scFates

Louis Faure

Feb 16, 2024

MAIN

scFates is a scalable Python suite for tree inference and advanced pseudotime analysis.

The related work is now available in Bioinformatics:

```
Louis Faure, Ruslan Soldatov, Peter V. Kharchenko, Igor Adameyko
scFates: a scalable python package for advanced pseudotime and bifurcation analysis from␣
↪single cell data
Bioinformatics, btac746; doi: https://doi.org/10.1093/bioinformatics/btac746
```

It initially is a translation from crestree, a R package developed for the analysis of neural crest fates during murine embryonic development (Soldatov et al., Science, 2019), and used in another study of neural crest derived sensory neurons (Faure et al., Nature Communications, 2020).

The initial R version included a tree inference approach inspired from SimplePPT, this version now adds the choice of using ElPiGraph, another method for principal graph learning (Albergante et al., Entropy, 2020). scFates is fully compatible with scanpy, and contains GPU-accelerated functions for faster and scalable inference.

CHAPTER

ONE

ANALYSIS KEY STEPS

- learn a principal tree on the space of your choice (gene expression, PCA, diffusion, . . .).
- obtain a pseudotime value upon selection of a root (or two roots).
- test and fit features significantly changing along the tree, cluster them.
- perform differential expression between branches.
- identify branch-specific early activating features and probe their correlations prior to bifurcation.

1.1 Installation

scFates is continuously tested with python 3.11, it is recommended to use a Miniconda environment.

1.1.1 PyPI

scFates is available on pypi, you can install it using:

```
pip install scFates
```

or the latest development version can be installed from GitHub using:

```
pip install git+https://github.com/LouisFaure/scFates
```

1.1.2 With all dependencies

- *scFates.pp.find_overdispersed()*, *scFates.tl.test_association()*, *scFates.tl.fit()*, *scFates.tl.test_fork()*, *scFates.tl.activation()*: Require R package mgcv interfaced via python package rpy2:

  ```
  conda create -n scFates -c conda-forge -c r python=3.11 r-mgcv rpy2 -y
  conda activate scFates
  pip install scFates
  ```

to avoid any possible crashes due to rpy2 not finding the R install on conda, run the following import command:

```
import os, sys
os.environ['R_HOME'] = sys.exec_prefix+"/lib/R/"
import scFates
```

- *scFates.tl.cellrank_to_tree()*: Requires cellrank to be installed in order to function:

```
pip install cellrank
```

1.1.3 On Apple Silicon

Installing mgcv using conda/mamba on Apple Silicon lead to the package not being able to find some dynamic libraries (BLAS). In that case it is recommended to install it separately:

```
mamba create -n scFates -c conda-forge -c bioconda -c defaults python numpy=1.24.4
↪"libblas=*=*accelerate" rpy2 -y
mamba activate scFates
Rscript -e 'install.packages("mgcv",repos = "http://cran.us.r-project.org")'
```

1.1.4 GPU dependencies (optional)

If you have a nvidia GPU, scFates can leverage CUDA computations for speedups for the following functions:

scFates.pp.filter_cells(), *scFates.pp.batch_correct()*, *scFates.pp.diffusion()*, *scFates.tl.tree()*, *scFates.tl.cluster()*

The latest version of rapids framework is required (at least 0.17) it is recommanded to create a new conda environment:

```
conda create -n scFates-gpu -c rapidsai -c nvidia -c conda-forge -c defaults cuml=21.12␣
↪cugraph=21.12 python=3.8 cudatoolkit=11.0 -y
conda activate scFates-gpu
pip install git+https://github.com/j-bac/elpigraph-python.git
pip install scFates
```

1.2 API

Import scFates as:

```
import scFates as scf
```

Some convenient preprocessing functions translated from pagoda2 have been included:

1.2.1 Pre-processing

pp.filter_cells(adata[, device, p_level, ...])	Filter cells using on gene/molecule relationship.
pp.batch_correct(adata, batch_key[, layer, ...])	batch correction of the count matrix.
pp.find_overdispersed(adata[, gam_k, alpha, ...])	Find overdispersed gene, using pagoda2 strategy.
pp.diffusion(adata[, n_components, knn, ...])	Wrapper to generate diffusion maps using Palantir.

scFates.pp.filter_cells

scFates.pp.**filter_cells**(*adata*, *device='cpu'*, *p_level=None*, *subset=True*, *plot=False*, *copy=False*)
Filter cells using on gene/molecule relationship.

Code has been translated from pagoda2 R function gene.vs.molecule.cell.filter.

Parameters

adata : AnnData Annotated data matrix.

device Run gene and molecule counting on either *cpu* or on *gpu*.

p_level Statistical confidence level for deviation from the main trend, used for cell filtering (default=min(1e-3,1/adata.shape[0]))

subset if False, add a column *outlier* in adata.obs, otherwise subset the adata.

plot Plot the molecule distribution and the gene/molecule dependency fit.

copy Return a copy instead of writing to adata.

Returns

adata – if *copy=True* and *subset=True* it returns subsetted (removing outliers) or else add fields to *adata*:

.obs['outlier'] whether a cell is an outlier.

Return type anndata.AnnData

scFates.pp.batch_correct

scFates.pp.**batch_correct**(*adata*, *batch_key*, *layer='X'*, *depth_scale=1000.0*, *device='cpu'*, *inplace=True*)
batch correction of the count matrix.

Code has been translated from pagoda2 R function setCountMatrix (plain model).

Parameters

adata Annotated data matrix.

batch_key Column name to use for batch.

layer Which layer to correct, if layer doesn't exist, then correct X and save to layer

depth_scale Depth scale.

device Run method on either *cpu* or on *gpu*.

copy Return a copy instead of writing to adata.

Returns

adata – if *inplace=False* it returns the corrected matrix, else it update field to *adata*:

.X batch-corrected count matrix.

Return type anndata.AnnData

scFates.pp.find_overdispersed

scFates.pp.**find_overdispersed**(*adata*, *gam_k=5*, *alpha=0.05*, *layer='X'*, *plot=False*, *copy=False*)
Find overdispersed gene, using pagoda2 strategy.

Code has been translated from pagoda2 R function adjustVariance.

Parameters

adata Annotated data matrix.

gam_k : int (default: 5) The k used for the generalized additive model.

alpha : float (default: 0.05) The criterion used to measure statistical significance.

layer : str (default: 'X') Which layer to use.

plot : bool (default: False) Plot selected genes.

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

Returns

adata –

if *copy=True* it returns or else add fields to *adata*:

.var['res'] residuals of GAM fit.

.var['lp'] p-value.

.var['lpa'] BH adjusted p-value.

.var['qv'] percentile of qui-squared distribution.

.var['highly_variable'] feature is over-dispersed.

Return type anndata.AnnData

scFates.pp.diffusion

scFates.pp.**diffusion**(*adata*, *n_components=10*, *knn=30*, *alpha=0*, *multiscale=True*, *n_eigs=None*, *device='cpu'*, *n_pcs=50*, *save_uns=False*, *copy=False*)
Wrapper to generate diffusion maps using Palantir.

Parameters

adata : AnnData Annotated data matrix.

n_components : int (default: 10) Number of diffusion components.

knn : int (default: 30) Number of nearest neighbors for graph construction.

alpha : float (default: 0) Normalization parameter for the diffusion operator.

multiscale : bool (default: True) Whether to get mutliscale diffusion space (calls palantir.utils.determine_multiscale_space).

n_eigs : int | NoneOptional[int] (default: None) if multiscale is True, how much components to retain.

device : {'cpu', 'gpu'}Literal['cpu', 'gpu'] (default: 'cpu') Run method on either *cpu* or on *gpu*.

do_PCA Whether to perform PCA or not.

n_pcs : int (default: 50) Number of PC components.

seed Get reproducible results for the GPU implementation.

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

Returns

adata – if *copy=True* it returns AnnData, else it update field to *adata*:

.obsm['X_diffusion'] if *multiscale = False*, diffusion space.

.obsm['X_multiscale_diffusion'] if *multiscale = True*, multiscale diffusion space.

.uns['diffusion'] dict containing results from Palantir.

Return type anndata.AnnData

1.2.2 Tree inference

tl.tree(adata[, Nodes, use_rep, ndims_rep, ...])	Generate a principal tree.
tl.curve(adata[, Nodes, use_rep, ndims_rep, ...])	Generate a principal curve.
tl.circle(adata[, Nodes, use_rep, ...])	Generate a principal circle.
tl.cellrank_to_tree(adata, time, Nodes[, ...])	Converts CellRank [Lange21] fate probabilities into a principal tree that can be analysed by scFates.
tl.explore_sigma(adata, Nodes[, use_rep, ...])	Explore varisou sigma parameters for best tree fitting.

scFates.tl.tree

scFates.tl.**tree**(*adata*, *Nodes=None*, *use_rep=None*, *ndims_rep=None*, *weight_rep=None*, *method='ppt'*, *init=None*, *ppt_sigma=0.1*, *ppt_lambda=1*, *ppt_metric='euclidean'*, *ppt_nsteps=50*, *ppt_err_cut=0.005*, *ppt_gpu_tpb=16*, *epg_lambda=0.01*, *epg_mu=0.1*, *epg_trimmingradius=inf*, *epg_initnodes=2*, *epg_extend_leaves=False*, *epg_verbose=False*, *device='cpu'*, *plot=False*, *basis='umap'*, *seed=None*, *copy=False*, ***kwargs*)

Generate a principal tree.

Learn a simplified representation on any space, compsed of nodes, approximating the position of the cells on a given space such as gene expression, pca, diffusion maps, ... If *method=='ppt'*, uses simpleppt implementation from [Soldatov19]. If *method=='epg'*, uses Elastic Principal Graph approach from [Albergante20].

Parameters

adata : AnnData Annotated data matrix.

Nodes : int | NoneOptional[int] (default: None) Number of nodes composing the principial tree, use a range of 10 to 100 for ElPiGraph approach and 100 to 2000 for PPT approach.

use_rep : str | NoneOptional[str] (default: None) Choose the space to be learned by the principal tree.

ndims_rep : int | NoneOptional[int] (default: None) Number of dimensions to use for the inference.

weight_rep : str | NoneOptional[str] (default: None) If *ppt*, use a weight matrix for learning the tree.

method : {'ppt', 'epg'}Literal['ppt', 'epg'] (default: 'ppt') If *ppt*, uses simpleppt approach, *ppt_lambda* and *ppt_sigma* are the parameters controlling the algorithm. If *epg*, uses ComputeElasticPrincipalTree function from elpigraph python package, *epg_lambda epg_mu* and *epg_trimmingradius* are the parameters controlling the algorithm.

init : DataFrame | NoneOptional[DataFrame] (default: None) Initialise the point positions.

ppt_sigma : float | int | NoneUnion[float, int, None] (default: 0.1) Regularization parameter for simpleppt [Mao15].

ppt_lambda : float | int | NoneUnion[float, int, None] (default: 1) Parameter for simpleppt, penalty for the tree length [Mao15].

ppt_metric : str (default: 'euclidean') The metric to use to compute distances in high dimensional space. For compatible metrics, check the documentation of sklearn.metrics.pairwise_distances if using cpu or cuml.metrics.pairwise_distances if using gpu.

ppt_nsteps : int (default: 50) Number of steps for the optimisation process of simpleppt.

ppt_err_cut : float (default: 0.005) Stop simpleppt algorithm if proximity of principal points between iterations less than defiend value.

ppt_gpu_tpb : int (default: 16) Threads per block parameter for cuda computations.

epg_lambda : float | int | NoneUnion[float, int, None] (default: 0.01) Parameter for ElPiGraph, coefficient of 'stretching' elasticity [Albergante20].

epg_mu : float | int | NoneUnion[float, int, None] (default: 0.1) Parameter for ElPiGraph, coefficient of 'bending' elasticity [Albergante20].

epg_trimmingradius : Optional (default: inf) Parameter for ElPiGraph, trimming radius for MSE-based data approximation term [Albergante20].

epg_initnodes : int | NoneOptional[int] (default: 2) numerical 2D matrix, the k-by-m matrix with k m-dimensional positions of the nodes in the initial step

epg_extend_leaves : bool (default: False) Parameter for ElPiGraph, calls `elpigraph.ExtendLeaves()` after graph learning.

epg_verbose : bool (default: False) show verbose output of epg algorithm

device : {'cpu', 'gpu'}Literal['cpu', 'gpu'] (default: 'cpu') Run either mehtod on *cpu* or on *gpu*

plot : bool (default: False) Plot the resulting tree.

basis : str | NoneOptional[str] (default: 'umap') Basis onto which the resulting tree should be projected.

seed : int | NoneOptional[int] (default: None) A numpy random seed.

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

****kwargs** Arguments passsed to `elpigraph.computeElasticPrincipalTree()`

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.uns['ppt'] dictionnary containing information from simpelppt tree if method='ppt'

.uns['epg'] dictionnary containing information from elastic principal tree if method='epg'

.obsm['R'] soft assignment of cells to principal points

.uns['graph']['B'] adjacency matrix of the principal points

.uns['graph']['F'] coordinates of principal points in representation space

Return type anndata.AnnData

scFates.tl.curve

scFates.tl.**curve**(*adata*, *Nodes=None*, *use_rep=None*, *ndims_rep=None*, *init=None*, *epg_lambda=0.01*, *epg_mu=0.1*, *epg_trimmingradius=inf*, *epg_extend_leaves=False*, *epg_verbose=False*, *device='cpu'*, *plot=False*, *basis='umap'*, *seed=None*, *copy=False*, ***kwargs*)

Generate a principal curve.

Learn a curved representation on any space, composed of nodes, approximating the position of the cells on a given space such as gene expression, pca, diffusion maps, … Uses ElpiGraph algorithm.

Parameters

adata : AnnData Annotated data matrix.

Nodes : int | NoneOptional[int] (default: None) Number of nodes composing the principial tree, use a range of 10 to 100 for ElPiGraph approach and 100 to 2000 for PPT approach.

use_rep : str | NoneOptional[str] (default: None) Choose the space to be learned by the principal tree.

ndims_rep : int | NoneOptional[int] (default: None) Number of dimensions to use for the inference.

epg_lambda : float | int | NoneUnion[float, int, None] (default: 0.01) Parameter for ElPiGraph, coefficient of 'stretching' elasticity [Albergante20].

epg_mu : float | int | NoneUnion[float, int, None] (default: 0.1) Parameter for ElPiGraph, coefficient of 'bending' elasticity [Albergante20].

epg_trimmingradius : Optional (default: inf) Parameter for ElPiGraph, trimming radius for MSE-based data approximation term [Albergante20].

epg_extend_leaves : bool (default: False) Parameter for ElPiGraph, calls `elpigraph.ExtendLeaves()` after graph learning.

epg_verbose : bool (default: False) show verbose output of epg algorithm

device : {'cpu', 'gpu'}Literal['cpu', 'gpu'] (default: 'cpu') Run method on either *cpu* or on *gpu*

plot : bool (default: False) Plot the resulting tree.

basis : str | NoneOptional[str] (default: 'umap') Basis onto which the resulting tree should be projected.

seed : int | NoneOptional[int] (default: None) A numpy random seed.

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

****kwargs** Arguments passsed to `elpigraph.computeElasticPrincipalCurve()`

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.uns['epg'] dictionnary containing information from elastic principal curve

.obsm['X_R'] soft assignment of cells to principal points

.uns['graph']['B'] adjacency matrix of the principal points

.uns['graph']['F'] coordinates of principal points in representation space

Return type anndata.AnnData

scFates.tl.circle

scFates.tl.**circle**(*adata*, *Nodes=None*, *use_rep=None*, *ndims_rep=None*, *epg_lambda=0.01*, *epg_mu=0.1*, *epg_trimmingradius=inf*, *epg_verbose=False*, *device='cpu'*, *plot=False*, *basis='umap'*, *seed=None*, *copy=False*, ***kwargs*)

Generate a principal circle.

Learn a circled representation on any space, composed of nodes, approximating the position of the cells on a given space such as gene expression, pca, diffusion maps, … Uses ElpiGraph algorithm.

Parameters

adata : AnnData Annotated data matrix.

Nodes : int | NoneOptional[int] (default: None) Number of nodes composing the principial tree, use a range of 10 to 100 for ElPiGraph approach and 100 to 2000 for PPT approach.

use_rep : str | NoneOptional[str] (default: None) Choose the space to be learned by the principal tree.

ndims_rep : int | NoneOptional[int] (default: None) Number of dimensions to use for the inference.

epg_lambda : float | int | NoneUnion[float, int, None] (default: 0.01) Parameter for ElPiGraph, coefficient of 'stretching' elasticity [Albergante20].

epg_mu : float | int | NoneUnion[float, int, None] (default: 0.1) Parameter for ElPiGraph, coefficient of 'bending' elasticity [Albergante20].

epg_trimmingradius : Optional (default: inf) Parameter for ElPiGraph, trimming radius for MSE-based data approximation term [Albergante20].

epg_verbose : bool (default: False) show verbose output of epg algorithm

device : {'cpu', 'gpu'}Literal['cpu', 'gpu'] (default: 'cpu') Run method on either *cpu* or on *gpu*

plot : bool (default: False) Plot the resulting tree.

basis : str | NoneOptional[str] (default: 'umap') Basis onto which the resulting tree should be projected.

seed : int | NoneOptional[int] (default: None) A numpy random seed.

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

****kwargs** Arguments passsed to `elpigraph.computeElasticPrincipalCircle()`

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.uns['epg'] dictionnary containing information from elastic principal curve

.obsm['X_R'] soft assignment of cells to principal points

.uns['graph']['B'] adjacency matrix of the principal points

.uns['graph']['F'] coordinates of principal points in representation space

Return type anndata.AnnData

scFates.tl.cellrank_to_tree

scFates.tl.**cellrank_to_tree**(*adata*, *time*, *Nodes*, *method='ppt'*, *ppt_lambda=100*, *auto_root=False*, *root_params={}*, *reassign_pseudotime=False*, *copy=False*, ***kwargs*)

Converts CellRank [Lange21] fate probabilities into a principal tree that can be analysed by scFates.

It combines the projection generated by *cr.pl.circular_projection* with any measure of differentiation (CytoTRACE, latent time). A tree is fitted onto this new embedding.

Parameters

adata Annotated data matrix.

time time key to use for the additional dimension used in combination with *cr.pl.circular_projection*.

Nodes : int Number of nodes that compose the principal graph.

method : {'ppt', 'epg'}Literal['ppt', 'epg'] (default: 'ppt') If ppt, uses simpleppt approach, *ppt_lambda* and *ppt_sigma* are the parameters controlling the algorithm. If *epg*, uses ComputeElasticPrincipalTree function from elpigraph python package, *epg_lambda epg_mu* and *epg_trimmingradius* are the parameters controlling the algorithm.

ppt_lambda : int (default: 100) Parameter for simpleppt, penalty for the tree length [Mao15]. Usually works well at default for the conversion.

auto_root : bool (default: False) Automatically select the root tip using the time key.

min_val min_val parameter from *scFates.tl.root()*

reassign_pseudotime : bool (default: False) whether use the time key to replace the distances comptued from the tree.

copy Return a copy instead of writing to adata.

kwargs arguments to pass to function *scFates.tl.tree()*.

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.obsm['X_fates'] representation generated by combining the time key with projection generated by `cellrank.pl.circular_projection()`.

.uns['ppt'] dictionnary containing information from simpelppt tree if method='ppt'

.uns['epg'] dictionnary containing information from elastic principal tree if method='epg'

.uns['graph']['B'] adjacency matrix of the principal points

.uns['graph']['R'] soft assignment of cells to principal point in representation space

.uns['graph']['F'] coordinates of principal points in representation spac

Return type anndata.AnnData

scFates.tl.explore_sigma

scFates.tl.**explore_sigma**(*adata*, *Nodes*, *use_rep=None*, *ndims_rep=None*, *sigmas=[1000, 100, 10, 1, 0.1, 0.01]*, *nsteps=1*, *metric='euclidean'*, *seed=None*, *plot=False*, *second_round=False*, ***kwargs*)

Explore varisou sigma parameters for best tree fitting. Given that high sigma tend to collapse the principal points into the middle of the whole data (meaning taking in account all the datapoints regardless their locality), it is possible to explore which sigma is best by detecting at which level the tree stops collapsing.

Parameters

adata Annotated data matrix.

Nodes Number of nodes composing the principial tree, use a range of 10 to 100 for ElPiGraph approach and 100 to 2000 for PPT approach.

use_rep Choose the space to be learned by the principal tree.

ndims_rep Number of dimensions to use for the inference.

sigmas Range of sigma parameters to test.

device Run method on either *cpu* or on *gpu*.

nsteps Number of SimplePPT iteration, usually 1 is enough.

metric Distance metric to use.

seed A numpy random seed.

plot Plot the resulting tree.

second_round Perform a second exploration, on a restricted sigma parameters based on the first estimated sigma.

****kwargs** Arguments passsed to `elpigraph.computeElasticPrincipalCircle()`

Returns **sigma** – suggested sigma value

Return type float

1.2.3 Tree operations

tl.cleanup(adata[, minbranchlength, leaves, ...])	Remove spurious branches from the tree.
tl.subset_tree(adata[, root_milestone, ...])	Subset the fitted tree.
tl.attach_tree(adata, adata_branch[, linkage])	Attach a tree to another!
tl.simplify(adata[, n_nodes, copy])	While keeping nodes defining forks and tips (milestones), reduce the number of nodes composing the segments.
tl.convert_to_soft(adata, sigma, lam[, ...])	Convert an hard assignment matrix to a soft one, allowing for probabilistic mapping.

scFates.tl.cleanup

scFates.tl.**cleanup**(*adata*, *minbranchlength=3*, *leaves=None*, *copy=False*)

Remove spurious branches from the tree.

