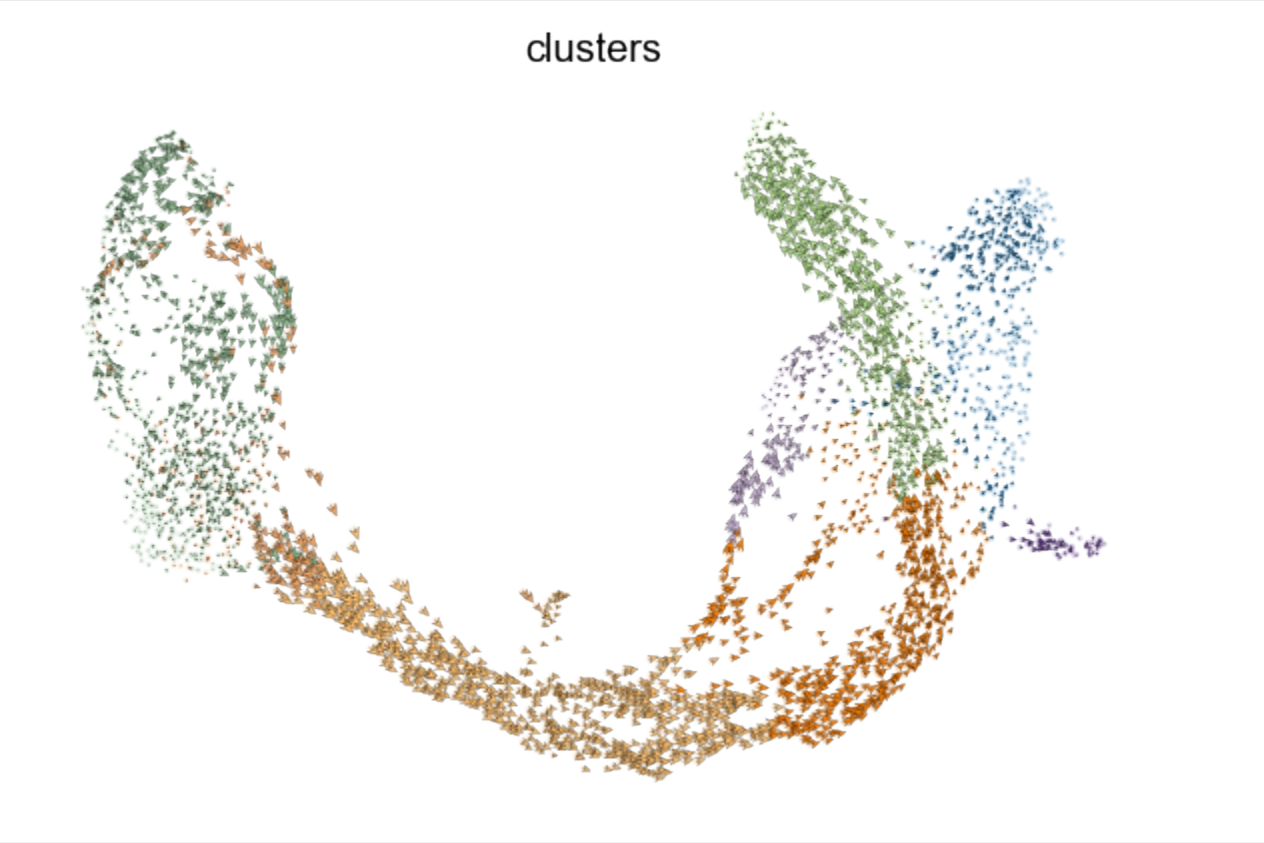
```
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
```



The velocity vector field displayed as streamlines yields fine-grained insights into the developmental processes. It accurately delineates the cycling population of ductal cells and endocrine progenitors. Further, it illuminates cell states of lineage commitment, cell-cycle exit, and endocrine cell differentiation.

The most fine-grained resolution of the velocity vector field we get at single-cell level, with each arrow showing the direction and speed of movement of an individual cell. That reveals, e.g., the early endocrine commitment of Ngn3-cells (yellow) and a clear-cut difference between near-terminal -cells (blue) and transient -cells (green).

[9]:
```
scv.pl.velocity_embedding(adata, arrow_length=3, arrow_size=2, dpi=120)
```



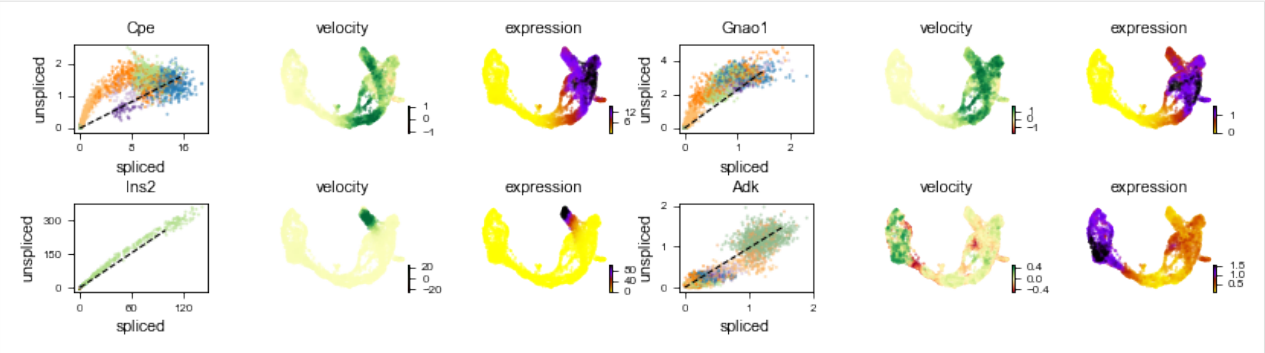
4.7.5 Interprete the velocities

This is perhaps the most important part as we advise the user not to limit biological conclusions to the projected velocities, but to examine individual gene dynamics via phase portraits to understand how inferred directions are supported by particular genes.

See the gif here to get an idea of how to interpret a spliced vs. unspliced phase portrait. Gene activity is orchestrated by transcriptional regulation. Transcriptional induction for a particular gene results in an increase of (newly transcribed) precursor unspliced mRNAs while, conversely, repression or absence of transcription results in a decrease of unspliced mRNAs. Spliced mRNAs is produced from unspliced mRNA and follows the same trend with a time lag. Time is a hidden/latent variable. Thus, the dynamics needs to be inferred from what is actually measured: spliced and unspliced mRNAs as displayed in the phase portrait.

Now, let us examine the phase portraits of some marker genes, visualized with `scv.pl.velocity(adata, gene_names)` or `scv.pl.scatter(adata, gene_names)`.

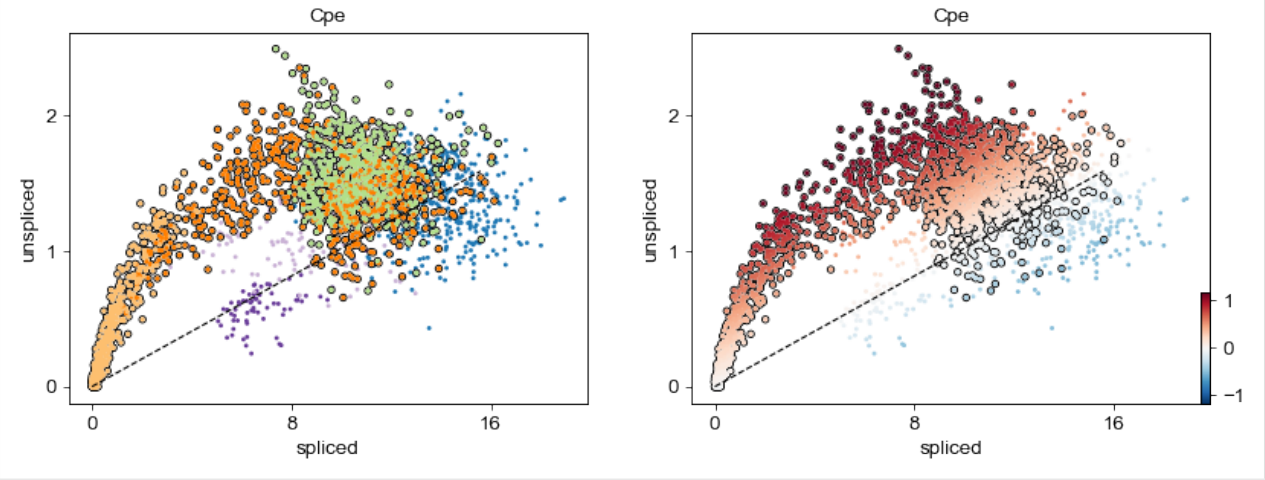
[10]:
```
scv.pl.velocity(adata, ['Cpe',  'Gnao1', 'Ins2', 'Adk'], ncols=2)
```



The black line corresponds to the estimated 'steady-state' ratio, i.e. the ratio of unspliced to spliced mRNA abundance which is in a constant transcriptional state. RNA velocity for a particular gene is determined as the residual, i.e. how much an observation deviates from that steady-state line. Positive velocity indicates that a gene is up-regulated, which occurs for cells that show higher abundance of unspliced mRNA for that gene than expected in steady state. Conversely, negative velocity indicates that a gene is down-regulated.

For instance *Cpe* explains the directionality in the up-regulated Ngn3 (yellow) to Pre-endocrine (orange) to -cells (green), while *Adk* explains the directionality in the down-regulated Ductal (dark green) to Ngn3 (yellow) to the remaining endocrine cells.

[11]:
```
scv.pl.scatter(adata, 'Cpe', color=['clusters', 'velocity'],
               add_outline='Ngn3 high EP, Pre-endocrine, Beta')
```



4.7.6 Identify important genes

We need a systematic way to identify genes that may help explain the resulting vector field and inferred lineages. To do so, we can test which genes have cluster-specific differential velocity expression, being siginificantly higher/lower compared to the remaining population. The module `scv.tl.rank_velocity_genes` runs a differential velocity t-test and outpus a gene ranking for each cluster. Thresholds can be set (e.g. `min_corr`) to restrict the test on a selection of gene candidates.

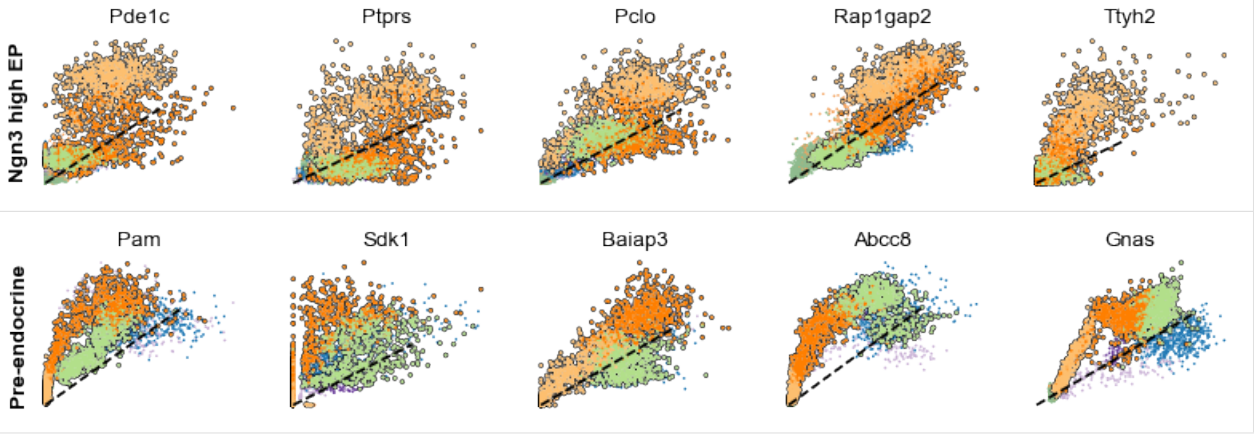
[12]:
```
scv.tl.rank_velocity_genes(adata, groupby='clusters', min_corr=.3)

df = scv.DataFrame(adata.uns['rank_velocity_genes']['names'])
df.head()
```

```
ranking velocity genes
    finished (0:00:02) --> added
    'rank_velocity_genes', sorted scores by group ids (adata.uns)
    'spearmans_score', spearmans correlation scores (adata.var)
```

[12]:

	Ductal	Ngn3 low EP	Ngn3 high EP	Pre-endocrine	Beta	Alpha	Delta	Epsilon
0	Notch2	Ptpn3	Pde1c	Pam	Pax6	Zcchc16	Zdbf2	Tmcc3
1	Sox5	Hacd1	Ptprs	Sdk1	Unc5c	Nlgn1	Spock3	Heg1
2	Krt19	Hspa8	Pclo	Baiap3	Nnat	Nell1	Akr1c19	Gpr179
3	Hspa8	Gm8113	Rap1gap2	Abcc8	Tmem108	Prune2	Ptprt	Ica1
4	Ano6	Kcnq1	Ttyh2	Gnas	Ptprt	Ksr2	Snap25	Ncoa7

[13]:
```
kwargs = dict(frameon=False, size=10, linewidth=1.5,
              add_outline='Ngn3 high EP, Pre-endocrine, Beta')

scv.pl.scatter(adata, df['Ngn3 high EP'][:5], ylabel='Ngn3 high EP', **kwargs)
scv.pl.scatter(adata, df['Pre-endocrine'][:5], ylabel='Pre-endocrine', **kwargs)
```



The genes *Ptprs, Pclo, Pam, Abcc8, Gnas*, for instance, support the directionality from **Ngn3 high EP** (yellow) to **Pre-endocrine** (orange) to **Beta** (green).

4.7.7 Velocities in cycling progenitors

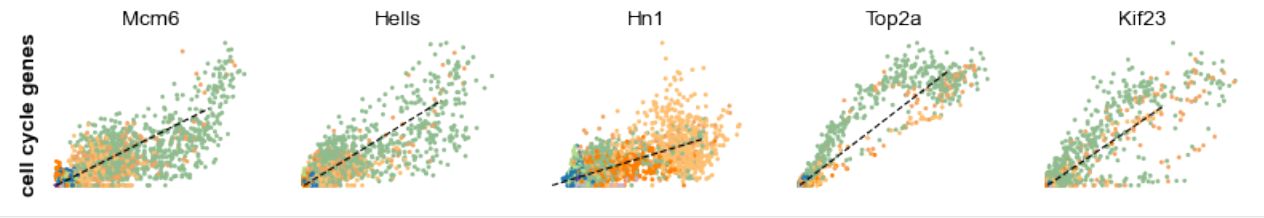
The cell cycle detected by RNA velocity, is biologically affirmed by cell cycle scores (standardized scores of mean expression levels of phase marker genes).

[14]:
```
scv.tl.score_genes_cell_cycle(adata)
scv.pl.scatter(adata, color_gradients=['S_score', 'G2M_score'], smooth=True, perc=[5,␣
↪95])
```

```
calculating cell cycle phase
-->     'S_score' and 'G2M_score', scores of cell cycle phases (adata.obs)
```

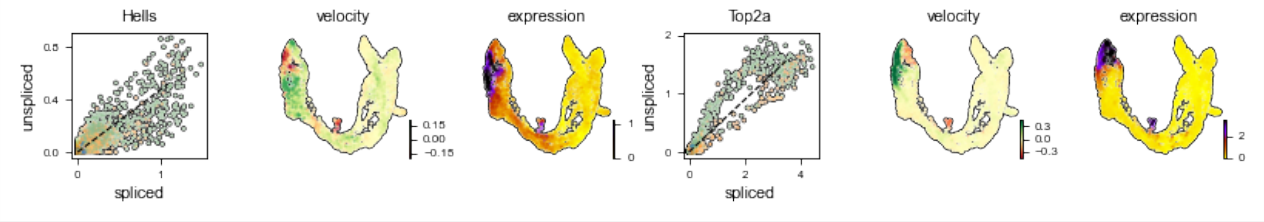
For the cycling Ductal cells, we may screen through S and G2M phase markers. The previous module also computed a spearmans correlation score, which we can use to rank/sort the phase marker genes to then display their phase portraits.

[15]:
```
s_genes, g2m_genes = scv.utils.get_phase_marker_genes(adata)
s_genes = scv.get_df(adata[:, s_genes], 'spearmans_score', sort_values=True).index
g2m_genes = scv.get_df(adata[:, g2m_genes], 'spearmans_score', sort_values=True).index

kwargs = dict(frameon=False, ylabel='cell cycle genes')
scv.pl.scatter(adata, list(s_genes[:2]) + list(g2m_genes[:3]), **kwargs)
```



Particularly *Hells* and *Top2a* are well-suited to explain the vector field in the cycling progenitors. *Top2a* gets assigned a high velocity shortly before it actually peaks in the G2M phase. There, the negative velocity then perfectly matches the immediately following down-regulation.

[16]:
```
scv.pl.velocity(adata, ['Hells', 'Top2a'], ncols=2, add_outline=True)
```

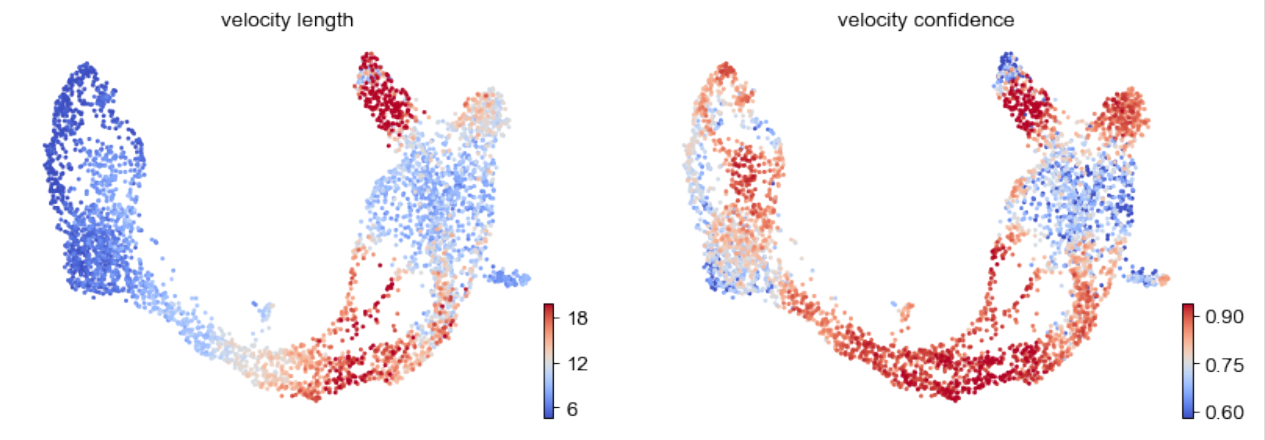


4.7.8 Speed and coherence

Two more useful stats: - The speed or rate of differentiation is given by the length of the velocity vector. - The coherence of the vector field (i.e., how a velocity vector correlates with its neighboring velocities) provides a measure of confidence.

[17]:
```
scv.tl.velocity_confidence(adata)
keys = 'velocity_length', 'velocity_confidence'
scv.pl.scatter(adata, c=keys, cmap='coolwarm', perc=[5, 95])
```

```
--> added 'velocity_length' (adata.obs)
--> added 'velocity_confidence' (adata.obs)
--> added 'velocity_confidence_transition' (adata.obs)
```



These provide insights where cells differentiate at a slower/faster pace, and where the direction is un-/determined.

On cluster-level, we find that differentiation substantially speeds up after cell cycle exit (Ngn3 low EP), keeping the pace during Beta cell production while slowing down during Alpha cell production.