Parameters

adata : AnnData Annotated data matrix.

minbranchlength : int (default: 3) Branches having less than the defined amount of nodes are discarded

leaves : int | NoneOptional[int] (default: None) Manually select branch tips to remove

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

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.uns['graph']['B'] subsetted adjacency matrix of the principal points.

.uns['graph']['R'] subsetted updated soft assignment of cells to principal point in representation space.

.uns['graph']['F'] subsetted coordinates of principal points in representation space.

Return type anndata.AnnData

scFates.tl.subset_tree

scFates.tl.**subset_tree**(*adata*, *root_milestone=None*, *milestones=None*, *mode='extract'*, *t_min=None*, *t_max=None*, *copy=False*)

Subset the fitted tree.

if pseudotime parameter used, cutoff tree by removing cells/nodes after or before defined pseudotime.

Parameters

adata : AnnData Annotated data matrix.

root_milestone : str | NoneOptional[str] (default: None) tip defining progenitor branch.

milestones : Iterable | NoneOptional[Iterable] (default: None) tips defining the progenies branches.

mode : {'extract', 'substract', 'pseudotime'}Literal['extract', 'substract', 'pseudotime'] (default: 'extract') whether to substract or extract the mentionned path.

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

Returns

adata – subsetted dataset if *copy=True* it returns or else subsets these fields to *adata*:

.uns['graph']['B'] subsetted adjacency matrix of the principal points.

.uns['graph']['R'] subsetted updated soft assignment of cells to principal point in representation space.

.uns['graph']['F'] subsetted coordinates of principal points in representation space.

.obs['old_milestones'] previous milestones from initial tree.

Return type anndata.AnnData

scFates.tl.attach_tree

scFates.tl.**attach_tree**(*adata*, *adata_branch*, *linkage=None*)

Attach a tree to another!

Given that the datasets were initially processed together.

Parameters

adata : AnnData Annotated data matrix.

adata_branch : AnnData Annotated data matrix containing cells and tree to attach to the adata.

linkage : None | tupleOptional[tuple] (default: None) Force the attachment of the two tree between two respective milestones (main tree, branch), the adjacency matrix will not be updated.

Returns

adata – combined dataset with the following merged tree fields:

.uns['graph']['B'] merged adjacency matrix of the principal points.

.uns['graph']['R'] merged updated soft assignment of cells to principal point in representation space.

.uns['graph']['F'] merged coordinates of principal points in representation space.

Return type anndata.AnnData

scFates.tl.simplify

scFates.tl.**simplify**(*adata*, *n_nodes=10*, *copy=False*)

While keeping nodes defining forks and tips (milestones), reduce the number of nodes composing the segments.

This can be helpful to simplify visualisations.

Parameters

adata : AnnData Annotated data matrix.

n_nodes : int (default: 10) Number of nodes to keep per segments in between milestone nodes.

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

Returns **adata** – Dataset with simplified graph.

Return type anndata.AnnData

scFates.tl.convert_to_soft

scFates.tl.**convert_to_soft**(*adata*, *sigma*, *lam*, *n_steps=1*, *copy=False*)

Convert an hard assignment matrix to a soft one, allowing for probabilistic mapping.

Parameters

adata Annotated data matrix.

sigma : float Sigma parameter from SimplePPT.

lam : int Lambda parameter from SimplePPT.

n_steps : int (default: 1) Number of steps to run the solving of R and F

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

Returns

adata – dataset if *copy=True* it returns or else updated these fields to *adata*:

.uns['graph']['R'] converted from hard to soft assignment matrix.

.uns['graph']['F'] solved from newly converted soft assignment matrix.

Return type anndata.AnnData

1.2.4 Pseudotime analysis

tl.root(adata, root[, tips_only, min_val, ...])	Define the root of the trajectory.
tl.roots(adata, roots, meeting[, copy])	Define 2 roots of the tree.
tl.pseudotime(adata[, n_jobs, n_map, seed, copy])	Compute pseudotime.
tl.dendrogram(adata[, crowdedness, n_jobs])	Generate a single-cell dendrogram embedding.
tl.test_association(adata[, n_map, n_jobs, ...])	Determine a set of genes significantly associated with the trajectory.
tl.test_association_covariate(adata, group_key)	Separately test for associated features for each covariates on the same trajectory path.
tl.test_association_monocle3(adata[, ...])	Determine a set of genes significantly associated with the trajectory.
tl.fit(adata[, features, layer, n_map, ...])	Model feature expression levels as a function of tree positions.
tl.cluster(adata[, layer, n_neighbors, ...])	Cluster features.
tl.test_covariate(adata, group_key[, ...])	Test for branch differential gene expression between covariates on the same trajectory path.
tl.linearity_deviation(adata, ...[, ...])	Identifies genes that specifically characterize a given transition but not the progenitors neither the progenies.
tl.unroll_circle(adata[, copy])	Unroll circle to get full spectrum of pseudotime values along it.

scFates.tl.root

scFates.tl.**root**(*adata*, *root*, *tips_only=False*, *min_val=False*, *layer=None*, *copy=False*)
Define the root of the trajectory.

Parameters

adata Annotated data matrix.

root Either an Id (int) of the tip of the fork to be considered as a root. Or a key (str) from obs/X (such as CytoTRACE) for automatic selection.

tips_only Perform automatic assignment on tips only.

min_val Perform automatic assignment using minimum value instead.

layer If key is in X, choose which layer to use for the averaging.

copy Return a copy instead of writing to adata.

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.uns['graph']['root'] selected root.

.uns['graph']['pp_info'] for each PP, its distance vs root and segment assignment.

.uns['graph']['pp_seg'] segments network information.

Return type anndata.AnnData

scFates.tl.roots

scFates.tl.**roots**(*adata*, *roots*, *meeting*, *copy=False*)
Define 2 roots of the tree.

Parameters

adata : AnnData Annotated data matrix.

roots list of tips or forks to be considered a roots.

meeting node ID of the meeting point fo the two converging paths.

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

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.uns['graph']['root'] farthest root selected.

.uns['graph']['root2'] 2nd root selected.

.uns['graph']['meeting'] meeting point on the tree.

.uns['graph']['pp_info'] for each PP, its distance vs root and segment assignment).

.uns['graph']['pp_seg'] segments network information.

Return type anndata.AnnData

scFates.tl.pseudotime

scFates.tl.**pseudotime**(*adata*, *n_jobs=1*, *n_map=1*, *seed=None*, *copy=False*)

Compute pseudotime.

Projects cells onto the tree, and uses distance from the root as a pseudotime value.

Parameters

adata : AnnData Annotated data matrix.

n_jobs : int (default: 1) Number of cpu processes to use in case of performing multiple mapping.

n_map : int (default: 1) number of probabilistic mapping of cells onto the tree to use. If n_map=1 then likelihood cell mapping is used.

seed : int | NoneOptional[int] (default: None) A numpy random seed for reproducibility for muliple mappings

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

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.obs['edge'] assigned edge.

.obs['t'] assigned pseudotime value.

.obs['seg'] assigned segment of the tree.

.obs['milestone'] assigned region surrounding forks and tips.

.uns['pseudotime_list'] list of cell projection from all mappings.

Return type anndata.AnnData

scFates.tl.dendrogram

scFates.tl.**dendrogram**(*adata*, *crowdedness=1*, *n_jobs=1*)

Generate a single-cell dendrogram embedding.

This representation aims in simplifying and abstracting the view of the tree, it follows URD style represenation of the cells.

Parameters

adata : AnnData Annotated data matrix.

crowdedness : float (default: 1) will influence the repartition of the cells along the segments.

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.obs['X_dendro'] new embedding generated.

.uns['dendro_segments'] tree segments used for plotting.

Return type anndata.AnnData

scFates.tl.test_association

scFates.tl.**test_association**(*adata*, *n_map=1*, *n_jobs=1*, *spline_df=5*, *fdr_cut=0.0001*, *A_cut=1*, *st_cut=0.8*, *reapply_filters=False*, *plot=False*, *copy=False*, *layer=None*)

Determine a set of genes significantly associated with the trajectory.

Feature expression is modeled as a function of pseudotime in a branch-specific manner, using cubic spline regression $g_i \sim t_i$ for each branch independently. This tree-dependent model is then compared with an unconstrained model $g_i \sim 1$ using F-test.

The models are fit using *mgcv* R package.

Benjamini-Hochberg correction is used to adjust for multiple hypothesis testing.

Parameters

adata : AnnData Annotated data matrix.

layer : str | NoneOptional[str] (default: None) adata layer to use for the test.

n_map : int (default: 1) number of cell mappings from which to do the test.

n_jobs : int (default: 1) number of cpu processes used to perform the test.

spline_df : int (default: 5) dimension of the basis used to represent the smooth term.

fdr_cut : float (default: 0.0001) FDR (Benjamini-Hochberg adjustment) cutoff on significance; significance if FDR < fdr_cut.

A_cut : int (default: 1) amplitude is max of predicted value minus min of predicted value by GAM. significance if A > A_cut.

st_cut : float (default: 0.8) cutoff on stability (fraction of mappings with significant (fdr,A) pair) of association; significance, significance if st > st_cut.

reapply_filters : bool (default: False) avoid recomputation and reapply fitlers.

plot : bool (default: False) call scf.pl.test_association after the test.

root restrain the test to a subset of the tree (in combination with leaves).

leaves restrain the test to a subset of the tree (in combination with root).

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

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.var['p_val'] p-values from statistical test.

.var['fdr'] corrected values from multiple testing.

.var['st'] proportion of mapping in which feature is significant.

.var['A'] amplitue of change of tested feature.

'.var['signi']` feature is significantly changing along pseuodtime

.uns['stat_assoc_list'] list of fitted features on the tree for all mappings.

Return type anndata.AnnData

scFates.tl.test_association_covariate

scFates.tl.**test_association_covariate**(*adata*, *group_key*, *copy=False*, ***kwargs*)
Separately test for associated features for each covariates on the same trajectory path.

Parameters

adata : AnnData Annotated data matrix.

group_key : str key in *.obs* for the covariates to test.

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

****kwargs** Arguments passed to *scFates.tl.test_association()*

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.var['cov_pval' or 'covtrend_pval'] pvalues extracted from tests.

.var['cov_fdr' or 'covtrend_fdr'] FDR extracted from the pvalues.

.var['cov_signi' or 'covtrend_signi'] is the feature significant.

.var['A->B_lfc'] logfoldchange in expression between covariate A and B.

Return type anndata.AnnData

scFates.tl.test_association_monocle3

scFates.tl.**test_association_monocle3**(*adata*, *qval_cut=0.0001*, *n_jobs=1*, *copy=False*, ***kwargs*)
Determine a set of genes significantly associated with the trajectory.

the function *graph_test monocle3* R package is called, using *neighbor_graph = 'principal_graph'* parameter. The statistic tells you whether cells at nearby positions on a trajectory will have similar (or dissimilar) expression levels for the gene being tested.

Parameters

adata : AnnData Annotated data matrix.

qval_cut : float (default: 0.0001) cutoff on the corrected p-values to consider a feature as significant.

n_jobs : bool (default: 1) number of cpu processes used to perform the test.

spline_df dimension of the basis used to represent the smooth term.

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

Returns

adata – if *copy=True* it returns or else add fields to *adata*: *.var['status']*

The feature could be tested.

.var['p_value'] p-values from statistical test.

.var['morans_test_statistic'] Statistics of the Moran test

.var['morans_I'] Moran's I is a measure of multi-directional and multi-dimensional spatial autocorrelation.

.var['q_value'] corrected values from multiple testing.

'.var['signi']` feature has q_value below cutoff.

Return type anndata.AnnData

scFates.tl.fit

scFates.tl.**fit**(*adata*, *features=None*, *layer=None*, *n_map=1*, *n_jobs=1*, *gamma=1.5*, *save_raw=True*, *copy=False*)

Model feature expression levels as a function of tree positions.

The models are fit using *mgcv* R package. Note that since adata can currently only keep the same dimensions for each of its layers. While the dataset is subsetted to keep only significant features, the unsubsetted dataset is kept in adata.raw (save_raw parameter).

Parameters

adata : AnnData Annotated data matrix.

layer : str | NoneOptional[str] (default: None) adata layer to use for the fitting.

n_map : int (default: 1) number of cell mappings from which to do the test.

n_jobs : int (default: 1) number of cpu processes used to perform the test.

gamma : float (default: 1.5) stringency of penalty.

saveraw save the unsubsetted anndata to adata.raw

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

Returns

adata – if *copy=True* it returns subsetted or else subset (keeping only significant features) and add fields to *adata*:

.layers['fitted'] fitted features on the trajectory for all mappings.

Return type anndata.AnnData

scFates.tl.cluster

scFates.tl.**cluster**(*adata*, *layer='fitted'*, *n_neighbors=20*, *n_pcs=50*, *metric='cosine'*, *resolution=1*, *device='cpu'*, *copy=False*)

Cluster features. Uses scanpy backend when using cpu, and cuml when using gpu. Dataset is transposed, PCA is calulcated and a nearest neighbor graph is generated from PC space. Leiden algorithm is used for community detection.

Parameters

adata : AnnData Annotated data matrix.

layer Layer of feature to calculate clusters, by default fitted features

n_neighbors : int (default: 20) Number of neighbors.

n_pcs : int (default: 50) Number of PC to keep for PCA.

metric : str (default: 'cosine') distance metric to use for clustering.

resolution : float (default: 1) Resolution parameter for leiden algorithm.

device : str (default: 'cpu') run the analysis on 'cpu' with phenograph, or on 'gpu' with grapheno.

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

Returns

adata – if *copy=True* it returns subsetted or else subset (keeping only significant features) and add fields to *adata*: *.var['cluters']*

module assignments for features.

Return type anndata.AnnData

scFates.tl.test_covariate

scFates.tl.**test_covariate**(*adata*, *group_key*, *features=None*, *seg=None*, *layer=None*, *trend_test=False*, *fdr_cut=0.01*, *n_jobs=1*, *n_map=1*, *copy=False*)

Test for branch differential gene expression between covariates on the same trajectory path.

Test of amplitude difference

The same is used as in *scFates.tl.test_fork()*. This uses the following model :

$g_i \sim \ s(pseudotime) + s(pseudotime) : Covariate + Covariate$

Where $s(.)$ denotes the penalized regression spline function and $s(pseudotime) : Covariate$ denotes interaction between the smoothed pseudotime and covariate terms. From this interaction term, the p-value is extracted.

Test of trend difference

Inspired from a preprint [Ji22], this test compares the following full model:

$g_i \sim \ s(pseudotime) + s(pseudotime) : Covariate + Covariate$

to the following reduced one:

$g_i \sim \ s(pseudotime) + s(pseudotime) + Covariate$

Comparison is done using ANOVA

Parameters

adata : AnnData Annotated data matrix.

group_key : str key in *.obs* for the covariates to test.

features : Iterable | NoneOptional[Iterable] (default: None) Which features to test (all significants by default).

seg : str | NoneOptional[str] (default: None) In the case of a tree, which segment to use for such test.

layer : str | NoneOptional[str] (default: None) layer to use for the test

trend_test : bool (default: False) Whether to perform the trend test instead of amplitude test.

n_jobs : int (default: 1) number of cpu processes used to perform the test.

n_map : int (default: 1) number of cell mappings from which to do the test (not implemented yet).

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

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.var['cov_pval' or 'covtrend_pval'] pvalues extracted from tests.

.var['cov_fdr' or 'covtrend_fdr'] FDR extracted from the pvalues.

.var['cov_signi' or 'covtrend_signi'] is the feature significant.

.var['A->B_lfc'] logfoldchange in expression between covariate A and B.

Return type anndata.AnnData

scFates.tl.linearity_deviation

scFates.tl.**linearity_deviation**(*adata*, *start_milestone*, *end_milestone*, *percentiles=[20, 80]*, *n_jobs=1*, *n_map=1*, *plot=False*, *basis='X_umap'*, *copy=False*)

Identifies genes that specifically characterize a given transition but not the progenitors neither the progenies.

This approach has been developped in the following study [Kameneva21]. Designed to test whether a bridge/transition is the result of a doublet population only, this test checks if a gene expression occuring in the transition/bridge could be explained by a linear mixture of expressions of that gene in progenitors and progenies. The gene expression profile of each cell of the bridge is modeled as a linear combination of mean gene expression profiles in progenitors and progenies.

For each gene in each cell in bridge is calculated the magnitude of the residuals not explained by the model. The mean residuals across all cells in the transition/bridge is then normalized to the standard deviation of the expression of a given gene. The obtained normalized mean residual values is used to prioritize the genes with distinctive expression patterns in the bridge population.

Parameters

adata : AnnData Annotated data matrix.

start_milestone tip defining progenitor branch.

end_milestone tips defining the progeny branch.

percentiles pseudotime percentiles to define the progenitor and progeny populations

n_jobs : int (default: 1) number of cpu processes used to perform the test.

n_map : int (default: 1) number of cell mappings from which to do the test.

plot : bool (default: False) plot the cells selection according to percentiles.

basis : str (default: 'X_umap') basis to use in case of plotting

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

Returns

adata – if *copy=True* it returns subsetted or else subset (keeping only significant features) and add fields to *adata*:

.var['A->B_rss'] pearson residuals of the linear fit.

.obs['A->B_lindev_sel'] cell selections used for the test.

Return type anndata.AnnData

scFates.tl.unroll_circle

scFates.tl.**unroll_circle**(*adata*, *copy=False*)

Unroll circle to get full spectrum of pseudotime values along it.

Parameters

adata : AnnData Annotated data matrix.

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

Returns

adata – if *copy=True* it returns or else update fields to *adata*:

.obs['t'] assigned pseudotime value.

.obs['seg'] assigned segment of the tree.

.obs['milestone'] assigned region surrounding forks and tips.

.uns['graph']['pp_seg'] segments network information.

.uns['graph']['pp_info'] for each PP, its distance vs root and segment assignment.

Return type anndata.AnnData

1.2.5 Bifurcation analysis

Branch specific feature extraction and classification

tl.test_fork(adata, root_milestone, milestones)	Test for branch differential gene expression and differential upregulation from progenitor to terminal state.
tl.branch_specific(adata, root_milestone, ...)	Assign genes differentially expressed between two post-bifurcation branches.
tl.activation(adata, root_milestone, milestones)	Identify pseudotime of activation of branch-specififc features.
tl.activation_lm(adata, root_milestone, ...)	A more robust version of *tl.activation*.

scFates.tl.test_fork

scFates.tl.**test_fork**(*adata*, *root_milestone*, *milestones*, *features=None*, *rescale=False*, *layer=None*, *n_jobs=1*, *n_map=1*, *copy=False*)

Test for branch differential gene expression and differential upregulation from progenitor to terminal state.

First, differential gene expression between two branches is performed. The following model is used:

$g_i \sim\ s(pseudotime) + s(pseudotime) : Branch + Branch$

Where $s()$ denotes the penalized regression spline function and $s(pseudotime) : Branch$ denotes interaction between the smoothed pseudotime and branch terms. From this interaction term, the p-value is extracted.

Then, each feature is tested for its upregulation along the path from progenitor to terminal state, using the linear model $g_i \sim\ pseudotime$.

Parameters

adata : AnnData Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

features : str | NoneOptional[str] (default: None) Which features to test (all by default).

rescale : bool (default: False) By default, analysis restrict to only cells having a pseudotime lower than the shortest branch maximum pseudotime, this can be avoided by rescaling the post bifurcation pseudotime of both branches to 1.

layer : str | NoneOptional[str] (default: None) layer to use for the test

n_map : int (default: 1) number of cell mappings from which to do the test.

n_jobs : int (default: 1) number of cpu processes used to perform the test.

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

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.uns['root_milestone->milestoneA<>milestoneB']['fork'] DataFrame with fork test results.

Return type anndata.AnnData

scFates.tl.branch_specific

scFates.tl.**branch_specific**(*adata*, *root_milestone*, *milestones*, *effect=None*, *stf_cut=0.7*, *up_A=0*, *up_p=0.05*, *copy=False*)

Assign genes differentially expressed between two post-bifurcation branches.

Parameters

adata : AnnData Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

effect : float | NoneOptional[float] (default: None) minimum expression differences to call gene as differentially upregulated.

stf_cut : float (default: 0.7) fraction of projections when gene passed fdr < 0.05.

up_A : float (default: 0) minimum expression increase at derivative compared to progenitor branches to call gene as branch-specific.

up_p : float (default: 0.05) p-value of expression changes of derivative compared to progenitor branches to call gene as branch-specific.

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

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.uns['root_milestone->milestoneA<>milestoneB']['fork']['branch'] assigned branch.

Return type anndata.AnnData

scFates.tl.activation

scFates.tl.**activation**(*adata*, *root_milestone*, *milestones*, *deriv_cut=0.15*, *pseudotime_offset=0*, *nwin=20*, *steps=5*, *n_map=1*, *copy=False*, *n_jobs=- 1*, *layer=None*)

Identify pseudotime of activation of branch-specifific features.