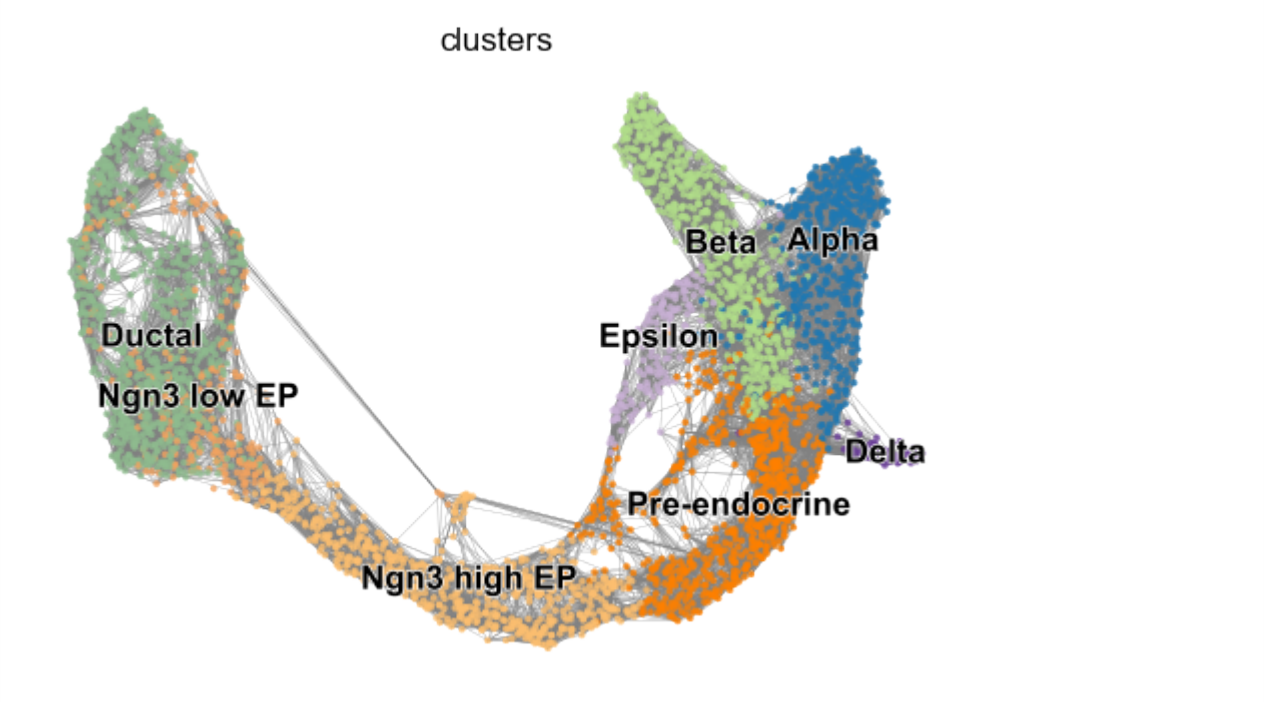
[18]:
```
df = adata.obs.groupby('clusters')[keys].mean().T
df.style.background_gradient(cmap='coolwarm', axis=1)
```

[18]:
```
<pandas.io.formats.style.Styler at 0x1293e3550>
```

4.7.9 Velocity graph and pseudotime

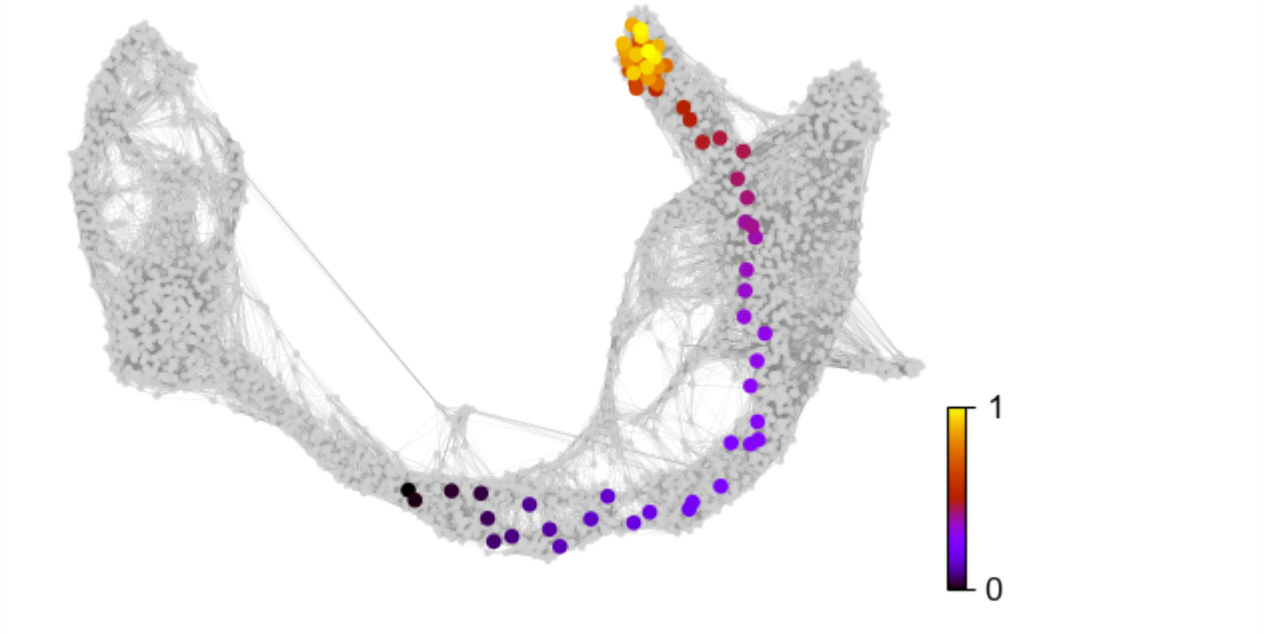
We can visualize the velocity graph to portray all velocity-inferred cell-to-cell connections/transitions. It can be confined to high-probability transitions by setting a `threshold`. The graph, for instance, indicates two phases of Epsilon cell production, coming from early and late Pre-endocrine cells.

[19]:
```
scv.pl.velocity_graph(adata, threshold=.1)
```



Further, the graph can be used to draw descendents/anscestors coming from a specified cell. Here, a pre-endocrine cell is traced to its potential fate.

[20]:
```
x, y = scv.utils.get_cell_transitions(adata, basis='umap', starting_cell=70)
ax = scv.pl.velocity_graph(adata, c='lightgrey', edge_width=.05, show=False)
ax = scv.pl.scatter(adata, x=x, y=y, s=120, c='ascending', cmap='gnuplot', ax=ax)
```



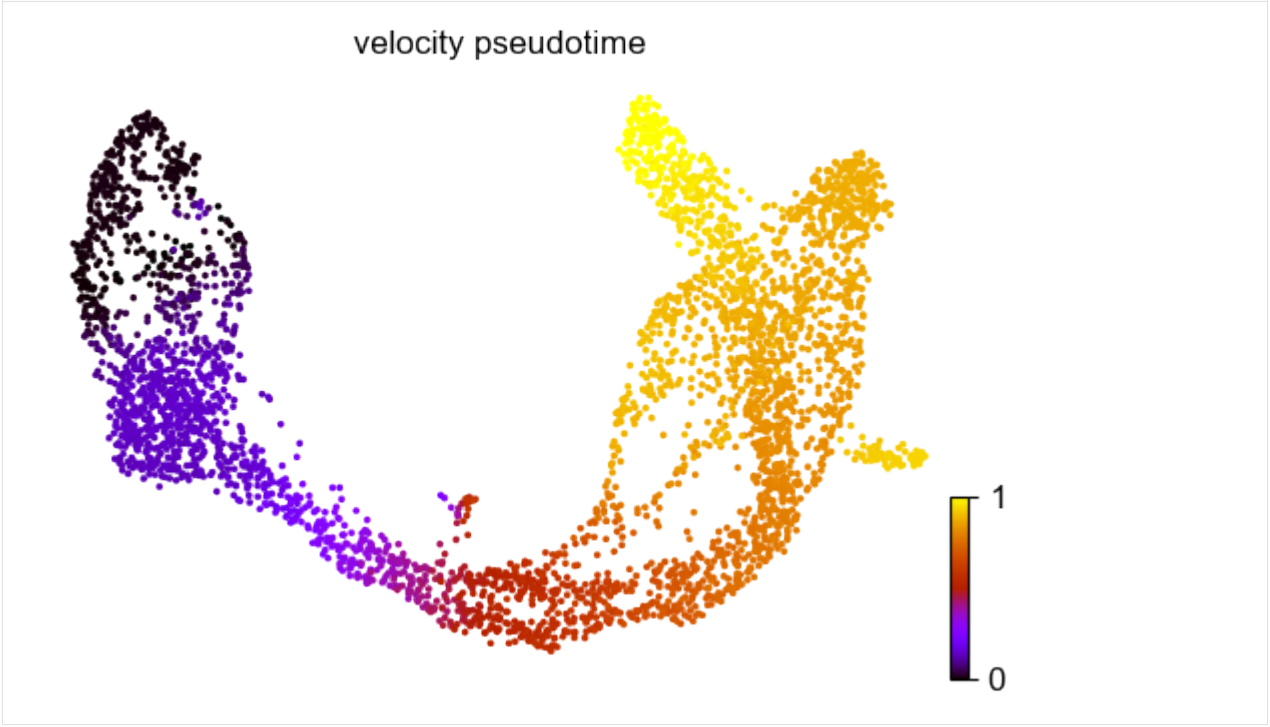
Finally, based on the velocity graph, a *velocity pseudotime* can be computed. After inferring a distribution over root cells from the graph, it measures the average number of steps it takes to reach a cell after walking along the graph

starting from the root cells.

Contrarily to diffusion pseudotime, it implicitly infers the root cells and is based on the directed velocity graph instead of the similarity-based diffusion kernel.

[21]:
```
scv.tl.velocity_pseudotime(adata)
scv.pl.scatter(adata, color='velocity_pseudotime', cmap='gnuplot')
```

```
computing terminal states
    identified 2 regions of root cells and 1 region of end points
    finished (0:00:00) --> added
    'root_cells', root cells of Markov diffusion process (adata.obs)
    'end_points', end points of Markov diffusion process (adata.obs)
```



4.7.10 PAGA velocity graph

PAGA graph abstraction has benchmarked as top-performing method for trajectory inference. It provides a graph-like map of the data topology with weighted edges corresponding to the connectivity between two clusters. Here, PAGA is extended by velocity-inferred directionality.