This aims in classifying the genes according to their their activation timing compared to the pseudotime of the bifurcation. Any feature activated before the bifurcation is considered as 'early', others are considered 'late'.

This is done by separating the path into bins of equal pseudotime, and then identifying successive bins having a change in expression higher than the parameter *deriv_cut*.

Parameters

adata : AnnData Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

deriv_cut : float (default: 0.15) a first passage of derivative at this cutoff (in proportion to the full dynamic range of the fitted feature) is considered as activation timing

pseudotime_offset : float (default: 0) consider a feature as early if it gets activated before: pseudotime at bifurcation-pseudotime_offset.

nwin : int (default: 20) windows of pseudotime to use for assessing activation timimg

steps : int (default: 5) number of steps dividing a window for that will slide along the pseudotime

n_map : int (default: 1) number of cell mappings from which to do the test.

n_jobs number of cpu processes used to perform the test.

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

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.uns['root_milestone->milestoneA<>milestoneB']['fork']['module'] classify feature as 'early' or 'late'.

.uns['root_milestone->milestoneA<>milestoneB']['fork']['activation'] pseudotime of activationh.

Return type anndata.AnnData

scFates.tl.activation_lm

scFates.tl.**activation_lm**(*adata*, *root_milestone*, *milestones*, *fdr_cut=0.05*, *stf_cut=0.8*, *pseudotime_offset=0*, *n_map=1*, *copy=False*, *n_jobs=- 1*, *layer=None*)

A more robust version of *tl.activation*.

This is considered to be a more robust version of *scFates.tl.activation()*. The common path between the two fates is retained for analysis, each feature is tested for its upregulation along the path from progenitor to the fork, using the linear model $g_i \sim pseudotime$.

Parameters

adata : AnnData Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

stf_cut : float (default: 0.8) fraction of projections when gene passed fdr < 0.05.

pseudotime_offset : float (default: 0) consider cells to retain up to the pseudotime_fork-pseudotime_offset.

n_map : int (default: 1) number of cell mappings from which to do the test.

n_jobs number of cpu processes used to perform the test.

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

layer layer to use for the test

Returns

adata – if *copy=True* it returns or else add fields to *adata*:

.uns['root_milestone->milestoneA<>milestoneB']['fork']['module'] classify feature as 'early' or 'late'.

.uns['root_milestone->milestoneA<>milestoneB']['fork']['slope'] slope calculated by the linear model.

.uns['root_milestone->milestoneA<>milestoneB']['fork']['pval'] pval resulting from linear model.

.uns['root_milestone->milestoneA<>milestoneB']['fork']['fdr'] corrected fdr value.

.uns['root_milestone->milestoneA<>milestoneB']['fork']['prefork_signi'] proportion of projections where fdr<0.05.

Return type anndata.AnnData

Correlation analysis

tl.module_inclusion(adata, root_milestone, ...)	Estimates the pseudotime onset of a feature within its fate-specific module.
tl.slide_cells(adata, root_milestone, milestones)	Assign cells in a probabilistic manner to non-intersecting windows along pseudotime.
tl.slide_cors(adata, root_milestone, milestones)	Obtain gene module correlations in the non-intersecting windows along pseudotime.
tl.synchro_path(adata, root_milestone, ...)	Estimates pseudotime trends of local intra- and inter-module correlations of fates-specific modules.
tl.synchro_path_multi(adata, root_milestone, ...)	Wrappers that call *tl.synchro_path* on the pairwise combination of all selected branches.

scFates.tl.module_inclusion

scFates.tl.**module_inclusion**(*adata*, *root_milestone*, *milestones*, *w=300*, *step=30*, *pseudotime_offset=0*, *module='early'*, *n_perm=10*, *n_map=1*, *map_cutoff=0.8*, *n_jobs=1*, *alp=10*, *autocor_cut=0.95*, *iterations=15*, *parallel_mode='window'*, *identify_early_features=False*, *layer=None*, *perm=False*, *winp=10*, *copy=False*)

Estimates the pseudotime onset of a feature within its fate-specific module.

Parameters

adata Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

w : int (default: 300) local window, in number of cells, to estimate correlations.

step : int (default: 30) steps, in number of cells, between local windows.

pseudotime_offset : all | floatUnion[all, float] (default: 0) restrict the cell selection up to a pseudotime offset after the fork

module : {'all', 'early'}Literal['all', 'early'] (default: 'early') restrict the gene selection to already classified early genes.

n_perm : int (default: 10) number of permutations used to estimate the background local correlations.

n_map : int (default: 1) number of probabilistic cells projection to use for estimates.

map_cutoff : float (default: 0.8) proportion of mapping in which inclusion pseudotimne was found for a given feature to keep it.

n_jobs : int (default: 1) number of cpu processes to perform estimates.

alp : int (default: 10) parameter regulating stringency of inclusion event.

autocor_cut : float (default: 0.95) cutoff on correlation of inclusion times between sequential iterations of the algorithm to stop it.

iterations : int (default: 15) maximum number of iterations of the algorithm.

parallel_mode : {'window', 'mappings'}Literal['window', 'mappings'] (default: 'window') whether to run in parallel over the windows of cells or the mappings.

identify_early_features : bool (default: False) classify a feature as early if its inclusion pseudotime is before the bifurcation

layer adata layer to use for estimates.

perm : bool (default: False) do local estimates for locally permuted expression matrix.

winp : int (default: 10) window of permutation in cells.

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

Returns

adata – if *copy=True* it returns subsetted or else subset (keeping only significant features) and add fields to *adata*:

.uns['root_milestone->milestoneA<>milestoneB']['module_inclusion'] Dataframes ontaining inclusion timing for each gene (rows) in each probabilistic cells projection (columns).

.uns['root_milestone->milestoneA<>milestoneB']['fork'] Updated with 'inclusion' pseudotime column and 'module column if *identify_early_features=True*'

Return type anndata.AnnData

scFates.tl.slide_cells

scFates.tl.**slide_cells**(*adata*, *root_milestone*, *milestones*, *win=50*, *mapping=True*, *copy=False*, *ext=False*)

Assign cells in a probabilistic manner to non-intersecting windows along pseudotime.

Parameters

adata : AnnData Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

win : int (default: 50) number of cell per local pseudotime window.

mapping : bool (default: True) project cells onto tree pseudotime in a probabilistic manner.

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

ext : bool (default: False) Output the list externally instead of writting to anndata

Returns

adata – if *copy=True* it returns subsetted or else subset (keeping only significant features) and add fields to *adata*:

.uns['root_milestone->milestoneA<>milestoneB']['cell_freq'] List of np.array containing probability assignment of cells on non intersecting windows.

Return type anndata.AnnData

scFates.tl.slide_cors

scFates.tl.**slide_cors**(*adata*, *root_milestone*, *milestones*, *genesetA=None*, *genesetB=None*, *perm=False*, *layer=None*, *copy=False*)

Obtain gene module correlations in the non-intersecting windows along pseudotime.

Parameters

adata : AnnData Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones : List tips defining the progenies branches.

layer : str | NoneOptional[str] (default: None) adata layer from which to compute the correlations.

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

Returns

adata – if *copy=True* it returns subsetted or else subset (keeping only significant features) and add fields to *adata*:

.uns['root_milestone->milestoneA<>milestoneB']['corAB'] Dataframe containing gene-gene correlation modules.

Return type anndata.AnnData

scFates.tl.synchro_path

scFates.tl.**synchro_path**(*adata*, *root_milestone*, *milestones*, *genesetA=None*, *genesetB=None*, *n_map=1*, *n_jobs=None*, *layer=None*, *perm=True*, *w=200*, *step=30*, *winp=10*, *knots=10*, *copy=False*)

Estimates pseudotime trends of local intra- and inter-module correlations of fates-specific modules.

Parameters

adata : AnnData Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

n_map number of probabilistic cells projection to use for estimates.

n_jobs number of cpu processes to perform estimates (per mapping).

layer : str | NoneOptional[str] (default: None) adata layer to use for estimates.

perm estimate control trends for local permutations instead of real expression matrix.

w local window, in number of cells, to estimate correlations.

step steps, in number of cells, between local windows.

winp window of permutation in cells.

knots number of knots for GAM fit of corAB on cells pre-fork

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

Returns

adata – if *copy=True* it returns subsetted or else subset (keeping only significant features) and add fields to *adata*:

.uns['root_milestone->milestoneA<>milestoneB']['synchro'] Dataframe containing mean local gene-gene correlations of all possible gene pairs inside one module, or between the two modules.

.obs['intercor root_milestone->milestoneA<>milestoneB'] loess fit of inter-module mean local gene-gene correlations prior to bifurcation

Return type anndata.AnnData

scFates.tl.synchro_path_multi

scFates.tl.**synchro_path_multi**(*adata*, *root_milestone*, *milestones*, *copy=False*, ***kwargs*)

Wrappers that call *tl.synchro_path* on the pairwise combination of all selected branches.

Parameters

adata : AnnData Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

kwargs arguments to pass to tl.synchro_path.

Returns

adata – if *copy=True* it returns subsetted or else subset (keeping only significant features) and add fields to *adata*:

.uns['root_milestone->milestoneA<>milestoneB']['synchro'] Dataframe containing mean local gene-gene correlations of all possible gene pairs inside one module, or between the two modules.

.obs['intercor root_milestone->milestoneA<>milestoneB'] loess fit of inter-module mean local gene-gene correlations prior to bifurcation

Return type anndata.AnnData

1.2.6 Plot

Trajectory

pl.graph(adata[, basis, size_nodes, ...])	Project principal graph onto embedding.
pl.trajectory(adata[, basis, ...])	Project trajectory onto embedding.
pl.trajectory_3d(adata[, basis, color, ...])	Project trajectory onto 3d embedding.
pl.dendrogram(adata[, root_milestone, ...])	Plot the single-cell dendrogram embedding.
pl.milestones(adata[, basis, annotate, ...])	Display the milestone graph in PAGA style.

scFates.pl.graph

scFates.pl.**graph**(*adata*, *basis=None*, *size_nodes=None*, *alpha_nodes=1*, *linewidth=2*, *alpha_seg=1*, *color_cells=None*, *tips=True*, *forks=True*, *nodes=[]*, *ax=None*, *show=None*, *save=None*, ***kwargs*)

Project principal graph onto embedding.

Parameters

adata : AnnData Annotated data matrix.

basis : str | NoneOptional[str] (default: None) Name of the *obsm* basis to use.

size_nodes : float | NoneOptional[float] (default: None) Size of the projected prinicpal points.

alpha_nodes : float (default: 1) Alpha of nodes.

linewidth : float (default: 2) Line width of the segments.

alpha_seg : float (default: 1) Alpha of segments.

color_cells : str | NoneOptional[str] (default: None) cells color

tips : bool (default: True) display tip ids.

forks : bool (default: True) display fork ids.

nodes : List | NoneOptional[List] (default: []) display any node id.

ax Add plot to existing ax

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

kwargs arguments to pass to scanpy.pl.embedding()

Returns

Return type If *show==False* a Axes

scFates.pl.trajectory

scFates.pl.**trajectory**(*adata*, *basis=None*, *root_milestone=None*, *milestones=None*, *color_seg='t'*, *cmap_seg='viridis'*, *layer_seg='fitted'*, *perc_seg=None*, *color_cells=None*, *scale_path=1*, *arrows=False*, *arrow_offset=10*, *show_info=True*, *ax=None*, *show=None*, *save=None*, ***kwargs*)

Project trajectory onto embedding.

Parameters

adata : AnnData Annotated data matrix.

basis : str | NoneOptional[str] (default: None) Name of the *obsm* basis to use.

root_milestone : str | NoneOptional[str] (default: None) tip defining progenitor branch.

milestones : str | NoneOptional[str] (default: None) tips defining the progenies branches.

col_seg color trajectory segments.

layer_seg : str | NoneOptional[str] (default: 'fitted') layer to use when coloring seg with a feature.

perc_seg : List | NoneOptional[List] (default: None) percentile cutoffs for segments.

color_cells : str | NoneOptional[str] (default: None) cells color.

scale_path : float (default: 1) changes the width of the path

arrows : bool (default: False) display arrows on segments (positioned at half pseudotime distance).

arrow_offset : int (default: 10) arrow offset in number of nodes used to obtain its direction.

show_info : bool (default: True) display legend/colorbar.

ax Add plot to existing ax

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

kwargs arguments to pass to `scanpy.pl.embedding()`.

Returns

Return type If *show==False* a `Axes`

scFates.pl.trajectory_3d

scFates.pl.**trajectory_3d**(*adata*, *basis='umap3d'*, *color=None*, *traj_width=5*, *cell_size=2*, *figsize=(900, 900)*, *cmap=None*)

Project trajectory onto 3d embedding.

Parameters

adata : AnnData Annotated data matrix.

basis : str (default: 'umap3d') Name of the *obsm* basis to use.

color : str | NoneOptional[str] (default: None) cells color.

traj_width : int (default: 5) segments width.

cell_size : int (default: 2) cell size.

figsize : tuple (default: (900, 900)) figure size in pixels.

cmap colormap of the cells.

Returns

Return type an interactive plotly graph figure.

scFates.pl.dendrogram

scFates.pl.**dendrogram**(*adata*, *root_milestone=None*, *milestones=None*, *color_milestones=False*, *color_seg='k'*, *linewidth_seg=3*, *alpha_seg=0.3*, *tree_behind=False*, *show_info=True*, *show=None*, *save=None*, ***kwargs*)

Plot the single-cell dendrogram embedding.

Parameters

adata Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

color_milestones : bool (default: False) color the cells with gradients combining pseudotime and milestones.

color_seg : str (default: 'k') color the segments, either a color, or 'seg' colors from obs.seg.

linewidth_seg : float (default: 3) linewidth of the segments.

alpha_seg : float (default: 0.3) alpha of the segments.

tree_behind : bool (default: False) whether to plot the segment in front or behind the cells.

show_info : bool (default: True) display the colorbar or not.

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

kwargs arguments to pass to `scanpy.pl.embedding()`.

Returns

Return type If *show==False* an object of `Axes`

scFates.pl.milestones

scFates.pl.**milestones**(*adata*, *basis=None*, *annotate=False*, *title='milestones'*, *subset=None*, *ax=None*, *show=None*, *save=None*, ***kwargs*)

Display the milestone graph in PAGA style.

Parameters

adata Annotated data matrix.

basis : str | NoneOptional[str] (default: None) Reduction to use for plotting.

annotate : bool (default: False) Display milestone labels on the plot.

title : str (default: 'milestones') Plot title to display.

subset : Iterable | NoneOptional[Iterable] (default: None) Subset cells.

ax Add plot to existing ax.

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

kwargs arguments to pass to `matplotlib.pyplot.scatter()`.

Returns

Return type If *show==False* an object of `Axes`

Pseudotime features

pl.test_association(adata[, log_A])	Plot a set of fitted features over pseudotime.
pl.single_trend(adata[, feature, ...])	Plot a single feature fit over pseudotime.
pl.trends(adata[, features, cluster, ...])	Plot a set of fitted features over pseudotime.
pl.matrix(adata, features[, nbins, layer, ...])	Plot a set of features as per-segment matrix plots of binned pseudotimes.
pl.linearity_deviation(adata, ...[, ...])	Plot the results generated by *tl.linearity_deviation*.
pl.binned_pseudotime_meta(adata, key[, ...])	Plot a dot plot of proportion of cells from a given category over binned sections of pseudotime.

scFates.pl.test_association

scFates.pl.**test_association**(*adata*, *log_A=False*)

Plot a set of fitted features over pseudotime.

Parameters

adata : AnnData Annotated data matrix.

log_A : bool (default: False) change the xaxis scale to log.

Returns

Return type just the plot.

scFates.pl.single_trend

scFates.pl.**single_trend**(*adata*, *feature=None*, *root_milestone=None*, *milestones=None*, *module=None*, *branch=None*, *basis='umap'*, *ylab='expression'*, *color_exp=None*, *alpha_expr=0.3*, *size_expr=2*, *fitted_linewidth=2*, *layer=None*, *cmap_seg='RdBu_r'*, *cmap_cells='RdBu_r'*, *plot_emb=True*, *wspace=None*, *figsize=(8, 4)*, *ax_trend=None*, *ax_emb=None*, *show=None*, *save=None*, ***kwargs*)

Plot a single feature fit over pseudotime.

Parameters

adata : AnnData Annotated data matrix.

feature : str | NoneOptional[str] (default: None) Name of the fitted feature.

root_milestone : str | NoneOptional[str] (default: None) if plotting module instead of feature, tip defining progenitor branch.

milestones : str | NoneOptional[str] (default: None) if plotting module instead of feature, tips defining the progenies branches.

module : None | {'early', 'late'}Optional[Literal['early', 'late']] (default: None) if plotting module instead of feature, whether to plot early or late modules.

branch : str | NoneOptional[str] (default: None) if plotting module instead of feature, plot fitted milestone-specific module.

basis : str (default: 'umap') Name of the *obsm* basis to use.

ylab : str (default: 'expression') ylabel of right plot.

colo_rexp color of raw datapoints on right plot.

alpha_expr : float (default: 0.3) alpha of raw datapoints on right plot.

size_expr : float (default: 2) size of raw datapoints on right plot.

fitted_linewidth : float (default: 2) linewidth of fitted line on right plot.

layer : str | NoneOptional[str] (default: None) layer to plot for the raw datapoints.

cmap_seg : str (default: 'RdBu_r') colormap for trajectory segments on left plot.

cmap_cells : str (default: 'RdBu_r') colormap for cells on left plot.

plot_emb : bool (default: True) plot the emb on the left side.

wspace : float | NoneOptional[float] (default: None) width space between emb and heatmap.

figsize : tuple (default: (8, 4)) figure size in inches.

ax_trend existing ax for trends, only works when emb plot is disabled.

ax_emb existing ax for embedding plot.

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

Returns

Return type If *show==False* a tuple of two `Axes`

scFates.pl.trends

scFates.pl.**trends**(*adata*, *features=None*, *cluster=None*, *highlight_features='A'*, *n_features=10*, *root_milestone=None*, *milestones=None*, *module=None*, *branch=None*, *annot=None*, *title=''*, *feature_cmap='RdBu_r'*, *pseudo_cmap='viridis'*, *plot_emb=True*, *plot_heatmap=True*, *wspace=None*, *show_segs=True*, *basis=None*, *heatmap_space=0.5*, *offset_names=0.15*, *fontsize=9*, *style='normal'*, *ordering='pearson'*, *ord_thre=0.7*, *figsize=None*, *axemb=None*, *show=None*, *output_mean=False*, *save=None*, *return_genes=None*, ***kwargs*)

Plot a set of fitted features over pseudotime.

Parameters

adata : AnnData Annotated data matrix.

features Name of the fitted features.

highlight_features : List | {'A', 'fdr'}Union[List, Literal['A', 'fdr']] (default: 'A') which features will be annotated on the heatmap, by default, features with highest amplitude are shown.

n_features : int (default: 10) number of top features to show if no list are provided.

root_milestone : str | NoneOptional[str] (default: None) tip defining progenitor branch.

milestones : str | NoneOptional[str] (default: None) tips defining the progenies branches.

module : None | {'early', 'late'}Optional[Literal['early', 'late']] (default: None) if bifurcation analysis as been performed, subset features to a specific module.

branch : str | NoneOptional[str] (default: None) if bifurcation analysis as been performed, subset features to a specific milestone.

annot : str | NoneOptional[str] (default: None) adds an annotation row on top of the heatmap.

title : str (default: '') add a title.

feature_cmap : str (default: 'RdBu_r') colormap for features.

pseudo_cmap : str (default: 'viridis') colormap for pseudotime.

plot_emb : bool (default: True) call pl.trajectory on the left side.

plot_heatmap : bool (default: True) show heatmap on the right side.

wspace : float | NoneOptional[float] (default: None) width space between emb and heatmap.

show_segs : bool (default: True) display segments on emb.

basis : str | NoneOptional[str] (default: None) Name of the *obsm* basis to use if plot_emb is True.

heatmap_space : float (default: 0.5) how much space does the heatmap take, in proportion of the whole plot space.

offset_names : float (default: 0.15) how far on the right the annotated features should be displayed, in proportion of the heatmap space.

fontsize : int (default: 9) font size of the feature annotations.

style : {'normal', 'italic', 'oblique'}Literal['normal', 'italic', 'oblique'] (default: 'normal') font style.

ordering : None | {'pearson', 'spearman', 'quantile', 'max'}Optional[Literal['pearson', 'spearman', 'quantile strategy to order the features on heatmap, quantile takes the mean pseudotime of the choosen value.

ord_thre for 'max': proportion of maximum of fitted value to consider to compute the mean pseudotime. for 'quantile': quantile to consider to compute the mean pseudotime. for 'pearson'/'spearman': proportion of max value to assign starting cell.

figsize : None | tupleOptional[tuple] (default: None) figure size.

axemb existing ax for plotting emb

output_mean : bool (default: False) output mean fitted values to adata.

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

return_genes : bool | NoneOptional[bool] (default: None) return list of genes following the order displayed on the heatmap.

****kwargs** if *plot_emb=True*, arguments passed to *scFates.pl.trajectory()* or *scFates.pl.dendrogram()* if *basis="dendro"*

Returns

Return type If *show*==*False* a matrix of `Axes`

scFates.pl.matrix

scFates.pl.**matrix**(*adata*, *features*, *nbins=5*, *layer='fitted'*, *norm='max'*, *annot_var=False*, *annot_top=True*, *link_seg=True*, *root_milestone=None*, *milestones=None*, *feature_style='normal'*, *feature_spacing=1*, *cmap=None*, *colorbar=True*, *colorbar_title=None*, *figsize=None*, *return_data=False*, *show=None*, *save=None*, ***kwargs*)

Plot a set of features as per-segment matrix plots of binned pseudotimes.