[ ]:
```
# PAGA requires to install igraph, if not done yet.
!pip install python-igraph --upgrade --quiet
```

[22]:
```
# this is needed due to a current bug - bugfix is coming soon.
adata.uns['neighbors']['distances'] = adata.obsp['distances']
adata.uns['neighbors']['connectivities'] = adata.obsp['connectivities']

scv.tl.paga(adata, groups='clusters')
df = scv.get_df(adata, 'paga/transitions_confidence', precision=2).T
df.style.background_gradient(cmap='Blues').format('{:.2g}')
```

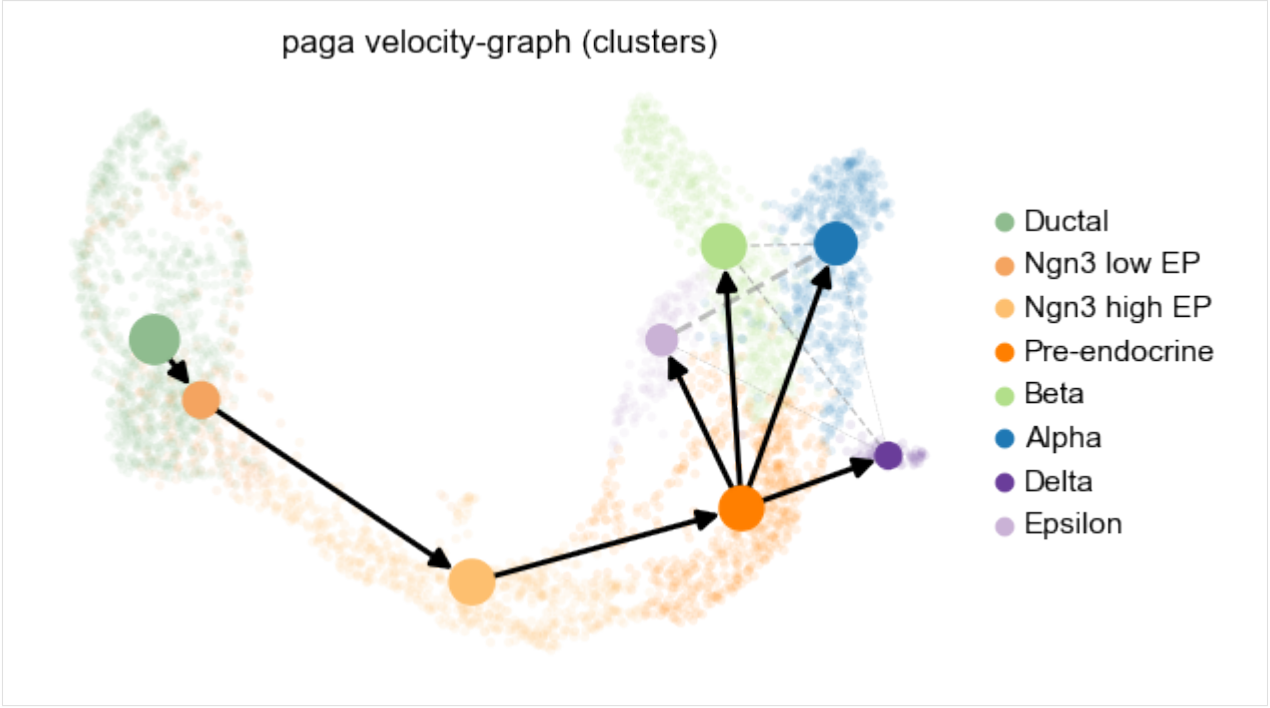
```
running PAGA
    finished (0:00:01) --> added
    'paga/transitions_confidence', connectivities adjacency (adata.uns)
    'paga/connectivities', connectivities adjacency (adata.uns)
    'paga/connectivities_tree', connectivities subtree (adata.uns)
```

[22]:
```
<pandas.io.formats.style.Styler at 0x12b7abbb0>
```

This reads from left/row to right/column, thus e.g. assigning a confident transition from Ductal to Ngn3 low EP.

This table can be summarized by a directed graph superimposed onto the UMAP embedding.

[23]:
```
scv.pl.paga(adata, basis='umap', size=50, alpha=.1,
            min_edge_width=2, node_size_scale=1.5)
```



4.8 Dynamical Modeling

Here, we use the generalized dynamical model to solve the full transcriptional dynamics.

That yields several additional insights such as latent time and identification of putative driver genes.

As in the previous tutorial, it is illustratively applied to endocrine development in the pancreas.

[ ]:
```
# update to the latest version, if not done yet.
!pip install scvelo --upgrade --quiet
```

[1]:
```
import scvelo as scv
scv.logging.print_version()
```

```
Running scvelo 0.2.0 (python 3.8.2) on 2020-05-15 00:27.
```

[2]:
```
scv.settings.verbosity = 3  # show errors(0), warnings(1), info(2), hints(3)
scv.settings.presenter_view = True  # set max width size for presenter view
scv.settings.set_figure_params('scvelo')  # for beautified visualization
```

4.8.1 Prepare the Data

Processing consists of gene selection, normalizing by total size, logarithmizing X, and computing moments for velocity estimation. See the previous tutorial for further explanation.

[3]:
```
adata = scv.datasets.pancreas()
```

[4]:
```
scv.pp.filter_and_normalize(adata, min_shared_counts=20, n_top_genes=2000)
scv.pp.moments(adata, n_pcs=30, n_neighbors=30)
```

```
Filtered out 20801 genes that are detected in less than 20 counts (shared).
Normalized count data: X, spliced, unspliced.
Logarithmized X.
computing neighbors
    finished (0:00:03) --> added
    'distances' and 'connectivities', weighted adjacency matrices (adata.obsp)
computing moments based on connectivities
    finished (0:00:00) --> added
    'Ms' and 'Mu', moments of spliced/unspliced abundances (adata.layers)
```

4.8.2 Dynamical Model

We run the dynamical model to learn the full transcriptional dynamics of splicing kinetics.

It is solved in a likelihood-based expectation-maximization framework, by iteratively estimating the parameters of reaction rates and latent cell-specific variables, i.e. transcriptional state and cell-internal latent time. It thereby aims to learn the unspliced/spliced phase trajectory for each gene.

[5]:
```
scv.tl.recover_dynamics(adata)
```

```
recovering dynamics
    finished (0:13:31) --> added
    'fit_pars', fitted parameters for splicing dynamics (adata.var)
```

[6]:
```
scv.tl.velocity(adata, mode='dynamical')
scv.tl.velocity_graph(adata)
```

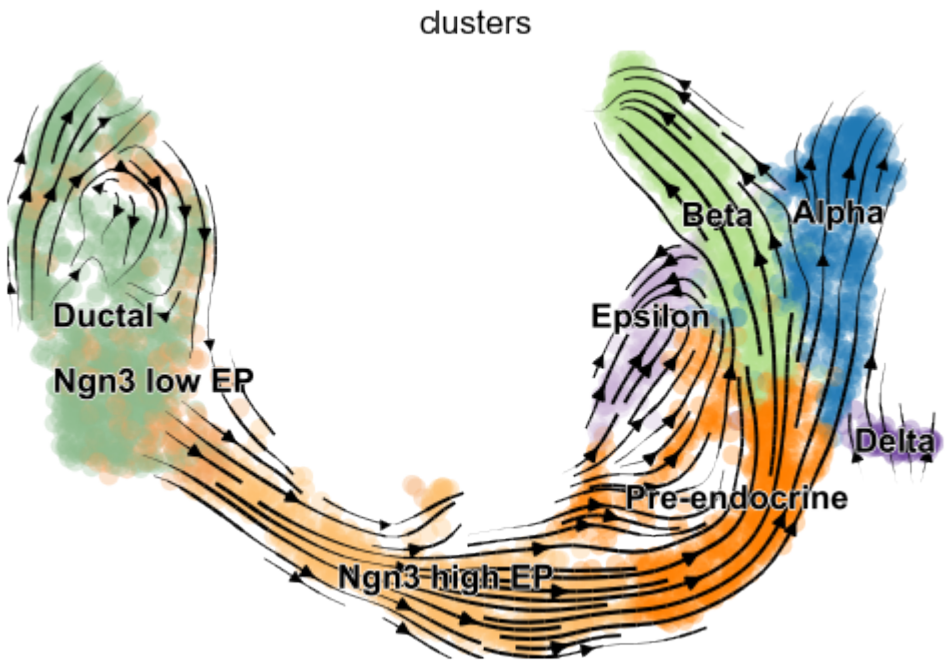
```
computing velocities
    finished (0:00:04) --> added
    'velocity', velocity vectors for each individual cell (adata.layers)
computing velocity graph
    finished (0:00:08) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
```

Running the dynamical model can take a while. Hence, you may want to store the results for re-use, with `adata.write('data/pancreas.h5ad'`, which can later be read with `adata = scv.read('data/pancreas.h5ad')`.

[7]:
```
#adata.write('data/pancreas.h5ad', compression='gzip')
#adata = scv.read('data/pancreas.h5ad')
```

[8]:
```
scv.pl.velocity_embedding_stream(adata, basis='umap')
```

```
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
```

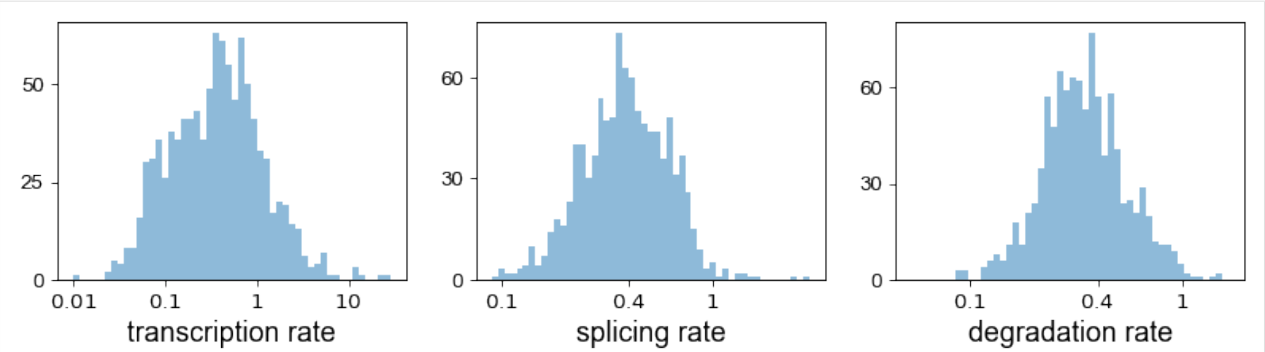


4.8.3 Kinetic rate paramters

The rates of RNA transcription, splicing and degradation are estimated without the need of any experimental data.

They can be useful to better understand the cell identity and phenotypic heterogeneity.

[9]:
```
df = adata.var
df = df[(df['fit_likelihood'] > .1) & df['velocity_genes'] == True]

kwargs = dict(xscale='log', fontsize=16)
with scv.GridSpec(ncols=3) as pl:
    pl.hist(df['fit_alpha'], xlabel='transcription rate', **kwargs)
    pl.hist(df['fit_beta'] * df['fit_scaling'], xlabel='splicing rate', xticks=[.1, .
↪4, 1], **kwargs)
    pl.hist(df['fit_gamma'], xlabel='degradation rate', xticks=[.1, .4, 1], **kwargs)

scv.get_df(adata, 'fit*', dropna=True).head()
```



[9]:

index	fit_r2	fit_alpha	fit_beta	fit_gamma	fit_t_	fit_scaling
Sntg1	0.401981	0.015726	0.005592	0.088792	23.404254	42.849447
Sbspon	0.624803	0.464865	2.437113	0.379645	3.785993	0.154771
Mcm3	0.292389	3.096367	39.995796	0.638543	2.049463	0.013943
Fam135a	0.384662	0.172335	0.118088	0.204538	11.239574	1.124040
Adgrb3	0.384552	0.046828	0.006750	0.196856	6.992542	71.850736

index	fit_std_u	fit_std_s	fit_likelihood	fit_u0	fit_s0
Sntg1	1.029644	0.030838	0.406523	0.0	0.0
Sbspon	0.058587	0.178859	0.252135	0.0	0.0
Mcm3	0.016253	0.673142	0.228207	0.0	0.0
Fam135a	0.356525	0.149868	0.283343	0.0	0.0
Adgrb3	2.153206	0.030417	0.250195	0.0	0.0

index	fit_pval_steady	fit_steady_u	fit_steady_s	fit_variance
Sntg1	0.159472	2.470675	0.094304	0.149138
Sbspon	0.182088	0.164805	0.430623	0.674312
Mcm3	0.467683	0.051432	1.927742	0.687468
Fam135a	0.387921	1.345830	0.393197	0.671096
Adgrb3	0.068851	5.214500	0.093570	0.556878

index	fit_alignment_scaling
Sntg1	5.355590
Sbspon	1.193015
Mcm3	0.887607
Fam135a	3.390194
Adgrb3	1.893389

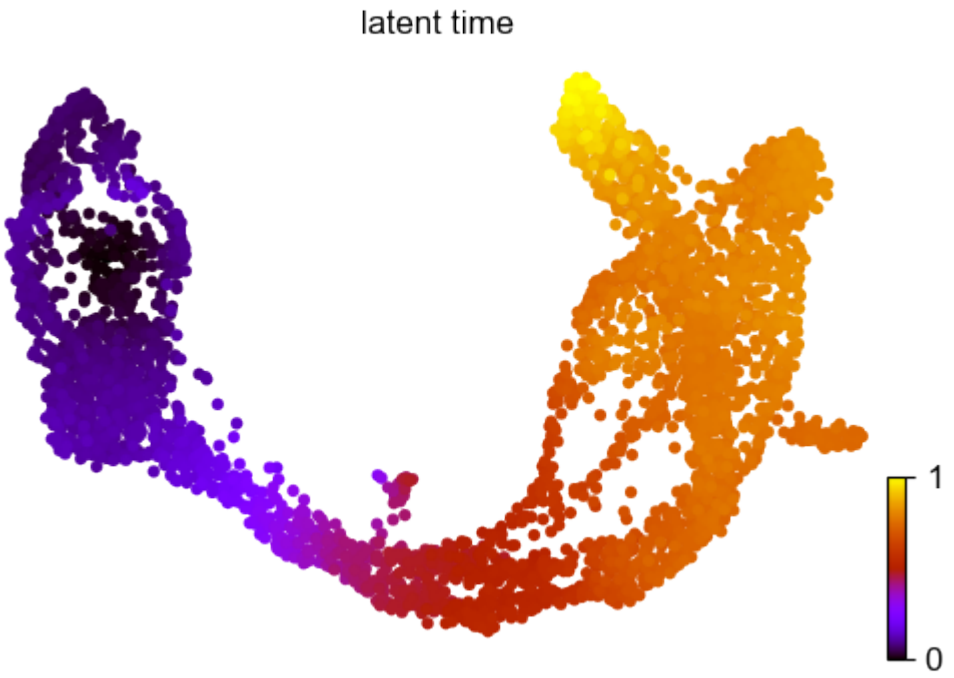
The estimated gene-specific parameters comprise rates of transription (`fit_alpha`), splicing (`fit_beta`), degradation (`fit_gamma`), switching time point (`fit_t_`), a scaling parameter to adjust for under-represented unspliced reads (`fit_scaling`), standard deviation of unspliced and spliced reads (`fit_std_u`, `fit_std_s`), the gene likelihood (`fit_likelihood`), inferred steady-state levels (`fit_steady_u`, `fit_steady_s`) with their corresponding p-values (`fit_pval_steady_u`, `fit_pval_steady_s`), the overall model variance (`fit_variance`), and a scaling factor to align the gene-wise latent times to a universal, gene-shared latent time (`fit_alignment_scaling`).

4.8.4 Latent time

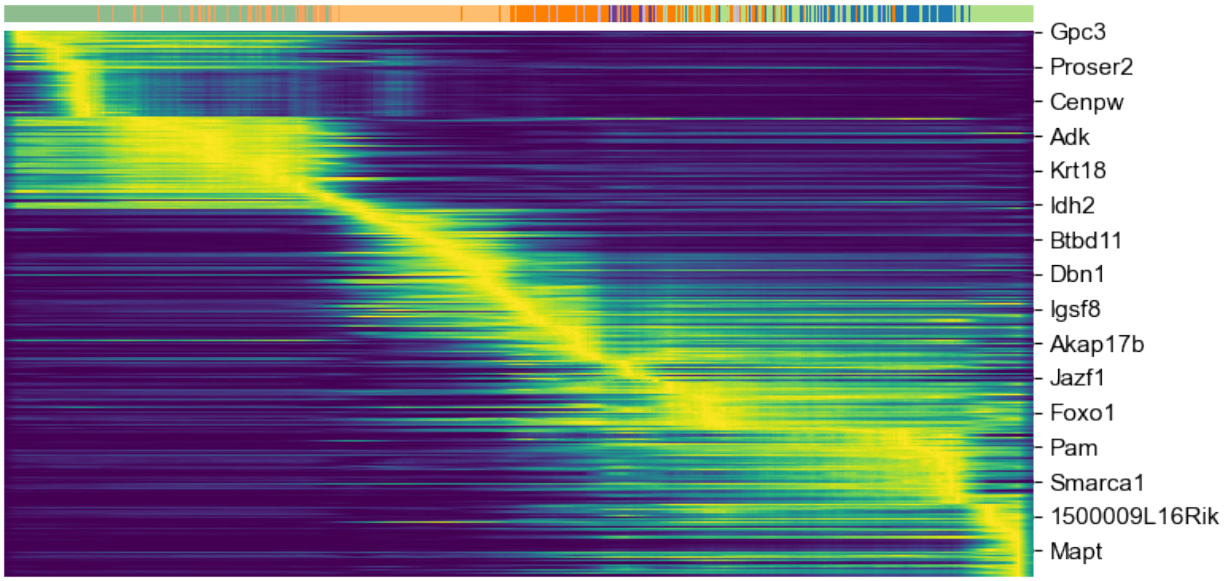
The dynamical model recovers the latent time of the underlying cellular processes. This latent time represents the cell's internal clock and approximates the real time experienced by cells as they differentiate, based only on its transcriptional dynamics.

[10]:
```
scv.tl.latent_time(adata)
scv.pl.scatter(adata, color='latent_time', color_map='gnuplot', size=80)
```

```
computing terminal states
    identified 2 regions of root cells and 1 region of end points
    finished (0:00:00) --> added
    'root_cells', root cells of Markov diffusion process (adata.obs)
    'end_points', end points of Markov diffusion process (adata.obs)
computing latent time
    finished (0:00:02) --> added
    'latent_time', shared time (adata.obs)
```



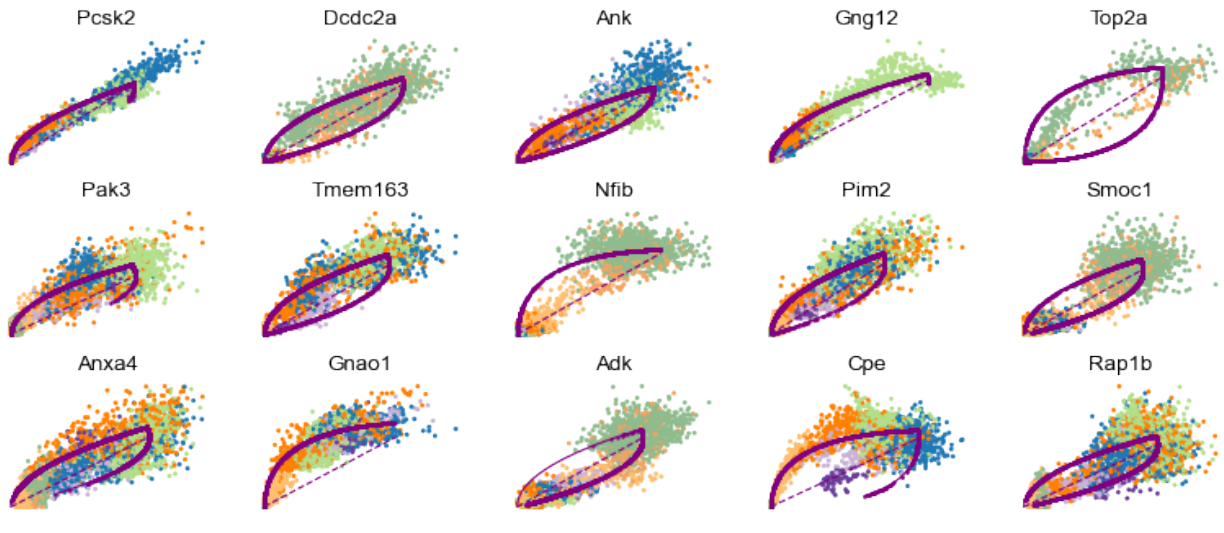
[11]:
```
top_genes = adata.var['fit_likelihood'].sort_values(ascending=False).index[:300]
scv.pl.heatmap(adata, var_names=top_genes, sortby='latent_time', col_color='clusters',
↪ n_convolve=100)
```



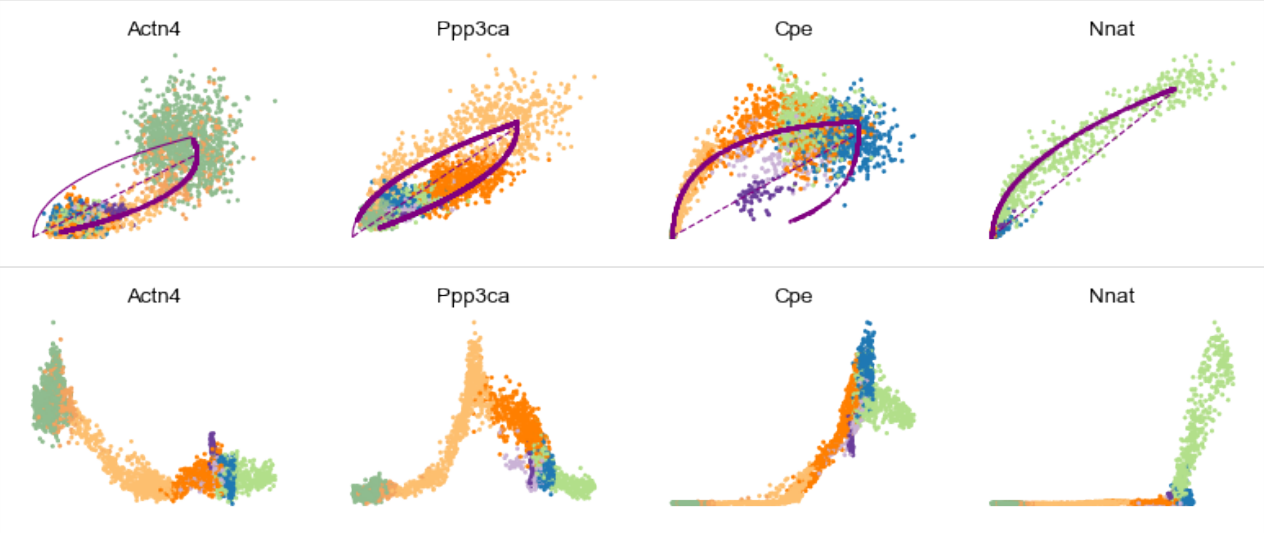
4.8.5 Top-likelihood genes

Driver genes display pronounced dynamic behavior and are systematically detected via their characterization by high likelihoods in the dynamic model.