Parameters

adata : AnnData Annotated data matrix.

features : Sequence Name of the features.

nbins : int (default: 5) Number of pseudotime bins per segment.

layer : str (default: 'fitted') Layer to use for the expression to display.

norm : {'max', 'minmax', 'none'}Literal['max', 'minmax', 'none'] (default: 'max') How to normalize the expression.

annot_var : bool (default: False) Annotate overall tree amplitude of expression of the marker (from .var['A']).

annot_top : bool (default: True) Display milestones gradient for each segment on top of plots.

link_seg : bool (default: True) Link the segment together to keep track of the the tree progression.

root_milestone : str | NoneOptional[str] (default: None) tip defining progenitor branch.

milestones : Iterable | NoneOptional[Iterable] (default: None) tips defining the progenies branches.

feature_style : str (default: 'normal') Font style of the feature labels.

feature_spacing : float (default: 1) When figsize is None, controls the the height of each rows.

cmap : str | NoneOptional[str] (default: None) colormap to use, by default is plt.rcParams["image.cmap"].

colorbar : bool (default: True) Show the colorbar.

colorbar_title : str | NoneOptional[str] (default: None) Set a custom colorbar title.

figsize : None | tupleOptional[tuple] (default: None) Custom figure size.

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

save_genes save list of genes following the order displayed on the heatmap.

****kwargs** arguments passed to `scanpy.pl.MatrixPlot`

Returns

Return type If *show*==*False* an array of `Axes`

scFates.pl.linearity_deviation

scFates.pl.**linearity_deviation**(*adata*, *start_milestone*, *end_milestone*, *ntop_genes=30*, *fontsize=8*, *show=None*, *save=None*)

Plot the results generated by *tl.linearity_deviation*.

Parameters

adata : AnnData Annotated data matrix.

start_milestone tip defining the starting point of analysed segment.

end_milestone tip defining the end point of analysed segment.

ntop_genes : int (default: 30) number of top genes to show.

fontsize : int (default: 8) Fontsize for gene names.

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

Returns

Return type If *show==False* a matrix of `Axes`

scFates.pl.binned_pseudotime_meta

scFates.pl.**binned_pseudotime_meta**(*adata*, *key*, *nbins=20*, *rotation=0*, *show_colorbar=False*, *rev=False*, *cmap='viridis'*, *ax=None*, *show=None*, *save=None*)

Plot a dot plot of proportion of cells from a given category over binned sections of pseudotime.

Parameters

adata Annotated data matrix.

key category to study.

nbins : int (default: 20) Number of pseudotime bin to generate.

rotation : int (default: 0) rotaton of the category labels.

show_colorbar : bool (default: False) display pseudotime colorbar.

cmap colormap of the pseudotime cbar.

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

Returns

Return type If *show==False* a tuple of `Axes`

Bifurcation & correlation analysis

pl.modules(adata, root_milestone, milestones)	Plot the mean expression of the early and late modules.
pl.test_fork(adata, root_milestone, milestones)	Plot results generated from tl.test_fork.
pl.slide_cors(adata, root_milestone, milestones)	Plot results generated from tl.slide_cors.
pl.synchro_path(adata, root_milestone, ...)	Plot results generated from tl.synchro_path.
pl.module_inclusion(adata, root_milestone, ...)	Plot results generated from tl.module_inclusion.

scFates.pl.modules

scFates.pl.**modules**(*adata*, *root_milestone*, *milestones*, *color='milestones'*, *module='all'*, *show_traj=False*, *layer=None*, *smooth=False*, *ax_early=None*, *ax_late=None*, *show=None*, *save=None*, ***kwargs*)

Plot the mean expression of the early and late modules.

Parameters

adata : AnnData Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

color : str (default: 'milestones') color the cells with variable from adata.obs.

module : {'early', 'late', 'all'}Literal['early', 'late', 'all'] (default: 'all') whether to show early, late or both modules.

show_traj : bool (default: False) show trajectory on the early module plot.

layer : str | NoneOptional[str] (default: None) layer to use to compute mean of module.

smooth : bool (default: False) whether to smooth the data using knn graph.

ax_early existing axes for early module.

ax_late existing axes for late module.

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

kwargs arguments to pass to *scFates.pl.trajectory()* if *show_traj=True*, else to scanpy.pl.embedding()

Returns

Return type If *show==False* a tuple of Axes

scFates.pl.test_fork

scFates.pl.**test_fork**(*adata*, *root_milestone*, *milestones*, *col=None*, *show=None*, *save=None*)

Plot results generated from tl.test_fork.

Parameters

adata : AnnData Annotated data matrix.

root_milestone tip defining progenitor branch.

col : None | listOptional[list] (default: None) color of the two sets of genes, by default to the color of the end milestones.

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

Returns

Return type If *show==False* a matrix of Axes

scFates.pl.slide_cors

scFates.pl.**slide_cors**(*adata*, *root_milestone*, *milestones*, *col=None*, *basis='umap'*, *win_keep=None*, *frame_emb=True*, *focus=None*, *top_focus=4*, *labels=None*, *fig_height=6*, *fontsize=16*, *fontsize_focus=18*, *point_size=20*, *show=None*, *save=None*, *kwargs_text={}*, *kwargs_con={}*, *kwargs_adjust={}*, ***kwargs*)

Plot results generated from tl.slide_cors.

Parameters

adata : AnnData Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

genesetA plot correlation with custom geneset.

genesetB plot correlation with custom geneset.

col : None | listOptional[list] (default: None) specify color for the two modules, by default according to their respective milestones.

basis : str (default: 'umap') Name of the *obsm* basis to use.

win_keep : None | listOptional[list] (default: None) plot only a subset of windows.

frame_emb : bool (default: True) add frame around emb plot.

focus add on the right side a scatter focusing one defined window.

top_focus highlight n top markers for each module, having the greatest distance to 0,0 coordinates.

labels : None | tupleOptional[tuple] (default: None) labels defining the two modules, named after the milestones if None, or 'A' and 'B' if less than two milestones is used.

fig_height : float (default: 6) figure height.

fontsize : int (default: 16) repulsion score font size.

fontsize_focus : int (default: 18) fontsize of x and y labels in focus plot.

point_size : int (default: 20) correlation plot point size.

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

kwargs_text : dict (default: {}) parameters for the text annotation of the labels.

kwargs_con : dict (default: {}) parameters passed on the ConnectionPatch linking the focused plot to the rest.

kwargs_adjust : dict (default: {}) parameters passed to adjust_text.

****kwargs** if *basis=dendro*, arguments passed to *scFates.pl.dendrogram()*

Returns

Return type If *show==False* a matrix of Axes

scFates.pl.synchro_path

scFates.pl.**synchro_path**(*adata*, *root_milestone*, *milestones*, *knots=10*, *alpha_multi=0.1*, *knots_multi=None*, *max_t=None*, *figsize=None*, *show=None*, *save=None*)

Plot results generated from tl.synchro_path.

Parameters

adata : AnnData Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

knots : int (default: 10) number of knots for gamfit.

alpha_multi : float (default: 0.1) alpha multiple mapping.

knots_multi : int | NoneOptional[int] (default: None) knots for overall mappings (plot disabled if not set).

max_t : float | NoneOptional[float] (default: None) set max pseudotime to show.

figsize : None | tupleOptional[tuple] (default: None) figure size.

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

Returns

Return type If *show==False* a matrix of Axes

scFates.pl.module_inclusion

scFates.pl.**module_inclusion**(*adata*, *root_milestone*, *milestones*, *bins*, *branch*, *feature=None*, *figsize=(6, 5)*, *max_t='max'*, *perm=False*, *show=None*, *save=None*)

Plot results generated from tl.module_inclusion.

Parameters

adata Annotated data matrix.

root_milestone tip defining progenitor branch.

milestones tips defining the progenies branches.

bins : int Number of bins to separate the pseudotime path.

branch : str Which endpoint to focus on.

feature : str | NoneOptional[str] (default: None) Plot a specific feature.

figsize : tuple (default: (6, 5)) Size of the figure.

max_t : {'fork', 'max'}Literal['fork', 'max'] (default: 'max') Until which pseudotime to limit the plot and binning.

perm : bool (default: False) Show permutation results.

show : bool | NoneOptional[bool] (default: None) show the plot.

save : str | bool | NoneUnion[str, bool, None] (default: None) save the plot.

Returns

Return type If *show==False* a matrix of Axes

1.2.7 Getting analysed data

get.fork_stats(adata, root_milestone, milestones)	Extract statistics from the fork analysis.
get.modules(adata, root_milestone, milestones)	Extract mean expression of identified early and late modules.
get.slide_cors(adata, root_milestone, ...)	Extract statistics from the sliding window correlation analysis.

scFates.get.fork_stats

scFates.get.**fork_stats**(*adata*, *root_milestone*, *milestones*, *module=None*, *branch=None*)

Extract statistics from the fork analysis.

Parameters

adata : AnnData Annotated data matrix.

root_milestone : str tip defining progenitor branch.

milestones : Iterable tips defining the progenies branches.

module : str | NoneOptional[str] (default: None) subset features to a specific module.

branch : str | NoneOptional[str] (default: None) subset features to a specific milestone.

Returns

Return type a pandas.DataFrame extracted from adata.uns.

scFates.get.modules

scFates.get.**modules**(*adata*, *root_milestone*, *milestones*, *layer=None*, *module='all'*)

Extract mean expression of identified early and late modules.

Parameters

adata : AnnData Annotated data matrix.

root_milestone : str tip defining progenitor branch.

milestones : Iterable tips defining the progenies branches.

layer : str | NoneOptional[str] (default: None) layer to use to calculate the mean.

module : {'early', 'late', 'all'}Literal['early', 'late', 'all'] (default: 'all') extract either the early, late or both modules.

Returns

Return type a pandas.DataFrame.

scFates.get.slide_cors

scFates.get.**slide_cors**(*adata*, *root_milestone*, *milestones*, *branch*, *geneset_branch*)
Extract statistics from the sliding window correlation analysis.

Parameters

adata : AnnData Annotated data matrix.

root_milestone : str tip defining progenitor branch.

milestones : Iterable tips defining the progenies branches.

branch : str subset features to a specific milestone.

geneset_branch : str which geneset to show correlations.

Returns

Return type a pandas.DataFrame extracted from adata.uns.

1.3 Release Notes

1.3.1 Version 1.0.7 February 15, 2566/2024

Fixed adjustedText version to avoid error, changed tests to python 3.11

1.3.2 Version 1.0.6 August 26, 2566/2023

Fixed exception in *scFates.tl.slide_cells()*

1.3.3 Version 1.0.5 August 25, 2566/2023

Fix int and bool check when ordering segments in *scFates.pl.trends()*.

1.3.4 Version 1.0.4 August 13, 2566/2023

- updated notebooks.
- relaxed mutli-mapping rule when using elpigraph.
- better handling of milestone renaming.
- prevent grid on module_inclusion plot
- correcting messages displayed by functions.

1.3.5 Version 1.0.3 July 23, 2566/2023

Updated elpigraph version to fix error with networkx, fixed typo.

1.3.6 Version 1.0.2 April 28, 2566/2023

Constrained pandas version requirement (<2.0) to avoid broken functions.

1.3.7 Version 1.0.1 March 10, 2566/2023

Minor fixes to make scFates compatible with the newer versions of matplotlib (3.5+). Constrained networkx requirement to avoid error happening in the last version 3.0.

1.3.8 Version 1.0.0 November 29, 2022

The tool is now published, it is considered stable enough to be released as v1.0.0

- *scFates.pl.trends()* displays an error message if no feature is plotted.
- *scFates.tl.tree()* now accept parameters transfer to elpigraph-python.

1.3.9 Version 0.9.1 August 28, 2022

- Switched to ElPiGraph approach to calculate pseudotime when using that algorithm, leading to more accurate pseudotime measurement.
- Added parameter *epg_extend_leaves* to call `elpigraph.ExtendLeaves()` during graph learning using ElPiGraph.
- Working function for *scFates.tl.test_association_monocle3()* (R file was missing from package).
- Fixed output from *scFates.tl.test_association_covariate()*.
- Allow no legend for `scFates.pl.covariate()`.

1.3.10 Version 0.9.0 August 18, 2022

Major release:

This release has several improvements from 0.8

Major changes:

- As discussed on issue #7, pseudotime calculation has been fixed when using elpigraph. The previous change introduced the issue of cells being assigned the pseudotime of their closest node only. Now the cells are assigned to their closest edge and have a pseudotime value according to their distance between the two nodes composing that edge.
- Added *scFates.tl.explore_sigma()*, a tool for SimplePPT that explore ranges of sigma parameters to avoid the ones which collapse the tree (see the related notebook) for more info).
- New approach to analyses circles, upon removal of edge linked to the root node, the graph is considered as two converging segments toward the furthest node. This allow to perform mulitple mapping without having cells being assigned either the lowest or the furthest pseudotime, leading to wrong assignement when taking the mean

of all mappings. The circle can be further unrolled with *scFates.tl.unroll_circle()* to assign a unique pseudotime value to all cells (for more info see the related notebook).

- added *scFates.tl.test_association_monocle3()*, to test whether features are significantly changing along the tree, using monocle3 approach (requires the package). This can be handy for large dataset where test_association is too slow (does not generate A parameter).
- Reworked *scFates.tl.cluster()*, now uses scanpy and leiden as backend, leading to faster gene module calculations.

1.3.11 Version 0.8.1 July 18, 2022

Minor release: - *pl.milestones_graph* has been removed, simplifying the dependency requirements - `scFates.tl.rename_milestones()` now accepts dictionaries - minor plot fixes

1.3.12 Version 0.8.0 June 29, 2022

This release is stable and ready for journal submission, it is meant to be ready to use and in line with all methods described in the manuscript.

Major changes:

- **breaking change!** pseudotime calculation is now deterministic, which differs from the previous implementation derived from crestree package. In the previous implementation, cells were assigned to a random position between a node and its closest neighbor. Now cells are assigned a pseudotime according to their soft assignment value between between the node and its closest neighbor.
- When calculating pseudotime over several mappings, the mean of all pseudotimes is saved in .obs, instead of taking the first mapping. Cell are assigned to their most assigned segment among all mappings, with corrections for cases were the pseudotime is over or under the limit of the segment.

Other changes:

- *scFates.pl.milestones()* has been converted into a embedding plot which colors the cells as a gradient following milestones. This plot will be called in any other plotting functions which as a coloring of cell paramter set to 'milestones'.
- Added *scFates.tl.convert_to_soft()* to convert ElPiGraph hard assignment R matrix output into a soft one, allowing for probabilistic mapping of cells.
- For plot with embeddings, the basis parameter is now automatically guessed if none is mentionned.
- Improved flexibility and consistency when plotting sub-trajectories
- Default parameters for *scFates.tl.module_inclusion()* have been modified, to focus more on already identified early genes. Inclusion of single gene can now be plotted.

1.3.13 Version 0.4.2 May 16, 2022

Minor release:

- Updated to latest elpigraph version available on pypi, induced slightly changes in principal graph results.
- Added cmap parameter to *scFates.pl.matrix()*, more responsive plotting.
- Fix presence of NAs as repulsion scores in *scFates.pl.slide_cors()*.

1.3.14 Version 0.4.1 March 25, 2022

Minor release focused mainly in plotting improvements:

- Better handling of cases between plot module trends and feature trends for *scFates.pl.single_trend()*.
- Added colorbar and normalization parameter to *scFates.pl.matrix()*.
- Ordering cells according to pseudotime in *scFates.pl.dendrogram()* when coloring by milestone gradients.
- Rasterize segments in *scFates.pl.trajectory()*.
- Fixed auto root selection for *scFates.tl.cellrank_to_tree()*

1.3.15 Version 0.4.0 February 25, 2022

Additions

- *scFates.tl.test_association_covariate()*, to separately test for associated features for each covariates on the same trajectory path.
- *scFates.tl.test_covariate()*, to test for branch differential gene expression between two covariates on the same trajectory path.

Improvements

- *scFates.tl.fit()* can be called for any features.
- *scFates.tl.test_association()* has now spline.df parameter.
- *scFates.pl.graph()* : Segments and nodes are now rasterized in pl.graph for lighter plotting.
- *scFates.pl.matrix()* can now return related dataset.
- *scFates.pl.slide_cors()* : Absolute repulsion score is now shown.

1.3.16 Version 0.3.2 February 12, 2022

Additions

- **:module:`scFates.get`** to easily extract data generated by various analyses. (*scFates.get.fork_stats()*, *scFates.get.modules()*, *scFates.get.slide_cors()*)
- *scFates.tl.simplify()*, subset a tree by cutting of any nodes and cells having a higher pseudotime value than a threshold.
- *scf.settings.set_figure_pubready()* to set publication ready figures (PDF/Arial output, needs Arial installed on the system)

Improvements/Fix

- **_!Affected results!_**: Effect calculation only consider compared cells when rescale=False in *scFates.tl.test_fork()*
- Merged `scFates.tl.limit_pseudotime()` with `scFates.tl.subset()`, can now cutoff before a set pseudotime (*t_min* parameter).
- *scFates.pl.slide_cors()* : Allow to focus on one window and annotate most repuslive genes. Fixed inverted colors for the gene modules when bifuraction analysis was applied.
- Flexibility improvements for *scFates.pl.matrix()*, *scFates.pl.single_trend()*, *scFates.pl.graph()*, *scFates.pl.synchro_path()*, *scFates.pl.modules()*

1.3.17 Version 0.3.1 January 4, 2022

Additions

- *scFates.pl.matrix()* a new and compact way for plotting features over a subset or the whole tree.
- `scFates.tl.limit_pseudotime()`, subset a tree by cutting of any nodes and cells having a higher pseudotime value than a threshold.
- *scf.settings.set_figure_pubready()* to set publication ready figures (PDF/Arial output, needs Arial installed on the system)

Improvements/Fix

- Solved *scFates.tl.dendrogram()* breaking down when version of seaborn is higher than v0.11.1
- *scFates.tl.cluster()*: Output more information.
- Better parallel handling of `tl.test_association()` for multiple mapping.
- Flexibility improvements for *scFates.pl.trends()*, *scFates.pl.single_trend()*, *scFates.pl.synchro_path()*, *scFates.pl.modules()*.

1.3.18 Version 0.3 November 11, 2021

Changes

- **_!Breaking change!_** R soft assignment matrix now is moved to *.obsm* for better flexibility (notably when subsetting). If using an older dataset: refit the tree (with the same parameters) to update to the new data organisation.
- Removal of LOESS for *scFates.tl.synchro_path()* (too slow). Using GAM instead, and only when calling *scFates.pl.synchro_path()*.
- Removal of critical transition related functions.

Improvements

- *scFates.pp.batch_correct()* Faster matrix saving.
- *scFates.tl.circle()*: Allow to use weights for graph fitting with simpleppt.
- *scFates.tl.subset_tree()*: Transfer segment colors to new tree when subsetting.
- *scFates.tl.circle()*: Better parallelism when doing on multiple mappings.
- *scFates.pl.binned_pseudotime_meta()*: More responsive plot.
- Better handling of R dependencies related errors.

1.3.19 Version 0.2.7 September 23, 2021

Additions

- *scFates.tl.circle()*, to fit a principal circle on high dimensions!
- *scFates.tl.dendrogram()* and *pl.dendrogram*, for generating and plotting a dendrogram URD style single-cell embedding for better interpretability
- `scFates.tl.extend_tips()` (replaces *tl.refine_pseudotime*) to avoid the compression of cells at the tips.
- *scFates.pl.binned_pseudotime_meta()*, a dotplot showing the proportion of cells for a given category, along binned pseudotime intervals.

New walkthroughs

- Tree operation walkthrough, for tree subsetting, attachment and extension.
- Basic trajectory walkthrough, for simple developmental transition.
- Going beyond scRNAseq, one can also apply scFates to other dynamical systems, such as neuronal recordings.

Improvements

- *scFates.tl.attach_tree()*: Allow to attach trees without milestones (using vertiex id instead).
- *scFates.tl.subset_tree()*: Better handling of tree subsetting when different root is used. Previosu milestones are saved.
- *scFates.pl.trends()* now respects embedding aspect ratio, can now save figure.

Changes

- any graph fitting functions relying in elpigraph now removes automatically non-assigned nodes, and reattach the separated tree at the level of removals in case the tree is broken into pieces.
- *scFates.pl.milestones()* default layout to dendrogram view (similar to *tl.dendrogram* layout).
- *scFates.tl.subset_tree()* default mode is “extract”.
- *scFates.pl.linearity_deviation()* has a font parameter, with a default value.

1.3.20 Version 0.2.6 August 29, 2021

Additions

- added *scFates.tl.subset_tree()* and *scFates.tl.attach_tree()*, functions that allow to perform linkage or cutting operations on tree or set of two trees.

Improvements

- Added possibility to show any metadata on top of *scFates.pl.trends()*
- *scFates.pl.trajectory()* can now color segments with nice gradients of milestone colors following pseudotime.
- Added check for sparsity in *scFates.pp.find_overdispersed()*, as it is a crucial parameter for finding overdispersed features.
- *scFates.tl.root()* can now automatically select a tip, and with a minimum value instead of a max.
- *scFates.pl.single_trend()* can now plot raw and fitted mean module along pseudotime, plots with embedding can now be saved as image.