[12]:
```
top_genes = adata.var['fit_likelihood'].sort_values(ascending=False).index
scv.pl.scatter(adata, basis=top_genes[:15], ncols=5, frameon=False)
```



[13]:
```
var_names = ['Actn4', 'Ppp3ca', 'Cpe', 'Nnat']
scv.pl.scatter(adata, var_names, frameon=False)
scv.pl.scatter(adata, x='latent_time', y=var_names, frameon=False)
```



4.8.6 Cluster-specific top-likelihood genes

Moreover, partial gene likelihoods can be computed for a each cluster of cells to enable cluster-specific identification of potential drivers.

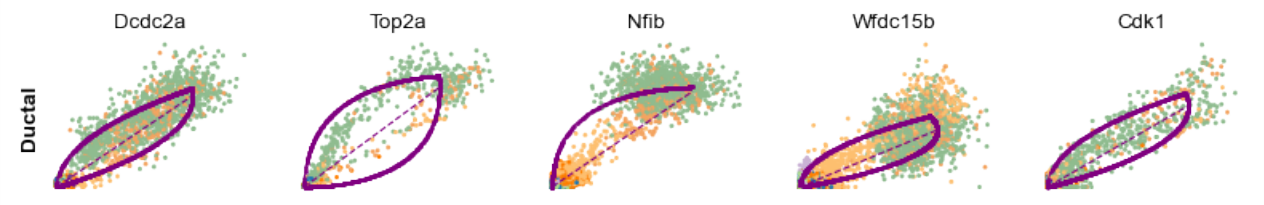
[14]:
```
scv.tl.rank_dynamical_genes(adata, groupby='clusters')
df = scv.get_df(adata, 'rank_dynamical_genes/names')
df.head(5)
```

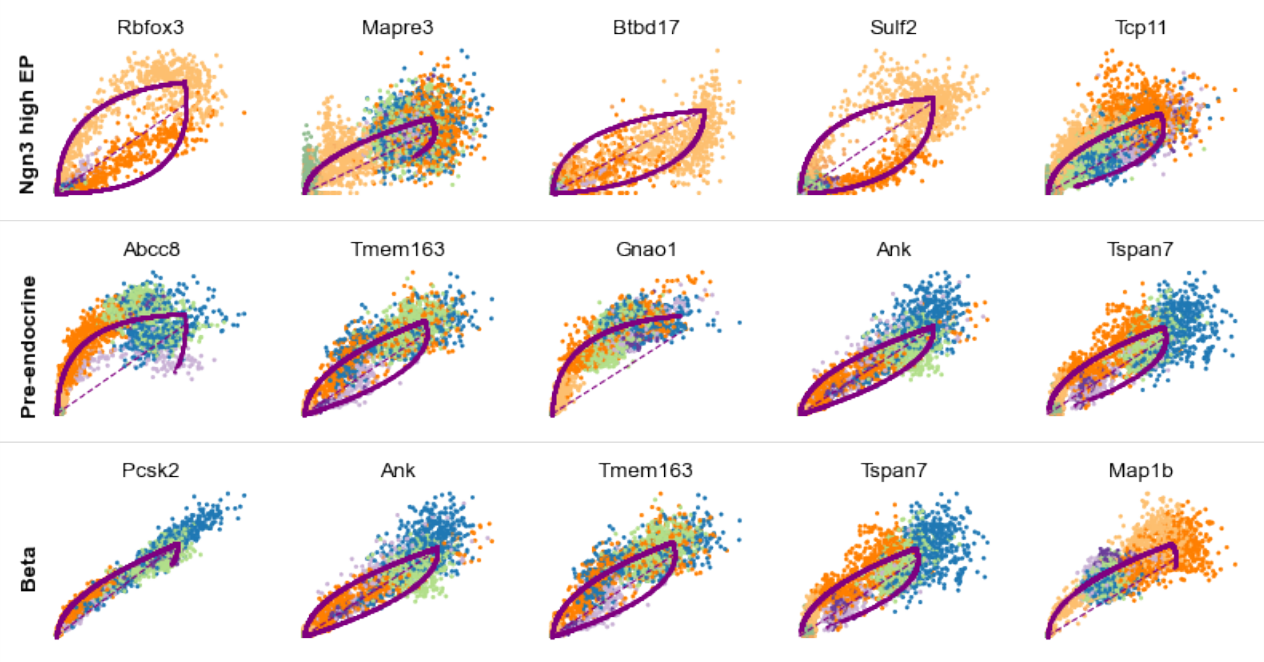
```
ranking genes by cluster-specific likelihoods
    finished (0:00:03) --> added
    'rank_dynamical_genes', sorted scores by group ids (adata.uns)
```

[14]:

	Ductal	Ngn3 low EP	Ngn3 high EP	Pre-endocrine	Beta	Alpha	Delta	Epsilon
0	Dcdc2a	Dcdc2a	Rbfox3	Abcc8	Pcsk2	Cpe	Pcsk2	Tox3
1	Top2a	Adk	Mapre3	Tmem163	Ank	Gnao1	Rap1b	Rnf130
2	Nfib	Mki67	Btbd17	Gnao1	Tmem163	Pak3	Pak3	Meis2
3	Wfdc15b	Rap1gap2	Sulf2	Ank	Tspan7	Pim2	Abcc8	Adk
4	Cdk1	Top2a	Tcp11	Tspan7	Map1b	Map1b	Klhl32	Rap1gap2

[15]:
```
for cluster in ['Ductal', 'Ngn3 high EP', 'Pre-endocrine', 'Beta']:
    scv.pl.scatter(adata, df[cluster][:5], ylabel=cluster, frameon=False)
```





4.9 Differential Kinetics

One important concern is dealing with systems that represent multiple lineages and processes, where genes are likely to show different kinetic regimes across subpopulations. Distinct cell states and lineages are typically governed by different variations in the gene regulatory networks and may hence exhibit different splicing kinetics. This gives rise to genes that display multiple trajectories in phase space.

To address this, the dynamical model can be used to perform a likelihood-ratio test for differential kinetics. This way, we can detect clusters that display kinetic behavior that cannot be well explained by a single model of the overall dynamics. Clustering cell types into their different kinetic regimes then allows fitting each regime separately.

For illustration, we apply differential kinetic analysis to dentate gyrus neurogenesis, which comprises multiple heterogeneous subpopulations.

[ ]:
```
# update to the latest version, if not done yet.
!pip install scvelo --upgrade --quiet
```

[1]:
```
import scvelo as scv
scv.logging.print_version()
```

```
Running scvelo 0.2.0 (python 3.8.2) on 2020-05-15 00:57.
```

[2]:
```
scv.settings.verbosity = 3  # show errors(0), warnings(1), info(2), hints(3)
scv.settings.presenter_view = True  # set max width size for presenter view
scv.settings.set_figure_params('scvelo')  # for beautified visualization
```

4.9.1 Prepare the Data

Processing consists of gene selection, log-normalizing, and computing moments. See the previous tutorials for further explanation.

[3]:
```
adata = scv.datasets.dentategyrus()
```

[4]:
```
scv.pp.filter_and_normalize(adata, min_shared_counts=30, n_top_genes=2000)
scv.pp.moments(adata, n_pcs=30, n_neighbors=30)
```

```
Filtered out 11019 genes that are detected in less than 30 counts (shared).
Normalized count data: X, spliced, unspliced.
Logarithmized X.
computing neighbors
    finished (0:00:02) --> added
    'distances' and 'connectivities', weighted adjacency matrices (adata.obsp)
computing moments based on connectivities
    finished (0:00:00) --> added
    'Ms' and 'Mu', moments of spliced/unspliced abundances (adata.layers)
```