1.3.21 Version 0.2.5 July 09, 2021

Addition/Changes

- code for SimplePPT algorithm has been moved to a standalone python package simpelppt.
- *scFates.tl.activation_lm()*, a more robust version of tl.activation, as it uses linear model to identify activation of feature prior to bifurcation.
- *scFates.tl.root()* can now automatically select root from any feature expression.

1.3.22 Version 0.2.4 May 31, 2021

As mentioned in the following issue, this release removes the need to install the following dependencies: Palantir, cellrank and rpy2. This allows for a faster installation of a base scFates package and avoid any possible issues caused by rpy2 and R conflicts.

Modifications/Improvements

- *scFates.pl.modules()*: added *smooth* parameter for knn smoothing of the plotted values.
- *scFates.pl.trajectory()*: better segment and fork coloring, now uses averaging weigthed by the soft assignment matrix R to generate values.

1.3.23 Version 0.2.3 May 17, 2021

Additions

- *scFates.tl.module_inclusion()* and its plotting counterpart, estimate the pseudotime of inclusion of a feature whitin its own module.
- *scFates.tl.linearity_deviation()* and its plotting counterpart, a test to assess whether a given bride could be the result of doublets or not.
- *scFates.tl.synchro_path_multi()*, called with more than two terminal states. This wrapper will call *scFates.tl.synchro_path()* on all pair combination theses endpoints.
- *scFates.tl.root()* can now automatically identify the root node of the tree, by projecting on it differentiation measurments such as CytoTRACE.

Modifications/Improvements

- More precise cell projection of critical transition index values via loess fit.

1.3.24 Version 0.2.2 Apr 27, 2021

Additions for conversion and downstream analysis

- `scFates.tl.critical_transition()`, with its plotting counterpart, calculate the critical transition index along the trajectory.
- `scFates.tl.criticality_drivers()`, identifies genes correlated with the projected critical transition index value on the cells.
- *scFates.pl.test_fork()*, plotting counterpart of *scFates.tl.test_fork()*, for better selection of threshold A.
- *scFates.tl.cellrank_to_tree()*, wrapper that convert results from CellRank analysis into a principal tree that can be subsequently analysed.

Additions for preprocessing

- *scFates.pp.diffusion()*, wrapper that performs Palantir.
- *scFates.pp.filter_cells()* a molecule by genes filter translated from pagoda2 R package.
- *scFates.pp.batch_correct()* a simple batch correction method translated from pagoda2 R package.
- *scFates.pp.find_overdispersed()*, translated from pagoda2 R package.

1.3.25 Version 0.2.0 Feb 25, 2021

Additons

- *scFates.tl.curve()* function, a wrapper of computeElasticPrincipalCurve from ElPiGraph, is now added to fit simple curved trajectories.
- Following this addition and for clarity, plotting functions `scFates.pl.tree()` and `scFates.pl.tree_3d()` have been respectively renamed *scFates.pl.graph()* and *scFates.pl.trajectory_3d()*.

Modifications on `scFates.tl.tree()` when simplePPT is used

- euclidean distance function is replaced by `sklearn.metrics.pairwise_distances()` for cpu and `cuml.metrics.pairwise_distances.pairwise_distances()` for gpu, leading to speedups. Non-euclidean metrics can now be used for distance calculations.
- Several steps of computation are now performed via numba functions, leading to speedups for both cpu and gpu.
- Thanks to rapids 0.17 release, `scipy.sparse.csgraph.minimum_spanning_tree()` is replaced by `cugraph.tree.minimum_spanning_tree.minimum_spanning_tree()` on gpu, providing great speed improvements when learning a graph with very high number of nodes.

`scFates.tl.test_fork()` modifications

- includes now a parameter that rescale the pseudotime length of the two post-bifurcation branches to 1. This allows for comparison between all cells, instead of only keeping cells with a pseudotime up to the maximum pseudotime of the shortest branch. This is useful especially when the two branches present highly different pseudotime length.
- can now perform DE on more than two branches (such in case of trifurcation).

Other modifications on crestree related downstream analysis functions

- tl.activation now uses a distance based (pseudotime) sliding window instead of cells, leading to a more robust identification of activation pseudotime.
- include a fully working `scFates.tl.refine_pseudotime()` function, which applies Palantir separately on each segment of the fitted tree in order to mitigate the compressed pseudotime of cells at the tips.
- *scFates.tl.slide_cors()* can be performed using user defined group of genes, as well as on a single segment of the trajectory.

1.3.26 Version 0.1 Nov 16, 2020

Version with downstream analysis functions closely related to the initial R package crestree. Includes ElPiGraph as an option to infer a principal graph.

1.4 References

1.5 Basic Curved trajectory analysis

The objective of this notebook is to learn how to perform linear (curve) trajectory inference from single cell data, starting from a count matrix. Features that significantly changes along the tree will then be extracted and clustered.

1.5.1 Preparing the environment for the tutorial

The following needs to be run in the command-line

```
conda create -n scFates -c conda-forge -c r python=3.8 r-mgcv rpy2 -y
conda activate scFates
conda install ipykernel
python -m ipykernel install --user --name scFates --display-name "scFates"
pip install scFates loompy
```

1.5.2 Importing modules and basic settings

[1]:
```
import warnings
warnings.simplefilter(action='ignore', category=Warning)
import os, sys
# to avoid any possible jupyter crashes due to rpy2 not finding the R install on conda
os.environ['R_HOME'] = sys.exec_prefix+"/lib/R/"
import scanpy as sc
import scFates as scf

adata = sc.read('hgForebrainGlut.loom',
                backup_url='http://pklab.med.harvard.edu/velocyto/hgForebrainGlut/
↪hgForebrainGlut.loom')
adata.var_names_make_unique()
sc.set_figure_params()
```

[2]:
```
sc.settings.verbosity = 3
sc.settings.logfile = sys.stdout
```

1.5.3 Pre-processing

[3]:
```
sc.pp.filter_genes(adata,min_cells=3)
sc.pp.normalize_total(adata)
sc.pp.log1p(adata,base=10)
sc.pp.highly_variable_genes(adata)
```

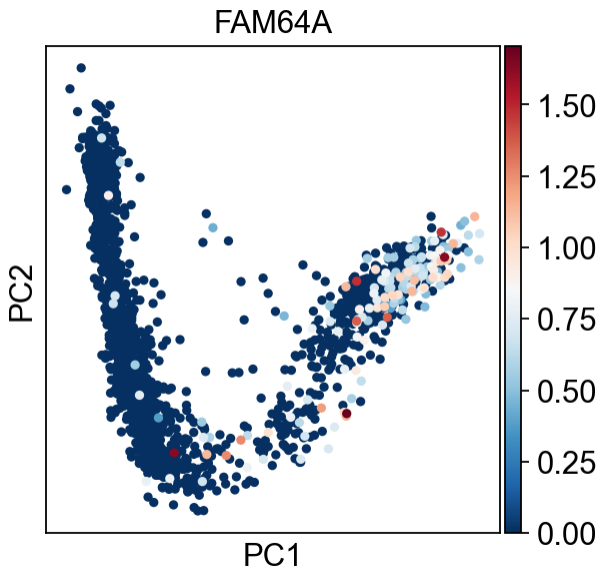
```
filtered out 18081 genes that are detected in less than 3 cells
normalizing counts per cell
    finished (0:00:00)
extracting highly variable genes
    finished (0:00:00)
--> added
    'highly_variable', boolean vector (adata.var)
    'means', float vector (adata.var)
    'dispersions', float vector (adata.var)
    'dispersions_norm', float vector (adata.var)
```

[4]:
```
adata.raw=adata
```

[5]:
```
adata=adata[:,adata.var.highly_variable]
sc.pp.scale(adata)
sc.pp.pca(adata)
```

```
... as `zero_center=True`, sparse input is densified and may lead to large memory␣
↪consumption
computing PCA
    on highly variable genes
    with n_comps=50
    finished (0:00:00)
```

[6]:
```
sc.pl.pca(adata,color="FAM64A",cmap="RdBu_r")
```



1.5.4 Learn curve using ElPiGraph algorithm

We will infer a principal curve on the 2 first PC components. Any dimensionality reduction in `.obsm` can be selected using `use_rep` parameter, and the number of dimension to retain can be set using `ndims_rep`.

[7]:
```
scf.tl.curve(adata,Nodes=30,use_rep="X_pca",ndims_rep=2,)
```

```
inferring a principal curve --> parameters used
    30 principal points, mu = 0.1, lambda = 0.01
    finished (0:00:01) --> added
    .uns['epg'] dictionnary containing inferred elastic curve generated from elpigraph.
    .obsm['X_R'] soft assignment of cells to principal points.
    .uns['graph']['B'] adjacency matrix of the principal points.
    .uns['graph']['F'], coordinates of principal points in representation space.
```

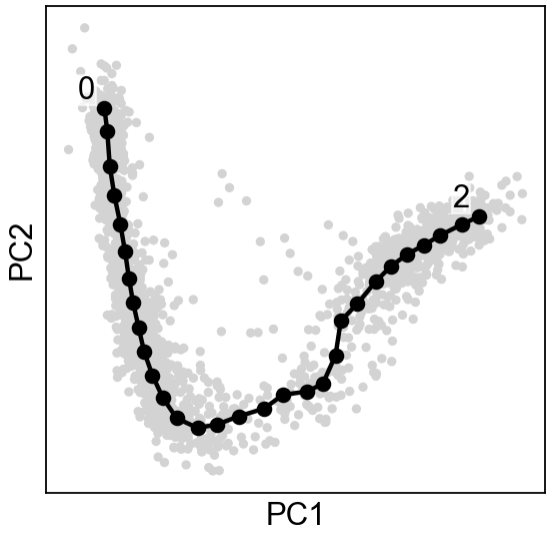
[8]:
```
adata.obsm['X_R']
```

[8]:
```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.72631575, 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.44049647, ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ]])
```

Plotting the tree

By default the plot function will annotate automatically the tips and the forks ids.

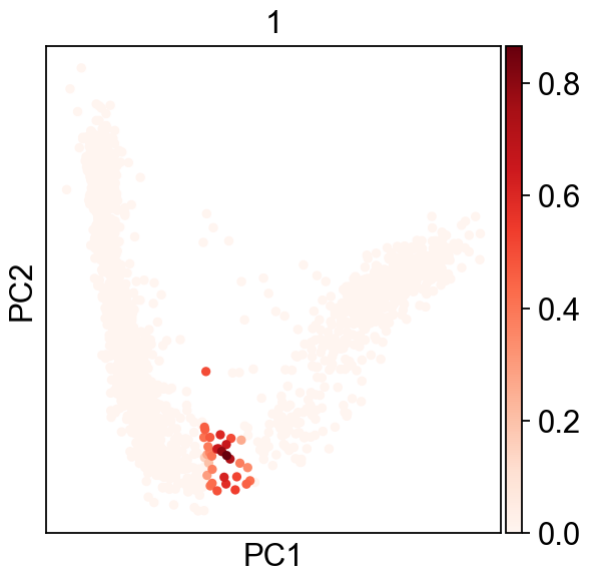
[9]:
```
scf.pl.graph(adata,basis="pca")
```



Showing soft assignment of cells

Now looking at node ID 2 cell assignments, we have continuous values:

[10]:
```
sc.pl.pca(sc.AnnData(adata.obsm["X_R"],obsm=adata.obsm),color="1",cmap="Reds")
```



Our cells are assigned to a node by a value between 0 and 1, which allow us to use probabilistic mappings to account for variability.

1.5.5 Selecting a root and computing pseudotime

Using FAM64A marker, we can confidently tell that the tip 1 is the root.

[11]:
```
scf.tl.root(adata,"FAM64A")
```

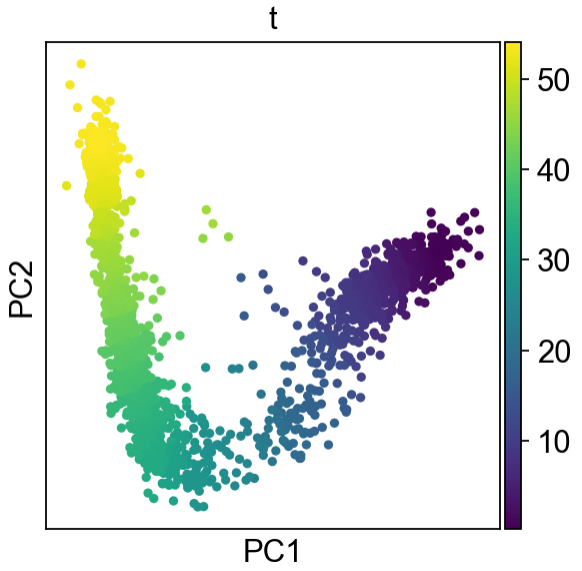
```
automatic root selection using FAM64A values
node 2 selected as a root --> added
    .uns['graph']['root'] selected root.
    .uns['graph']['pp_info'] for each PP, its distance vs root and segment assignment.
    .uns['graph']['pp_seg'] segments network information.
```

Here we are running 100 mappings to account for uncertainty, the pseudotime saved in obs will be the mean of all computed pseudotimes:

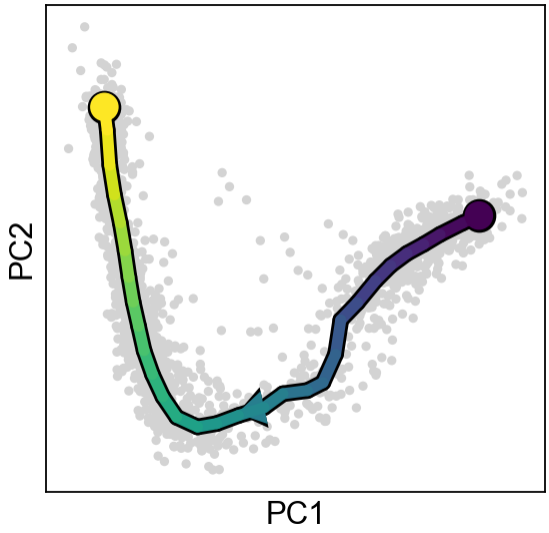
[12]:
```
scf.tl.pseudotime(adata,n_jobs=20,n_map=100,seed=42)
```

```
projecting cells onto the principal graph
    mappings: 100%|| 100/100 [00:20<00:00,  4.95it/s]
    finished (0:00:21) --> added
    .obs['edge'] assigned edge.
    .obs['t'] pseudotime value.
    .obs['seg'] segment of the tree assigned.
    .obs['milestones'] milestone assigned.
    .uns['pseudotime_list'] list of cell projection from all mappings.
```

[13]:
```
sc.pl.pca(adata,color="t")
```



[14]:
```
scf.pl.trajectory(adata,basis="pca",arrows=True,arrow_offset=3)
```

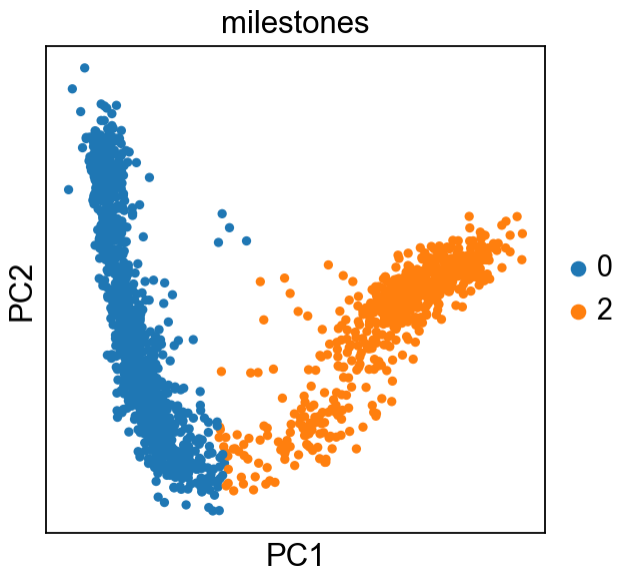


[15]:
```
adata=adata.raw.to_adata()
```

1.5.6 Assign and plot milestones

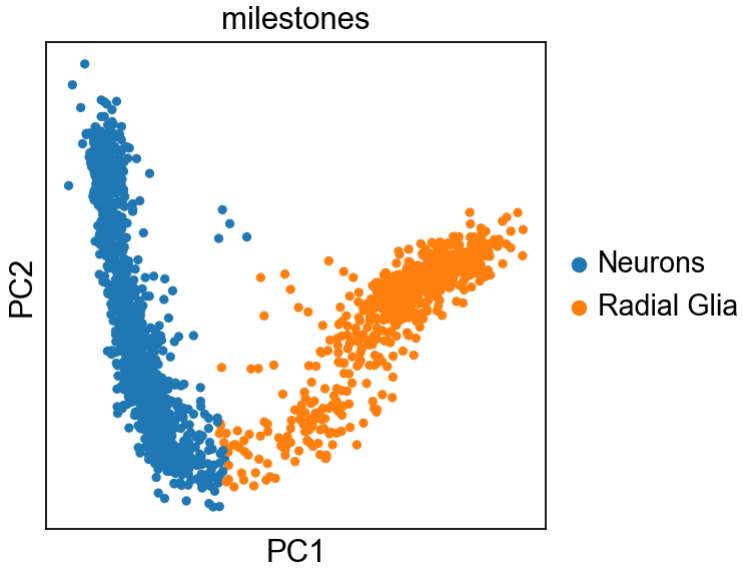
It is easier to keep track of the miletones by naming them with biological concepts (cell-type, state, . . .):

[16]:
```
sc.pl.pca(adata,color="milestones")
```



[17]:
```
scf.tl.rename_milestones(adata,new={"2":"Radial Glia","0": "Neurons"})
```

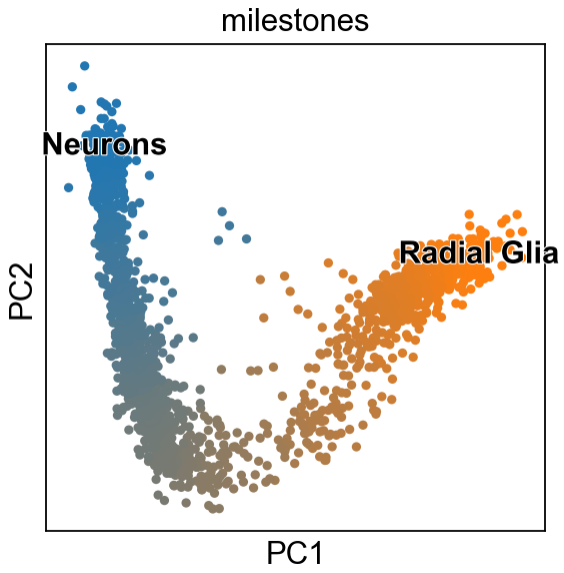
[18]:
```
sc.pl.pca(adata,color="milestones")
```



[19]:
```
adata.uns["graph"]["milestones"]
```

[19]:
```
{'Neurons': 0, 'Radial Glia': 2}
```

[20]:
```
scf.pl.milestones(adata,basis="pca",annotate=True)
```

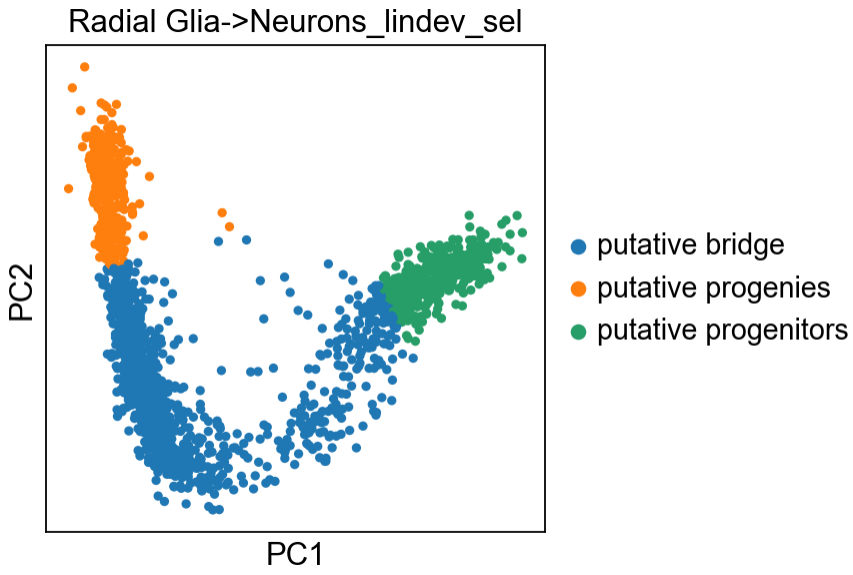


1.5.7 Linearity deviation assessment

In order to verify that the trajectory we are seeing is not the result of a lienar mixture of two population (caused by doublets), we perform the following test:

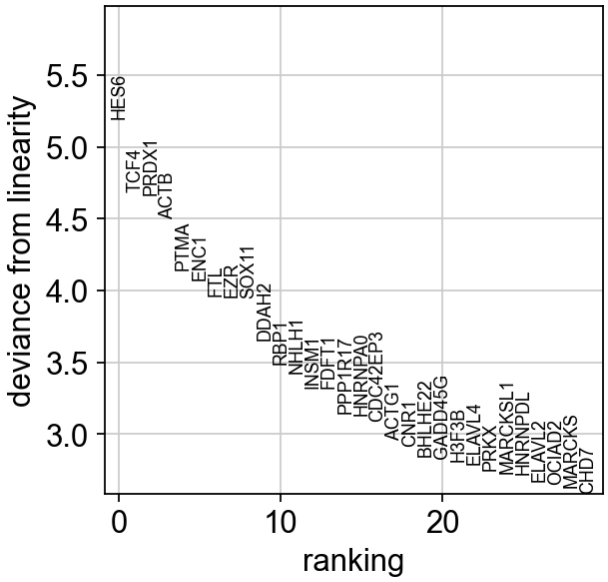
[21]:
```
scf.tl.linearity_deviation(adata,
                           start_milestone="Radial Glia",
                           end_milestone="Neurons",
                           n_jobs=20,plot=True,basis="pca")
```

```
Estimation of deviation from linearity
    cells on the bridge: 100%|| 990/990 [00:03<00:00, 254.71it/s]
```



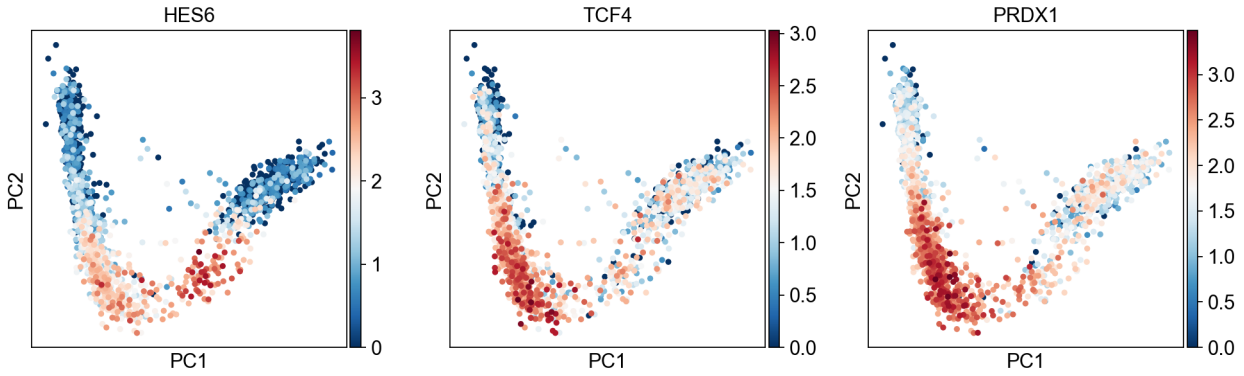
```
    finished (0:00:05) --> added
    .var['Radial Glia->Neurons_rss'], pearson residuals of the linear fit.
    .obs['Radial Glia->Neurons_lindev_sel'], cell selections used for the test.
```

[22]:
```
scf.pl.linearity_deviation(adata,
                           start_milestone="Radial Glia",
                           end_milestone="Neurons")
```



We have markers that highly deviate from linearity, most of which are biologically relevant. We can confidently say that this is a developmental bridge.

[23]:
```
sc.pl.pca(adata,color=["HES6","TCF4","PRDX1"],cmap="RdBu_r")
```



1.5.8 Significantly changing feature along pseudotime test

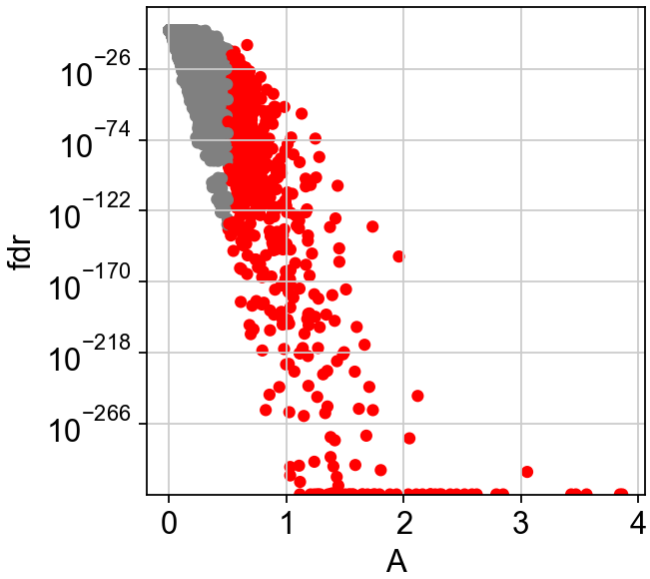
[24]:
```
scf.tl.test_association(adata,n_jobs=20)
```

```
test features for association with the trajectory
    single mapping : 100%|| 14657/14657 [03:39<00:00, 66.90it/s]
    found 147 significant features (0:03:39) --> added
    .var['p_val'] values from statistical test.
    .var['fdr'] corrected values from multiple testing.
    .var['st'] proportion of mapping in which feature is significant.
    .var['A'] amplitue of change of tested feature.
    .var['signi'] feature is significantly changing along pseudotime.
    .uns['stat_assoc_list'] list of fitted features on the graph for all mappings.
```

We can change the amplitude parameter to get more significant genes, this can be done without redoing all the tests (`reapply_filters` parameter)

[25]:
```
scf.tl.test_association(adata,reapply_filters=True,A_cut=.5)
scf.pl.test_association(adata)
```

```
reapplied filters, 635 significant features
```



1.5.9 Fitting & clustering significant features

Warning

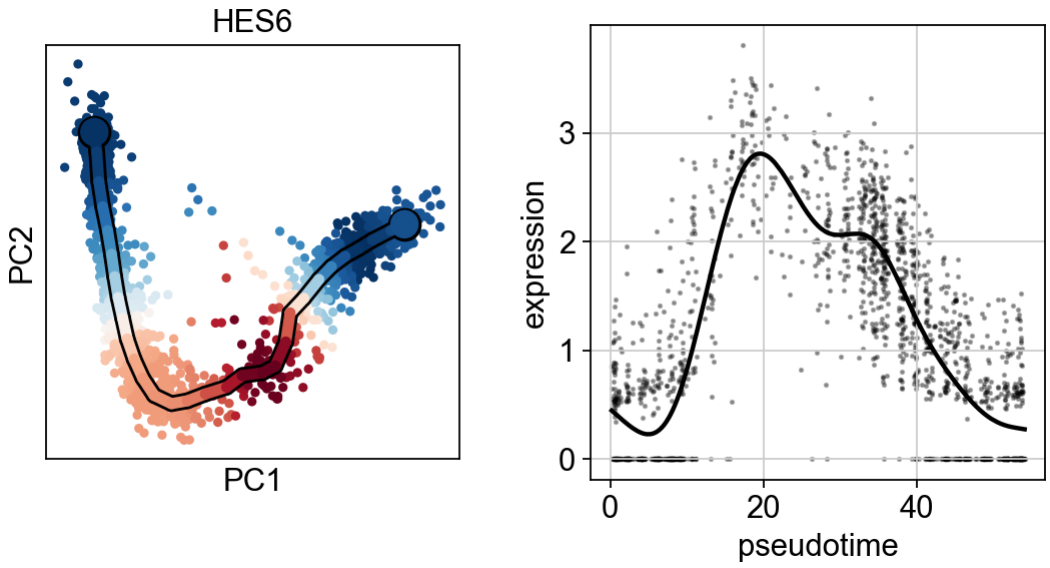
anndata format can currently only keep the same dimensions for each of its layers. This means that adding the layer for fitted features will lead to dataset subsetted to only those!

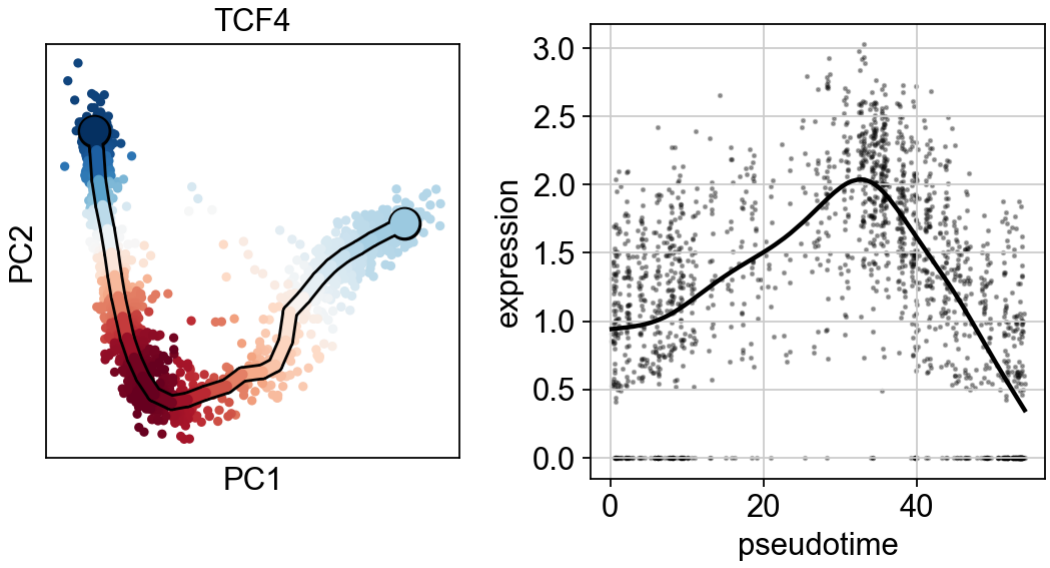
By default the function fit will keep the whole dataset under **adata.raw** (parameter save_raw)

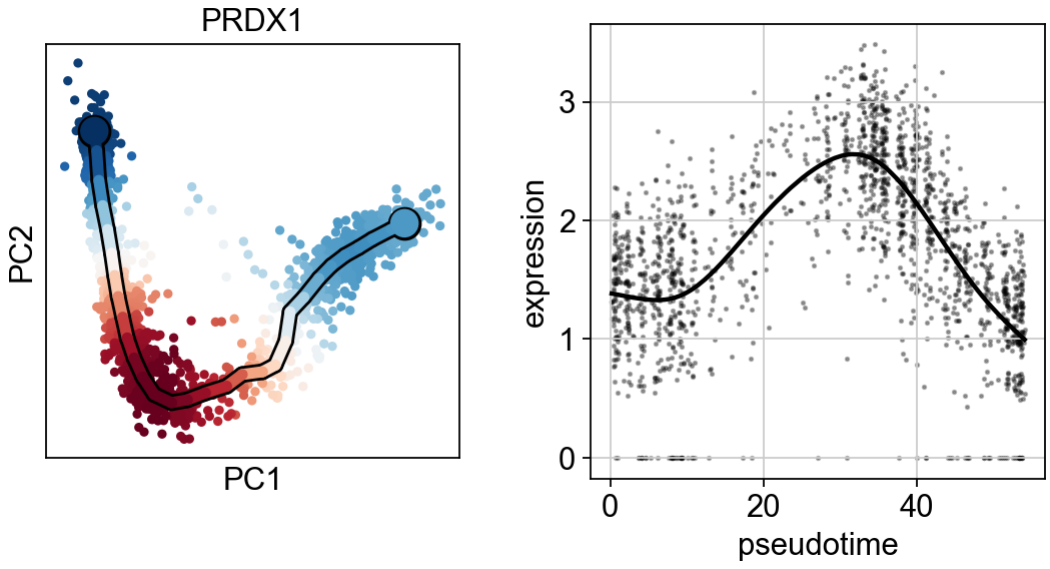
[26]:
```
scf.tl.fit(adata,n_jobs=20)
```

```
fit features associated with the trajectory
    single mapping : 100%|| 635/635 [00:17<00:00, 35.64it/s]
    finished (adata subsetted to keep only fitted features!) (0:00:18) --> added
    .layers['fitted'], fitted features on the trajectory for all mappings.
    .raw, unfiltered data.
```

[27]:
```
scf.pl.single_trend(adata,"HES6",basis="pca",color_exp="k")
scf.pl.single_trend(adata,"TCF4",basis="pca",color_exp="k")
scf.pl.single_trend(adata,"PRDX1",basis="pca",color_exp="k")
```







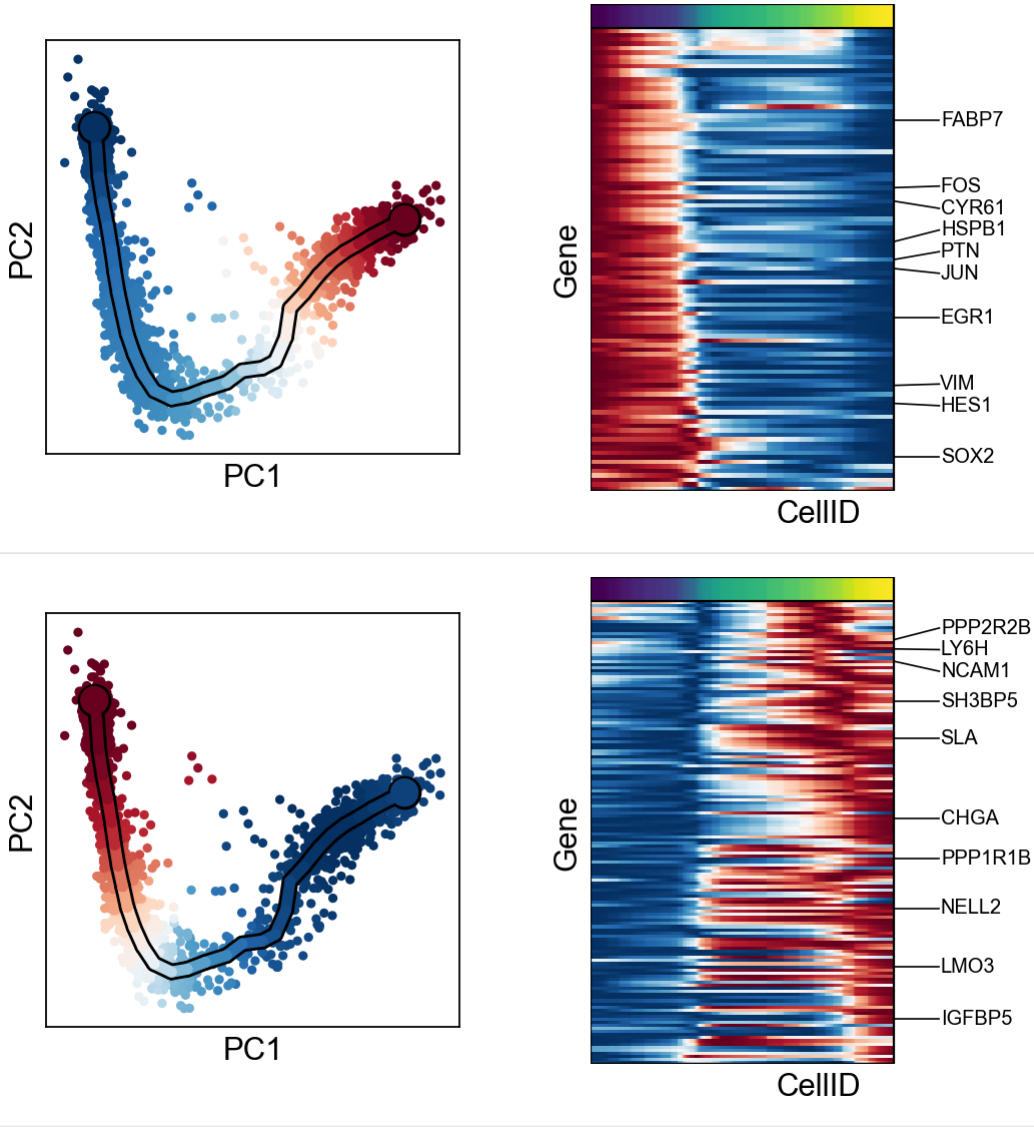
[28]:
```
scf.tl.cluster(adata,n_neighbors=100,metric="correlation")
```

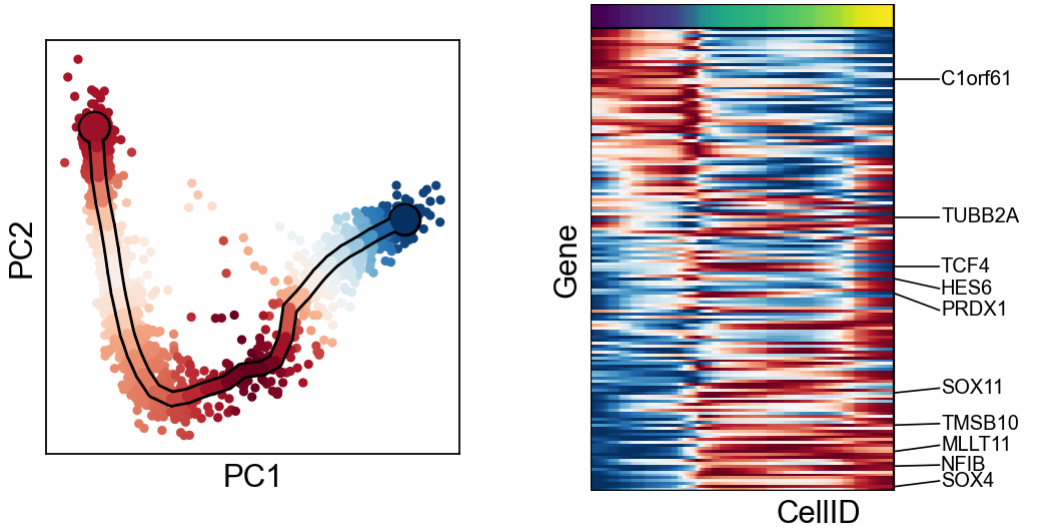
```
Clustering features using fitted layer
computing PCA
    with n_comps=50
    finished (0:00:01)
computing neighbors
    using 'X_pca' with n_pcs = 50
    finished: added to `.uns['neighbors']`
    `.obsp['distances']`, distances for each pair of neighbors
    `.obsp['connectivities']`, weighted adjacency matrix (0:00:11)
running Leiden clustering
    finished: found 6 clusters and added
    'leiden', the cluster labels (adata.obs, categorical) (0:00:01)
    finished (0:00:13) --> added
    .var['clusters'] identified modules.
```

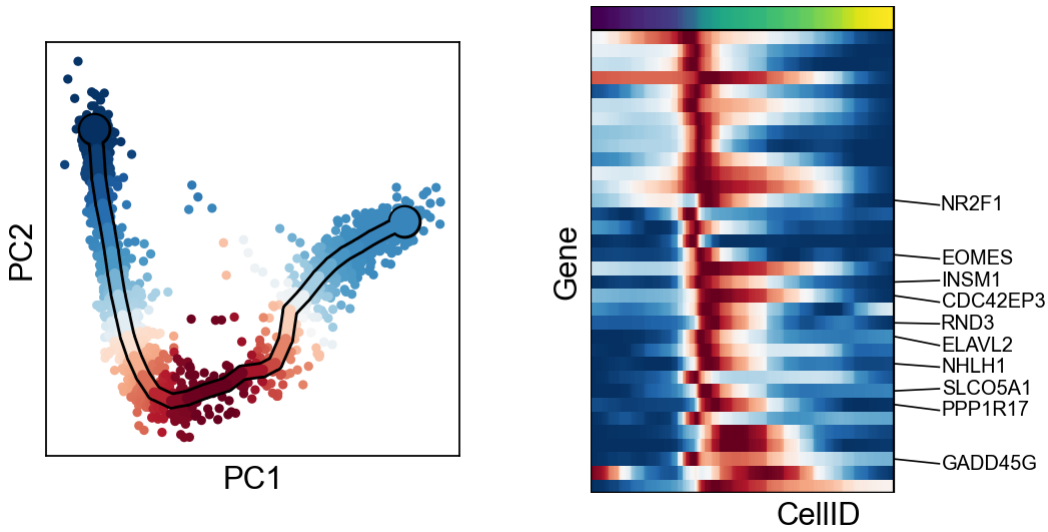
[29]:
```
adata.var.clusters.unique()
```