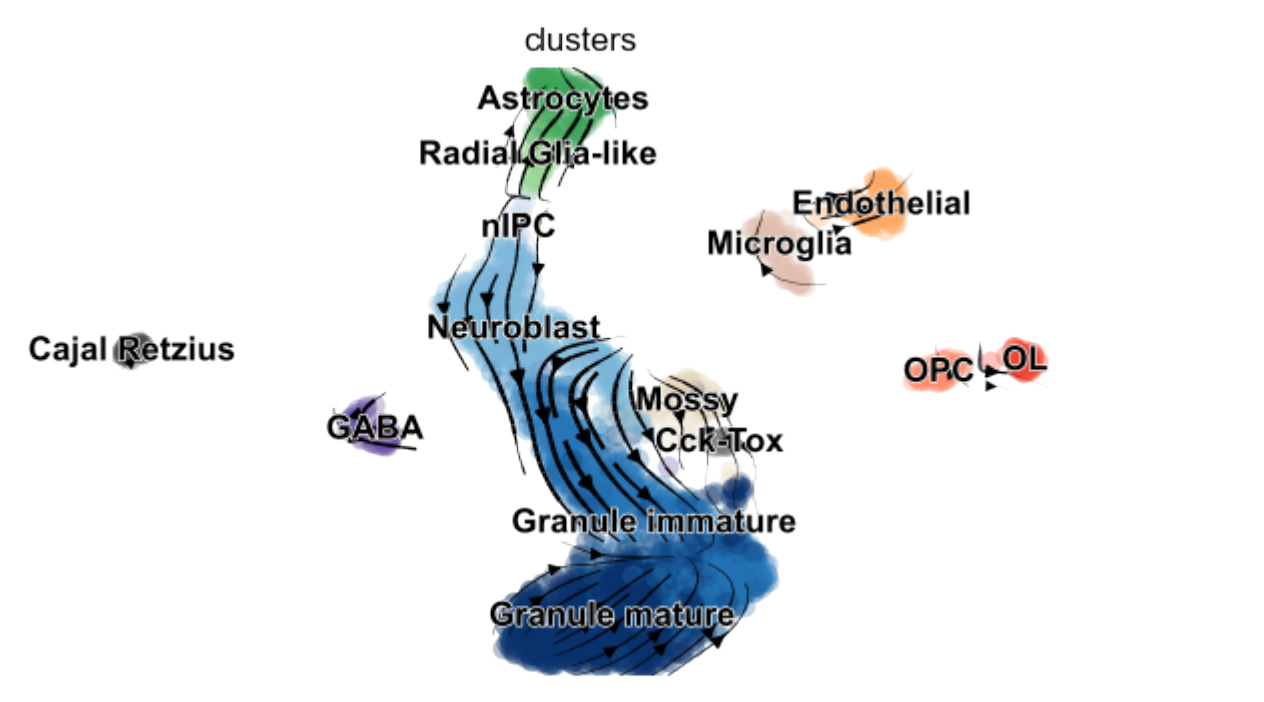
4.9.2 Basic Velocity Estimation

[5]:
```
scv.tl.velocity(adata)
scv.tl.velocity_graph(adata)
```

```
computing velocities
    finished (0:00:00) --> added
    'velocity', velocity vectors for each individual cell (adata.layers)
computing velocity graph
    finished (0:00:05) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
```

[6]:
```
scv.pl.velocity_embedding_stream(adata, basis='umap')
```

```
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
```



4.9.3 Differential Kinetic Test

Distinct cell types and lineages may exhibit different kinetics regimes as these can be governed by a different network structure. Even if cell types or lineages are related, kinetics can be differential due to alternative splicing, alternative polyadenylation and modulations in degradation.

The dynamical model allows us to address this issue with a likelihood ratio test for differential kinetics to detect clusters/lineages that display kinetic behavior that cannot be sufficiently explained by a single model for the overall dynamics. Each cell type is tested whether an independent fit yields a significantly improved likelihood.

The likelihood ratio, following an asymptotic chi-squared distribution, can be tested for significance. Note that for efficiency reasons, by default an orthogonal regression is used instead of a full phase trajectory to test whether a cluster is well explained by the overall kinetic or exhibits a different kinetic.

[7]:
```
var_names = ['Tmsb10', 'Fam155a', 'Hn1', 'Rpl6']
scv.tl.differential_kinetic_test(adata, var_names=var_names, groupby='clusters')
```

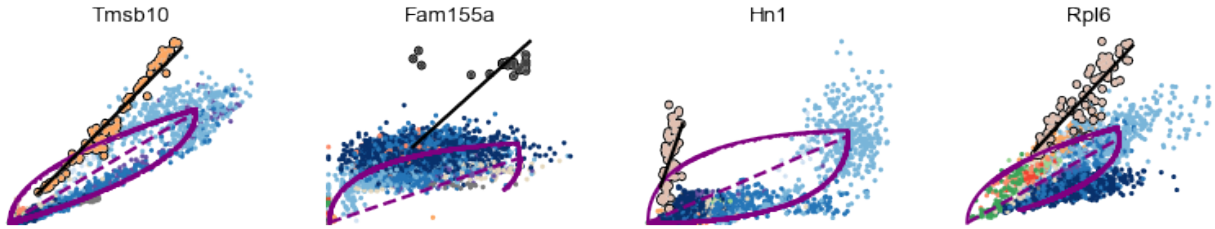
```
recovering dynamics
    finished (0:00:02) --> added
    'fit_pars', fitted parameters for splicing dynamics (adata.var)

outputs model fit of gene: Rpl6
testing for differential kinetics
    finished (0:00:00) --> added
    'fit_diff_kinetics', clusters displaying differential kinetics (adata.var)
    'fit_pval_kinetics', p-values of differential kinetics (adata.var)

outputs model fit of gene: Rpl6
```

[7]:
```
<scvelo.tools.dynamical_model.DynamicsRecovery at 0x12390f4a8>
```

[8]:
```
scv.get_df(adata[:, var_names], ['fit_diff_kinetics', 'fit_pval_kinetics'],␣
↪precision=2)
```

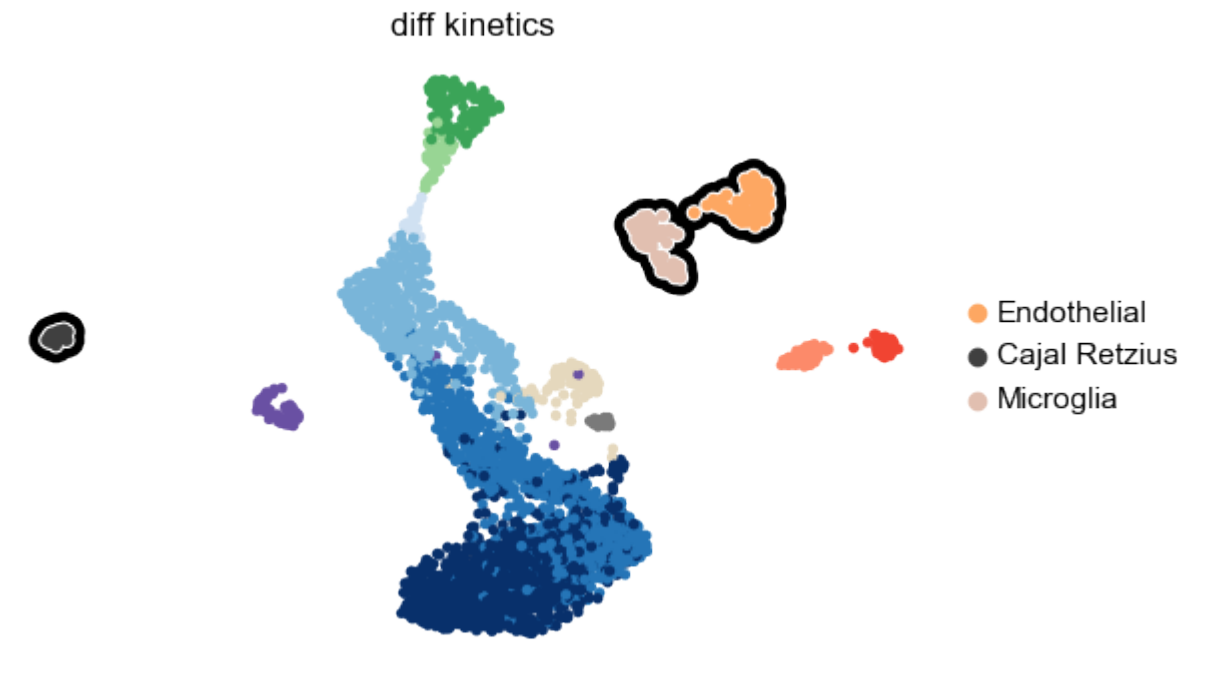
[8]:
```
        fit_diff_kinetics  fit_pval_kinetics
index
Tmsb10        Endothelial           6.02e-16
Fam155a     Cajal Retzius          8.35e-161
Hn1             Microglia           3.02e-03
Rpl6            Microglia           6.27e-15
```

[9]:
```
kwargs = dict(linewidth=2, add_linfit=True, frameon=False)
scv.pl.scatter(adata, basis=var_names, add_outline='fit_diff_kinetics', **kwargs)
```



In *Tmsb10*, for instance, Endothelial display a kinetic behaviour (outlined, with the black line fitted through), that cannot be well explained by the overall dynamics (purple curve).

[10]:
```
diff_clusters=list(adata[:, var_names].var['fit_diff_kinetics'])
scv.pl.scatter(adata, legend_loc='right', size=60, title='diff kinetics',
               add_outline=diff_clusters, outline_width=(.8, .2))
```



4.9.4 Testing top-likelihood genes

Screening through the top-likelihood genes, we find some gene-wise dynamics that display multiple kinetic regimes.

[11]:
```
scv.tl.recover_dynamics(adata)

#adata.write('data/pancreas.h5ad', compression='gzip')
#adata = scv.read('data/pancreas.h5ad')
```

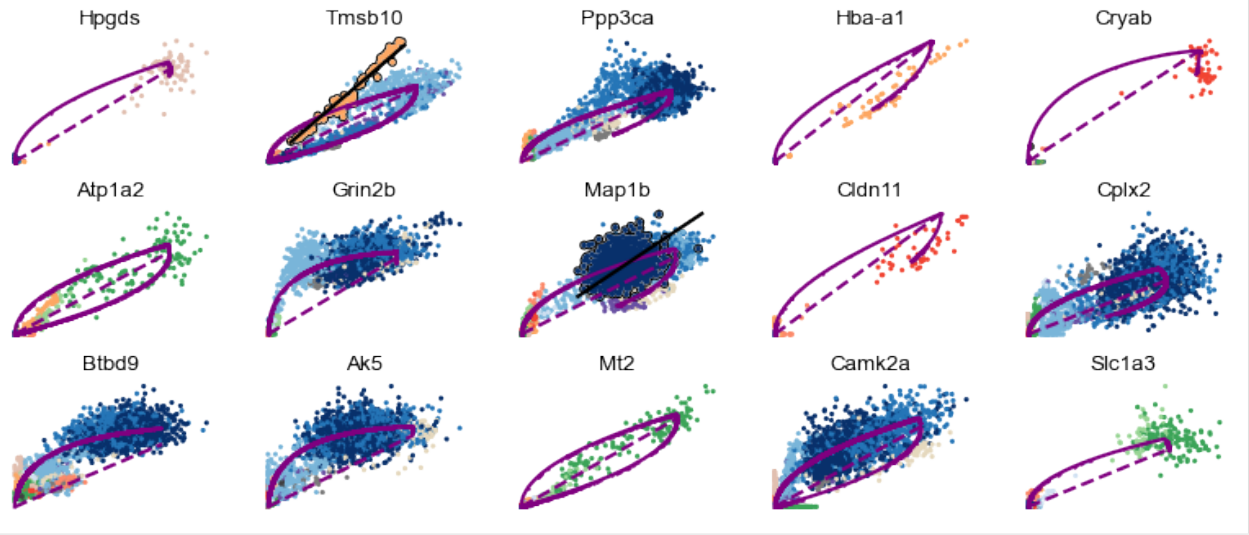
```
recovering dynamics
    finished (0:06:39) --> added
    'fit_pars', fitted parameters for splicing dynamics (adata.var)
```