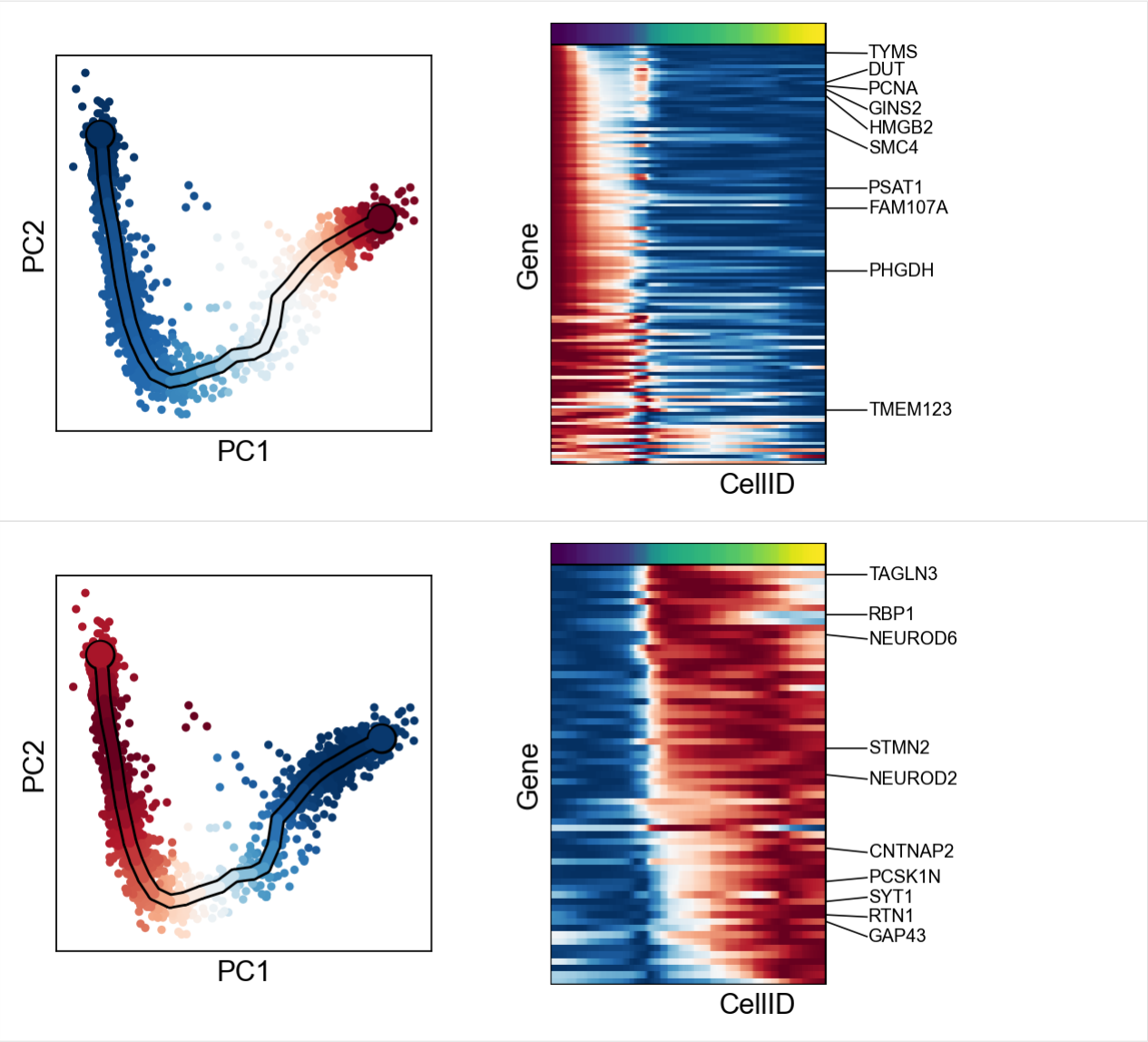
[29]:
```
['3', '1', '0', '5', '2', '4']
Categories (6, object): ['0', '1', '2', '3', '4', '5']
```

[30]:
```
for c in adata.var["clusters"].unique():
    scf.pl.trends(adata,features=adata.var_names[adata.var.clusters==c],basis="pca")
```


PC2
PC1
Gene
CellID
FABP7
FOS
CYR61
HSPB1
PTN
JUN
EGR1
VIM
HES1
SOX2
PC2
PC1
Gene
CellID
PPP2R2B
LY6H
NCAM1
SH3BP5
SLA
CHGA
PPP1R1B
NELL2
LMO3
IGFBP5


PC2
PC1
Gene
CellID
C1orf61
TUBB2A
TCF4
HES6
PRDX1
SOX11
TMSB10
MLLT11
NFIB
SOX4


PC2
PC1
Gene
CellID
NR2F1
EOMES
INSM1
CDC42EP3
RND3
ELAVL2
NHLH1
SLCO5A1
PPP1R17
GADD45G



1.6 Tree analysis - Bone marrow fates

This notebook reproduces the figure 2 from the supplementary method of the package manuscript. It consists in pre-processing human bone marrow dataset from Palantir paper, tree learning, feature testing and fitting, branch DE and bifurcation analysis

1.6.1 Setting up environment modules and basic settings

Generating the environment

The following needs to be run in the command-line

```
conda create -n scFates -c conda-forge -c r python=3.8 r-mgcv rpy2 -y
conda activate scFates

# Install new jupyter server
conda install -c conda-forge jupyter

# Or add to an existing jupyter server
conda install -c conda-forge ipykernel
python -m ipykernel install --user --name scFates --display-name "scFates"

# Install scFates
pip install scFates
```

Required additional packages

[1]:
```
import sys
!{sys.executable} -m pip -q install palantir fa2
```

Loading modules and settings

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import os, sys
# to avoid any possible jupyter crashes due to rpy2 not finding the R install on conda
os.environ['R_HOME'] = sys.exec_prefix+"/lib/R/"
from anndata import AnnData
import numpy as np
import pandas as pd
import scanpy as sc
import scFates as scf
import palantir
import matplotlib.pyplot as plt
sc.settings.verbosity = 3
sc.settings.logfile = sys.stdout

## fix palantir breaking down some plots
import seaborn
seaborn.reset_orig()
%matplotlib inline
sc.set_figure_params()
scf.set_figure_pubready()
```

```
findfont: Font family ['Raleway'] not found. Falling back to DejaVu Sans.
```

1.6.2 Preprocessing pipeline from Palantir

This cell follows the palantir tutorial notebook with some slight changes. Doing palantir diffusion maps is usually a good pre-preprocessing step before using elpigraph or ppt.

Load, normalize and log-transform count data

[3]:
```
counts = palantir.io.from_csv('https://github.com/dpeerlab/Palantir/raw/master/data/
↪marrow_sample_scseq_counts.csv.gz')
norm_df = sc.pp.normalize_per_cell(counts,copy=True)
norm_df = palantir.preprocess.log_transform(norm_df)
```

Perform PCA on highly variable genes

[4]:
```
adata=sc.AnnData(norm_df)
sc.pp.highly_variable_genes(adata, n_top_genes=1500, flavor='cell_ranger')
sc.pp.pca(adata)
pca_projections = pd.DataFrame(adata.obsm["X_pca"],index=adata.obs_names)
```

```
If you pass `n_top_genes`, all cutoffs are ignored.
extracting highly variable genes
    finished (0:00:00)
--> added
    'highly_variable', boolean vector (adata.var)
    'means', float vector (adata.var)
    'dispersions', float vector (adata.var)
    'dispersions_norm', float vector (adata.var)
computing PCA
    on highly variable genes
    with n_comps=50
    finished (0:00:00)
```

Run Palantir to obtain multiscale diffusion space

[5]:
```
dm_res = palantir.utils.run_diffusion_maps(pca_projections)
ms_data = palantir.utils.determine_multiscale_space(dm_res,n_eigs=4)
```

```
Determing nearest neighbor graph...
computing neighbors
    finished: added to `.uns['neighbors']`
    `.obsp['distances']`, distances for each pair of neighbors
    `.obsp['connectivities']`, weighted adjacency matrix (0:00:13)
```

Generate embedding from the multiscale diffusion space

[6]:
```
# generate neighbor draph in multiscale diffusion space
adata.obsm["X_palantir"]=ms_data.values
sc.pp.neighbors(adata,n_neighbors=30,use_rep="X_palantir")
```

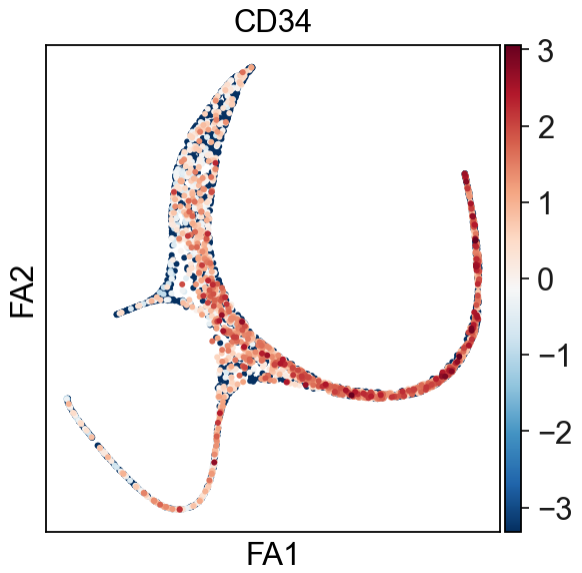
```
computing neighbors
    finished: added to `.uns['neighbors']`
    `.obsp['distances']`, distances for each pair of neighbors
    `.obsp['connectivities']`, weighted adjacency matrix (0:00:01)
```

[7]:
```
# draw ForceAtlas2 embedding using 2 first PCs as initial positions
adata.obsm["X_pca2d"]=adata.obsm["X_pca"][:,:2]
sc.tl.draw_graph(adata,init_pos='X_pca2d')
```

```
drawing single-cell graph using layout 'fa'
    finished: added
    'X_draw_graph_fa', graph_drawing coordinates (adata.obsm) (0:00:29)
```

Plotting results

[8]:
```
sc.pl.draw_graph(adata,color="CD34",color_map="RdBu_r")
```



1.6.3 Tree learning with SimplePPT

[9]:
```
scf.tl.tree(adata,method="ppt",Nodes=200,use_rep="palantir",
            device="cpu",seed=1,ppt_lambda=100,ppt_sigma=0.025,ppt_nsteps=200)
```

```
inferring a principal tree --> parameters used
    200 principal points, sigma = 0.025, lambda = 100, metric = euclidean
    fitting:  71%|   | 142/200 [00:10<00:04, 13.92it/s]
    converged
```

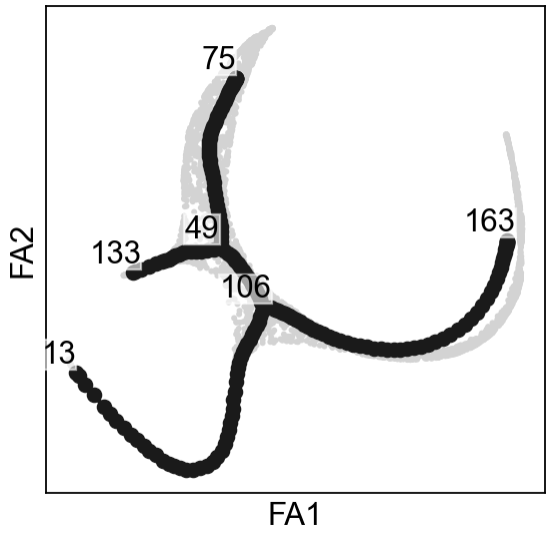
(continues on next page)

(continued from previous page)

```
    finished (0:00:10) --> added
    .uns['ppt'], dictionnary containing inferred tree.
    .obsm['X_R'] soft assignment of cells to principal points.
    .uns['graph']['B'] adjacency matrix of the principal points.
    .uns['graph']['F'] coordinates of principal points in representation space.
```

projecting results onto ForceAtlas2 embedding

[10]:
```
scf.pl.graph(adata)
```



Selecting a root and computing pseudotime

Using CD34 marker, we can confidently tell that the tip 163 is the root.

[11]:
```
scf.tl.root(adata,163)
```

```
node 163 selected as a root --> added
    .uns['graph']['root'] selected root.
    .uns['graph']['pp_info'] for each PP, its distance vs root and segment assignment.
    .uns['graph']['pp_seg'] segments network information.
```

Here we are going to generate 100 mappings of pseudotime, to account for cell asssignment uncertainty. to **.obs** will be saved the mean of all calculated pseudotimes.

[12]:
```
scf.tl.pseudotime(adata,n_jobs=20,n_map=100,seed=42)
```

```
projecting cells onto the principal graph
    mappings: 100%|| 100/100 [00:37<00:00,  2.67it/s]
    finished (0:00:41) --> added
    .obs['edge'] assigned edge.
    .obs['t'] pseudotime value.
    .obs['seg'] segment of the tree assigned.
```

(continues on next page)

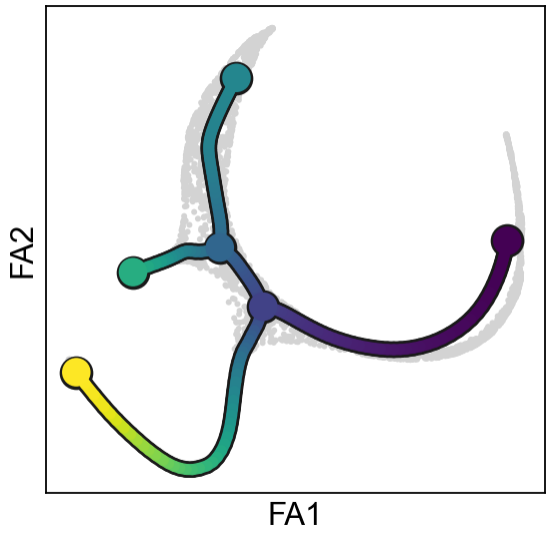
(continued from previous page)

```
    .obs['milestones'] milestone assigned.
    .uns['pseudotime_list'] list of cell projection from all mappings.
```

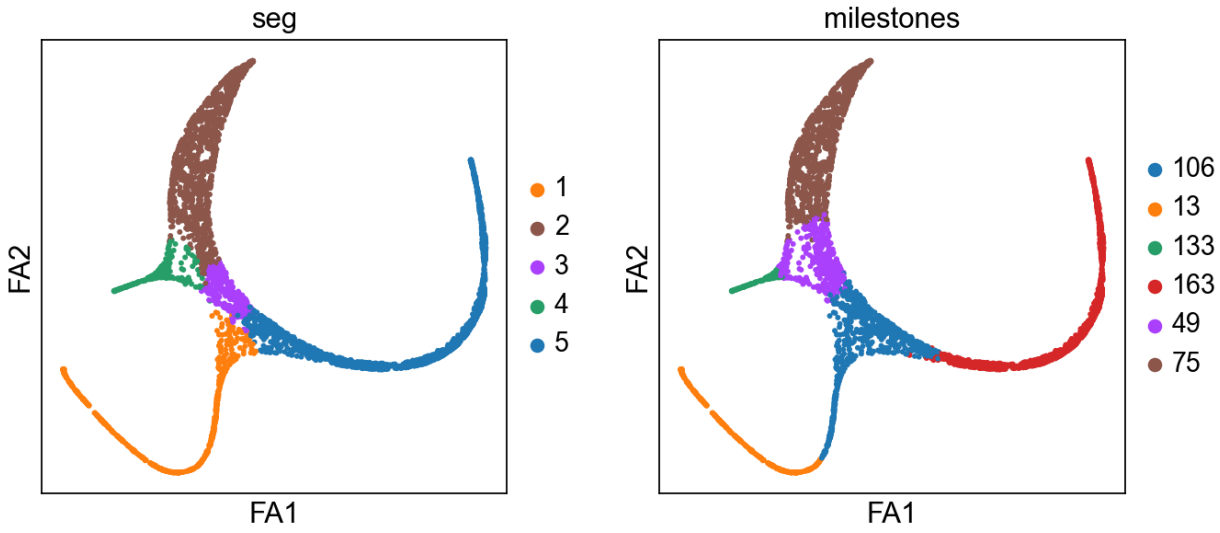
1.6.4 Representing the trajectory and tree

on top of existing embedding

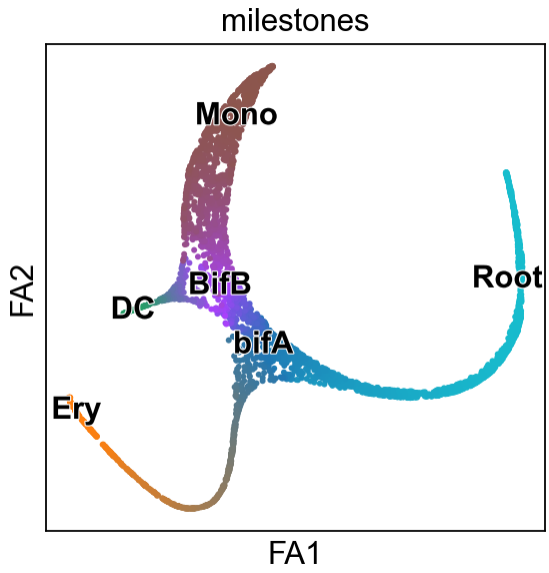
[13]:
```
scf.pl.trajectory(adata)
```



[14]:
```
sc.pl.draw_graph(adata,color=["seg","milestones"])
scf.tl.rename_milestones(adata,["bifA","Ery","DC","Root","BifB","Mono"])
# we change the color of the root milestone for better visualisations
adata.uns["milestones_colors"][3]="#17bece"
```

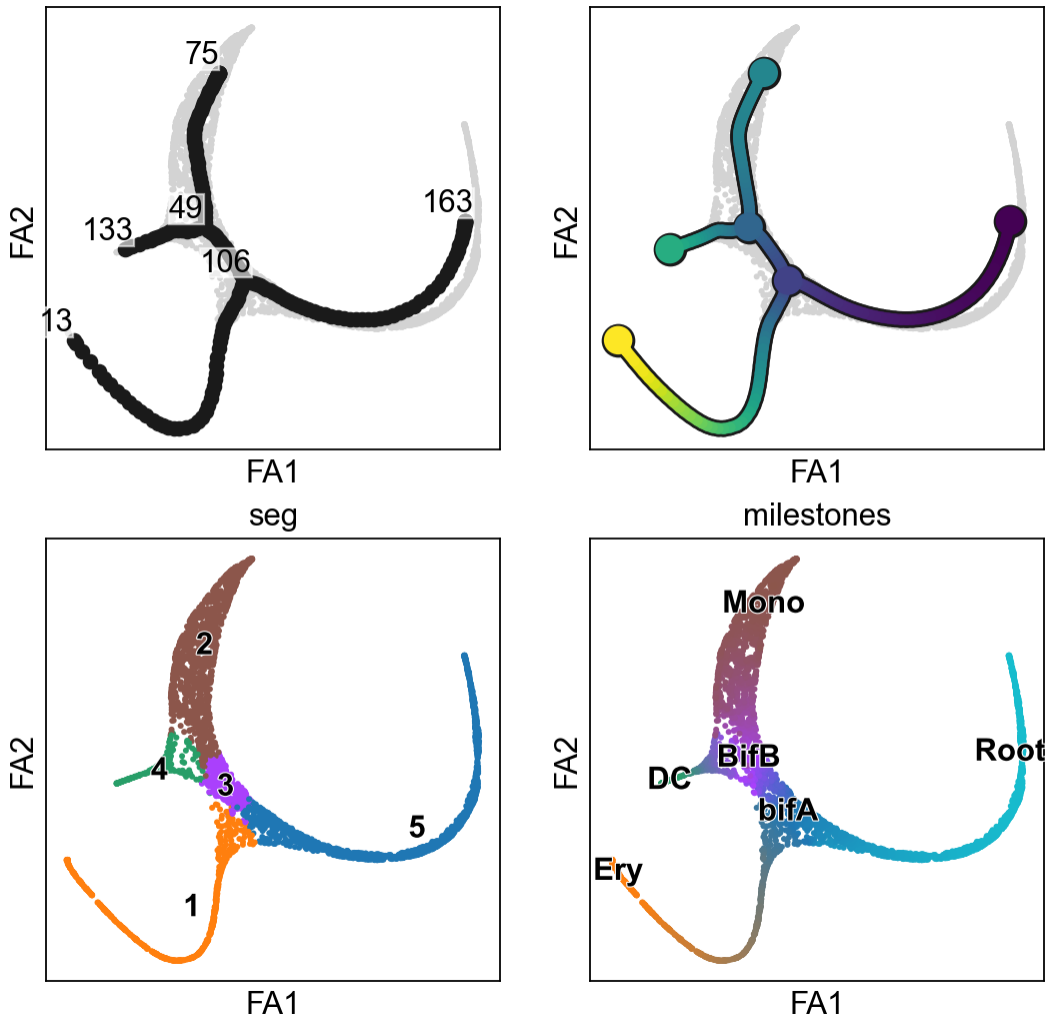


[15]:
```
scf.pl.milestones(adata,annotate=True)
```



[16]:
```
from pathlib import Path
Path("figures/").mkdir(parents=True, exist_ok=True)
```

[17]:
```
sc.set_figure_params()
fig, axs=plt.subplots(2,2,figsize=(8,8))
axs=axs.ravel()
scf.pl.graph(adata,basis="draw_graph_fa",show=False,ax=axs[0])
scf.pl.trajectory(adata,basis="draw_graph_fa",show=False,ax=axs[1])
sc.pl.draw_graph(adata,color=["seg"],legend_loc="on data",show=False,ax=axs[2],legend_
↪fontoutline=True)
scf.pl.milestones(adata,ax=axs[3],show=False,annotate=True)
plt.savefig("figures/A.pdf",dpi=300)
```

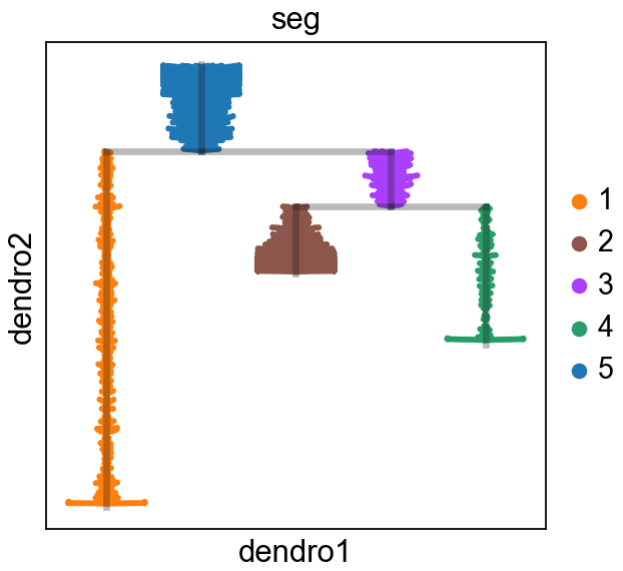


as a dendrogram representation

[18]:
```
scf.tl.dendrogram(adata)
```

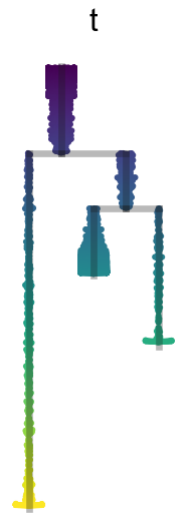
```
Generating dendrogram of tree
    segment : 100%|| 5/5 [00:16<00:00,  3.22s/it]
    finished (0:00:16) --> added
    .obsm['X_dendro'], new embedding generated.
    .uns['dendro_segments'] tree segments used for plotting.
```

[19]:
```
scf.pl.dendrogram(adata,color="seg")
```

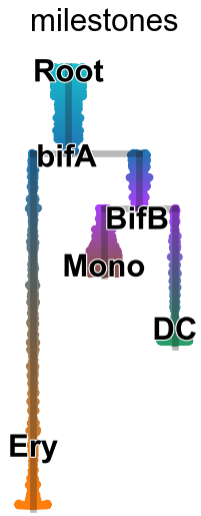


[20]:
```
sc.set_figure_params(figsize=(1.5,4),frameon=False,dpi_save=300)
scf.pl.dendrogram(adata,color="t",show_info=False,save="B1.pdf",cmap="viridis")
scf.pl.dendrogram(adata,color="milestones",legend_loc="on data",color_milestones=True,
↪legend_fontoutline=True,save="B2.pdf")
```