[12]:
```
top_genes = adata.var['fit_likelihood'].sort_values(ascending=False).index[:100]
scv.tl.differential_kinetic_test(adata, var_names=top_genes, groupby='clusters')
```

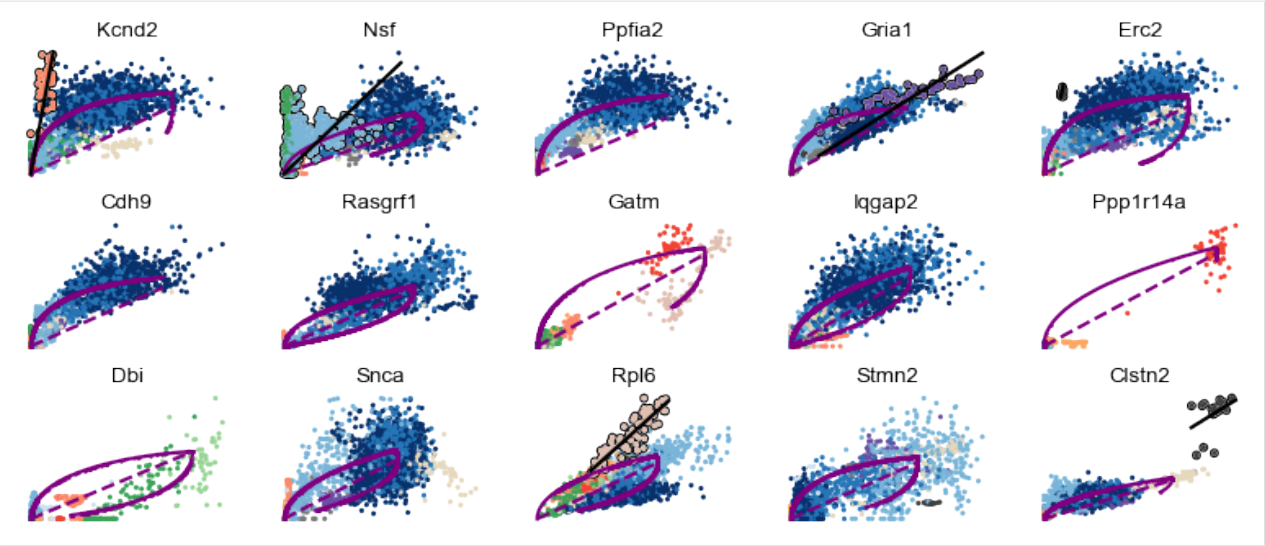
```
testing for differential kinetics
    finished (0:00:21) --> added
    'fit_diff_kinetics', clusters displaying differential kinetics (adata.var)
    'fit_pval_kinetics', p-values of differential kinetics (adata.var)
```

Particularly, cell types that are distinct from the main granule - such as Cck/Tox, GABA, Endothelial, and Microglia - occur frequently.

[13]:
```
scv.pl.scatter(adata, basis=top_genes[:15], ncols=5, add_outline='fit_diff_kinetics',␣
↪**kwargs)
```



[14]:
```
scv.pl.scatter(adata, basis=top_genes[15:30], ncols=5, add_outline='fit_diff_kinetics
↪', **kwargs)
```



4.9.5 Recompute velocities

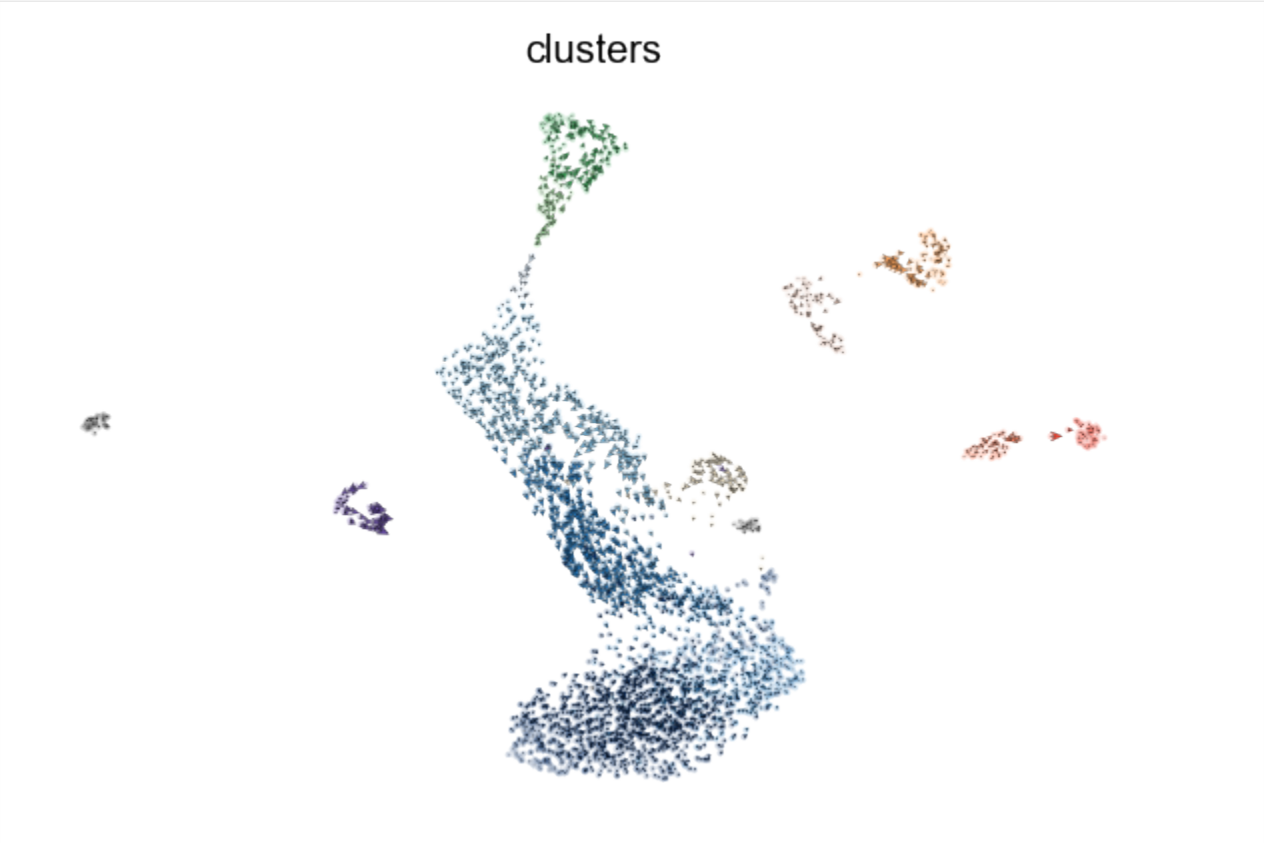
Finally, velocities can be recomputed leveraging the information of multiple competing kinetic regimes.

[15]:
```
scv.tl.velocity(adata, diff_kinetics=True)
scv.tl.velocity_graph(adata)
```

```
computing velocities
    finished (0:00:00) --> added
    'velocity', velocity vectors for each individual cell (adata.layers)
computing velocity graph
    finished (0:00:05) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
```

[16]:
```
scv.pl.velocity_embedding(adata, dpi=120, arrow_size=2, arrow_length=2)
```

```
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
```



4.10 Other Vignettes

4.10.1 Example Datasets

- Dentate Gyrus
- Pancreas

4.10.2 Nature Biotech Figures

- NBT Cover | cover | notebook
- Fig.1 Concept | figure | notebook
- Fig.2 Dentate Gyrus | figure | notebook
- Fig.3 Pancreas | figure | notebook
- Suppl. Figures | figure | notebook | runtime

All notebooks are deposited at GitHub. Found a bug? Feel free to submit an issue.

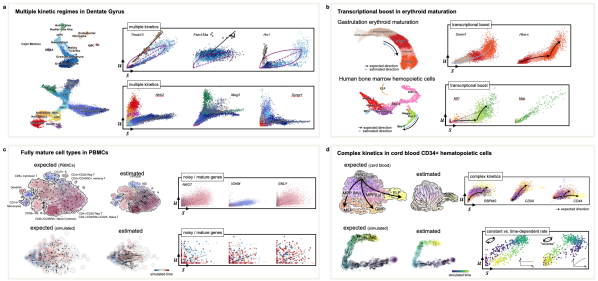
4.11 Challenges and Perspectives

This page complements our manuscript Bergen et al. (MSB, 2021) RNA velocity - Current challenges and future perspectives

We provide several examples to discuss potential pitfalls of current RNA velocity modeling approaches, and provide guidance on how the ensuing challenges may be addressed. Our aspiration is to suggest promising future directions and to stimulate a communities effort on further model extensions.

In the following, you find two vignettes with several use cases, as well as an in-depth analysis of time-dependent kinetic rate parameters.

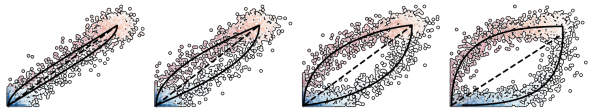
4.11.1 Potential pitfalls



This notebook reproduces Fig. 2 with several use cases, including multiple kinetics in Dentate Gyrus, transcriptional boost in erythroid lineage, and misleading arrow projections in mature PBMCs.

Notebook: Perspectives

4.11.2 Kinetic parameter analysis



This notebook reproduces Fig. 3, where we demonstrate how time-variable kinetic rates shape the curvature patterns of gene activation.

Notebook: Kinetic parameter analysis

BIBLIOGRAPHY

[Bergen20] Bergen *et al.* (2020), *Generalizing RNA velocity to transient cell states through dynamical modeling*, Nature Biotech.

[Manno18] La Manno *et al.* (2018), *RNA velocity of single cells*, Nature.

[Wolf18] Wolf *et al.* (2018), *Scanpy: large-scale single-cell gene expression data analysis*, Genome Biology.

[Wolf19] Wolf *et al.* (2019), *PAGA: graph abstraction reconciles clustering with trajectory inference*, Genome Biology.

PYTHON MODULE INDEX

S

INDEX

T

U

V