```
WARNING: saving figure to file figures/dendrogramB1.pdf
```




```
WARNING: saving figure to file figures/dendrogramB2.pdf
```



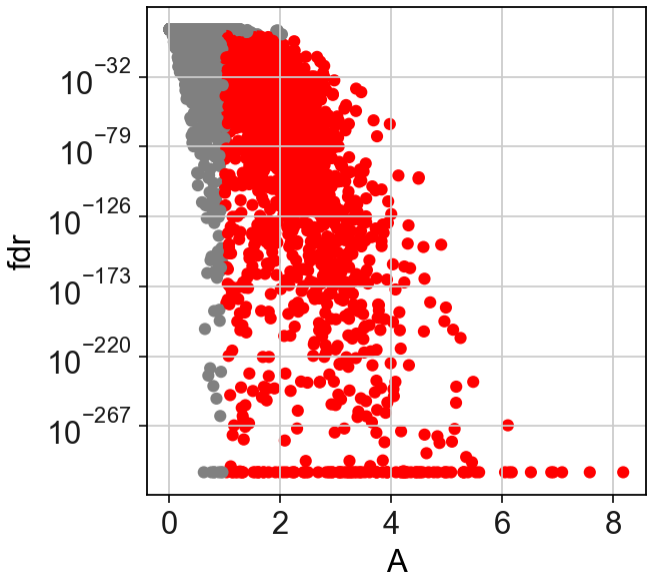
1.6.5 Test and fit features associated with the tree

Let's find out which genes are significantly changing along the tree.

[21]:
```
scf.tl.test_association(adata,n_jobs=20)
```

```
test features for association with the trajectory
    single mapping : 100%|| 16106/16106 [04:45<00:00, 56.38it/s]
    found 3393 significant features (0:04:45) --> added
    .var['p_val'] values from statistical test.
    .var['fdr'] corrected values from multiple testing.
    .var['st'] proportion of mapping in which feature is significant.
    .var['A'] amplitue of change of tested feature.
    .var['signi'] feature is significantly changing along pseudotime.
    .uns['stat_assoc_list'] list of fitted features on the graph for all mappings.
```

[22]:
```
sc.set_figure_params()
scf.pl.test_association(adata)
plt.savefig("figures/C.pdf",dpi=300)
```



We can now fit the significant genes.

Warning

anndata format can currently only keep the same dimensions for each of its layers. This means that adding the layer for fitted features will lead to dataset subsetted to only those!

By default the function fit will keep the whole dataset under **adata.raw** (parameter save_raw set to True by default)

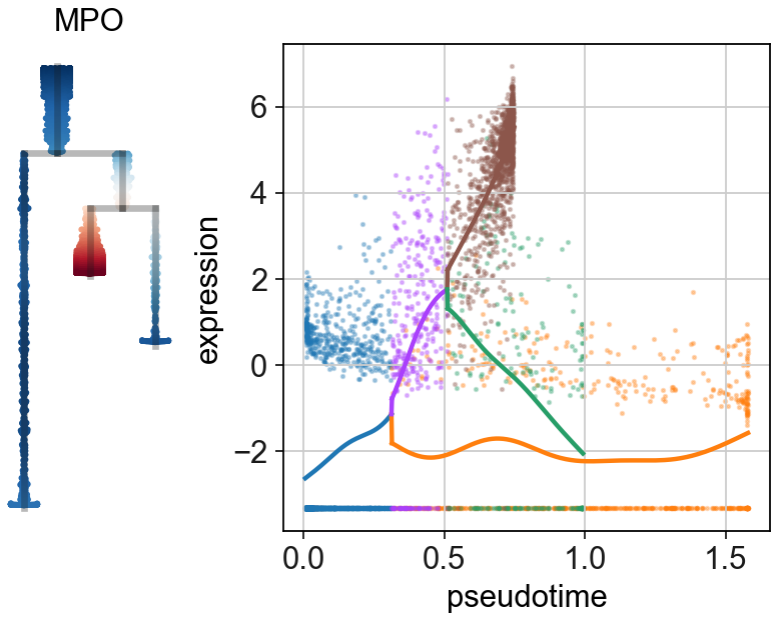
[23]:
```
scf.tl.fit(adata,n_jobs=20)
```

```
fit features associated with the trajectory
    single mapping : 100%|| 3393/3393 [03:13<00:00, 17.51it/s]
    finished (adata subsetted to keep only fitted features!) (0:03:19) --> added
    .layers['fitted'], fitted features on the trajectory for all mappings.
    .raw, unfiltered data.
```

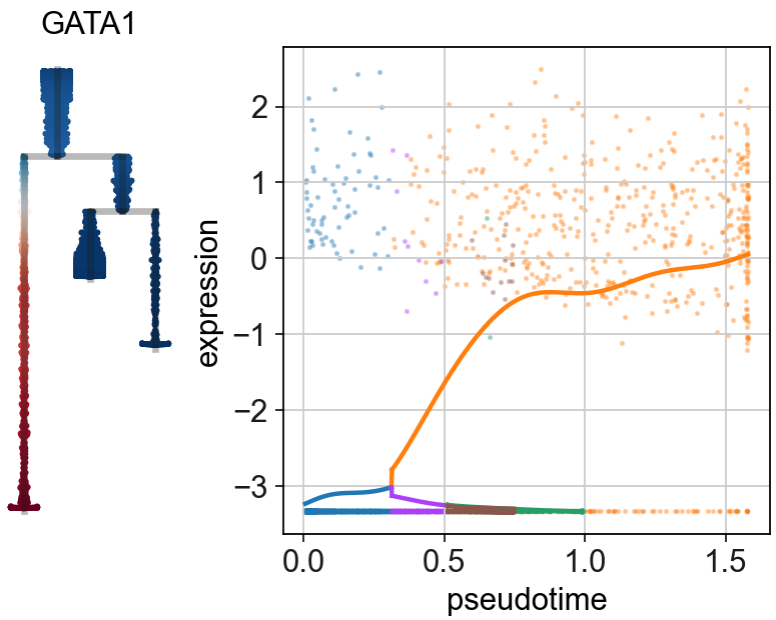
1.6.6 Plotting single features

[24]:
```
sc.set_figure_params(figsize=(.8,4),frameon=False)
scf.set_figure_pubready()
scf.pl.single_trend(adata,"MPO",basis="dendro",wspace=-.25,save="_E1.pdf")
scf.pl.single_trend(adata,"GATA1",basis="dendro",wspace=-.25,save="_E2.pdf")
```

```
WARNING: saving figure to file figures/single_trend_E1.pdf
```




```
WARNING: saving figure to file figures/single_trend_E2.pdf
```



1.6.7 All branches DE analysis

We are here testing for differential expression between all branches, the `rescale` parameter is used to considered all cells in each branch, regardless of their pseudotime:

[25]:
```
scf.tl.test_fork(adata,root_milestone="Root",milestones=["DC","Mono","Ery"],n_jobs=20,
↪rescale=True)
```

```
testing fork
    single mapping
    Differential expression: 100%|| 3393/3393 [01:30<00:00, 37.64it/s]
    test for upregulation for each leave vs root
    upreg DC: 100%|| 1590/1590 [00:01<00:00, 907.08it/s]
    upreg Mono: 100%|| 254/254 [00:00<00:00, 490.62it/s]
    upreg Ery: 100%|| 1549/1549 [00:01<00:00, 991.88it/s]
    finished (0:01:37) --> added
    .uns['Root->DC<>Mono<>Ery']['fork'], DataFrame with fork test results.
```

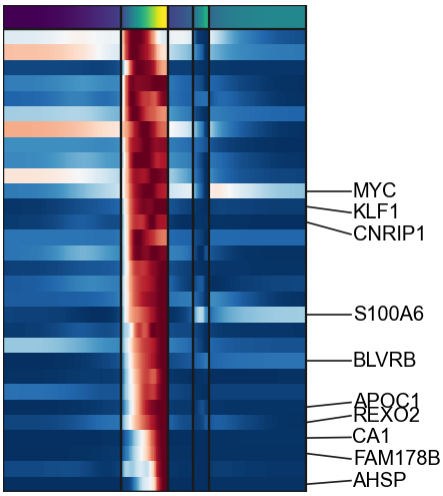
We can keep the features which display the strongest amplitude difference:

[26]:
```
scf.tl.branch_specific(adata,root_milestone="Root",milestones=["DC","Mono","Ery"],
↪effect=2)
```

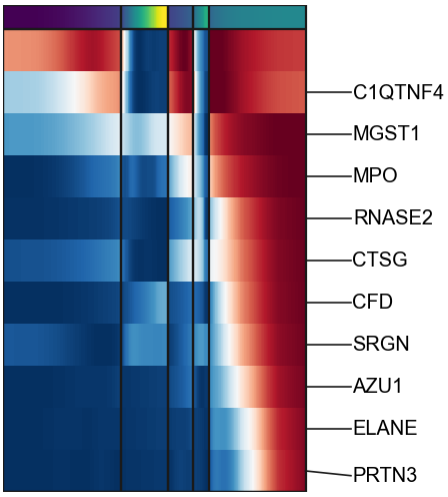
```
    branch specific features: Ery: 30, DC: 15, Mono: 11
    finished --> updated
    .uns['Root->DC<>Mono<>Ery']['fork'], DataFrame updated with additionnal 'branch'␣
↪column.
```

Displaying results using heatmap plots

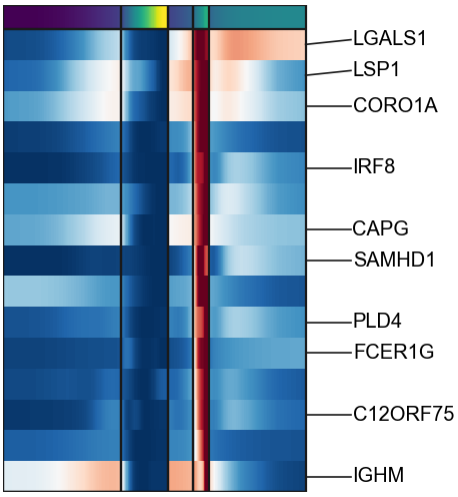
[27]:
```
g1=scf.pl.trends(adata,
                 root_milestone="Root",
                 milestones=["DC","Mono","Ery"],
                 branch="Ery",
                 plot_emb=False,ordering="max",return_genes=True)
```



[28]:
```
g2=scf.pl.trends(adata,
                 root_milestone="Root",
                 milestones=["DC","Mono","Ery"],
                 branch="Mono",
                 plot_emb=False,ordering="max",return_genes=True)
```



[29]:
```
g3=scf.pl.trends(adata,
                 root_milestone="Root",
                 milestones=["DC","Mono","Ery"],
                 branch="DC",
                 plot_emb=False,ordering="max",return_genes=True)
```



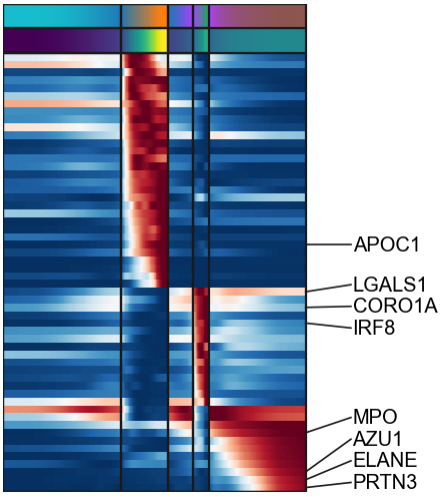
[30]:
```
gg=g1.tolist()+g3.tolist()+g2.tolist()
```

[31]:
```
import matplotlib.pyplot as plt
g=scf.pl.trends(adata,gg,figsize=(4,4),annot="milestones",n_features=8,
```

(continues on next page)

(continued from previous page)

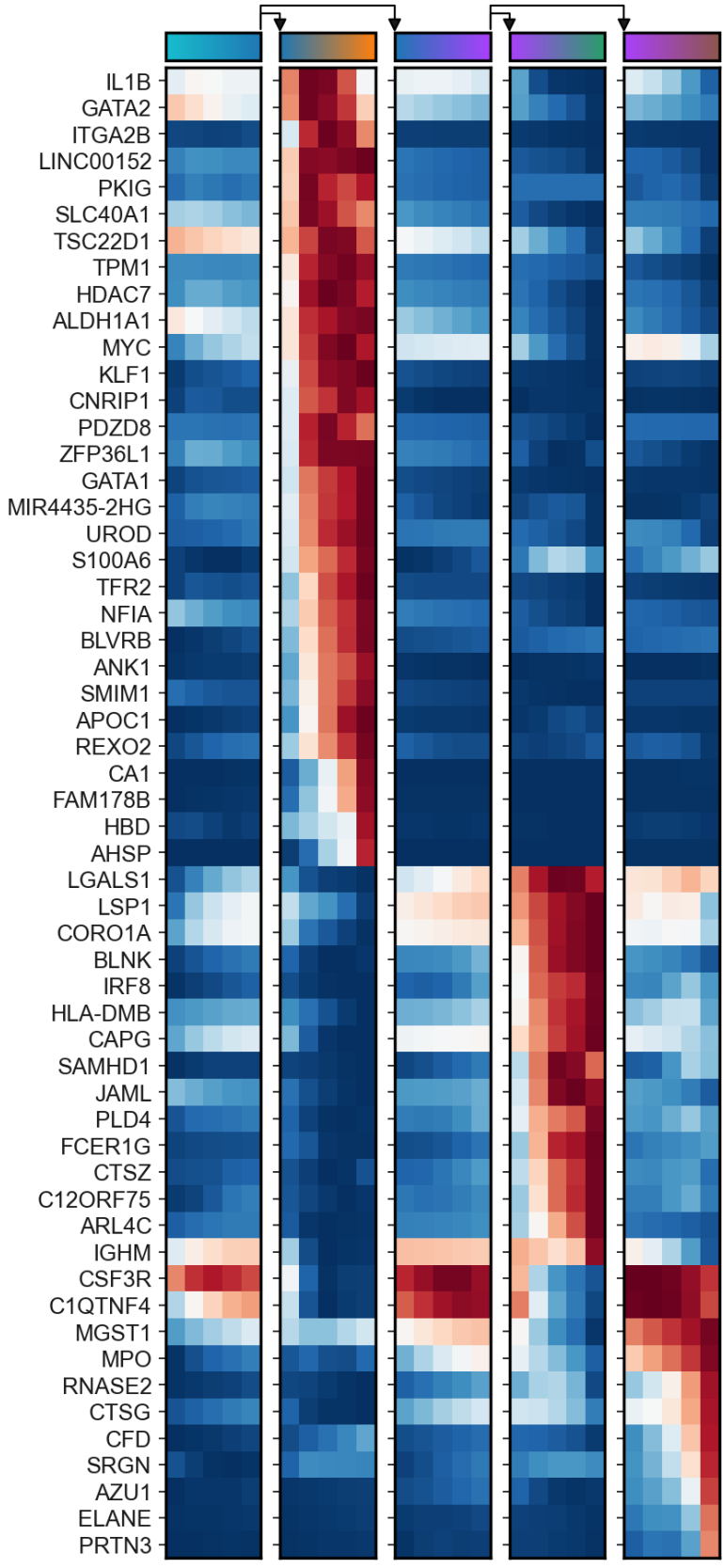
```
        plot_emb=False,ordering=None,return_genes=True)
plt.savefig("figures/D.pdf",dpi=300)
```



Displaying results using matrix plot

[32]:
```
sc.set_figure_params()
scf.pl.matrix(adata,gg,norm="minmax",cmap="RdBu_r",colorbar=False,save="_F.pdf")
```

```
WARNING: saving figure to file figures/matrix_F.pdf
```


IL1B
GATA2
ITGA2B
LINC00152
PKIG
SLC40A1
TSC22D1
TPM1
HDAC7
ALDH1A1
MYC
KLF1
CNRIP1
PDZD8
ZFP36L1
GATA1
MIR4435-2HG
UROD
S100A6
TFR2
NFIA
BLVRB
ANK1
SMIM1
APOC1
REXO2
CA1
FAM178B
HBD
AHSP
LGALS1
LSP1
CORO1A
BLNK
IRF8
HLA-DMB
CAPG
SAMHD1
JAML
PLD4
FCER1G
CTSZ
C12ORF75
ARL4C
IGHM
CSF3R
C1QTNF4
MGST1
MPO
RNASE2
CTSG
CFD
SRGN
AZU1
ELANE
PRTN3

1.6.8 Bifurcation analysis

Let's now focus on a specific bifurcation, where we can apply more advanced functions to detect early biasing

[33]:
```
scf.tl.test_fork(adata,root_milestone="Root",milestones=["DC","Mono"],n_jobs=20,
↪rescale=True)
```

```
testing fork
    single mapping
    Differential expression: 100%|| 3393/3393 [00:46<00:00, 72.93it/s]
    test for upregulation for each leave vs root
    upreg DC: 100%|| 2601/2601 [00:01<00:00, 1555.65it/s]
    upreg Mono: 100%|| 792/792 [00:01<00:00, 683.01it/s]
    finished (0:00:51) --> added
    .uns['Root->DC<>Mono']['fork'], DataFrame with fork test results.
```

[34]:
```
scf.tl.branch_specific(adata,root_milestone="Root",milestones=["DC","Mono"],effect=1.7)
```

```
    branch specific features: DC: 63, Mono: 23
    finished --> updated
    .uns['Root->DC<>Mono']['fork'], DataFrame updated with additionnal 'branch' column.
```

Early gene detection

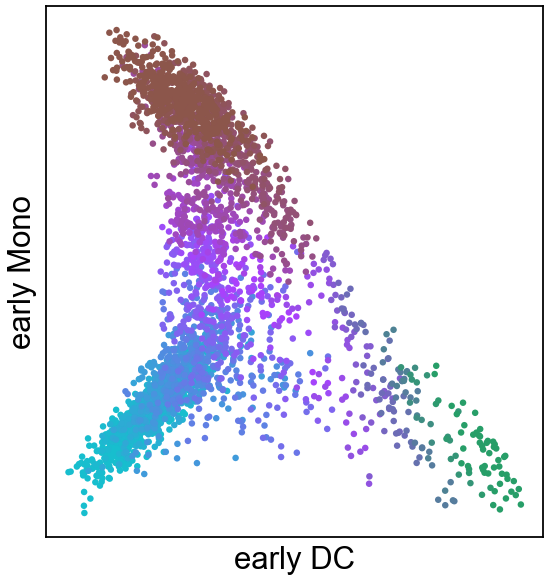
Here we use the linear model approach to detect early genes. We test $g_i \sim pseudotime$ in the progenitor branch only to estimate if the feature displays an upward trend before the fork

[35]:
```
scf.tl.activation_lm(adata,root_milestone="Root",milestones=["DC","Mono"],n_jobs=20)
```

```
    single mapping
    prefork activation: 100%|| 86/86 [00:00<00:00, 463.55it/s]
    40 early and 23 late features specific to leave DC
    16 early and 7 late features specific to leave Mono
    finished (0:00:00) --> updated
    .uns['Root->DC<>Mono']['fork'], DataFrame updated with additionnal 'slope','pval',
↪'fdr','prefork_signi' and 'module' columns.
```

[36]:
```
scf.pl.modules(adata,root_milestone="Root",milestones=["DC","Mono"],
               smooth=True,module="early",save="_G.pdf")
```

```
WARNING: saving figure to file figures/modules_G.pdf
```



Repulsion of early gene modules

For that we need to create non-interesecting windows of cells along the tree:

[37]:
```
scf.tl.slide_cells(adata,root_milestone="Root",milestones=["DC","Mono"],win=400)
```

```
--> added
    .uns['Root->DC<>Mono']['cell_freq'], probability assignment of cells on 9 non␣
↪intersecting windows.
```

In each of the windows we obtain gene-gene correlation of both branch specific ealry modules.

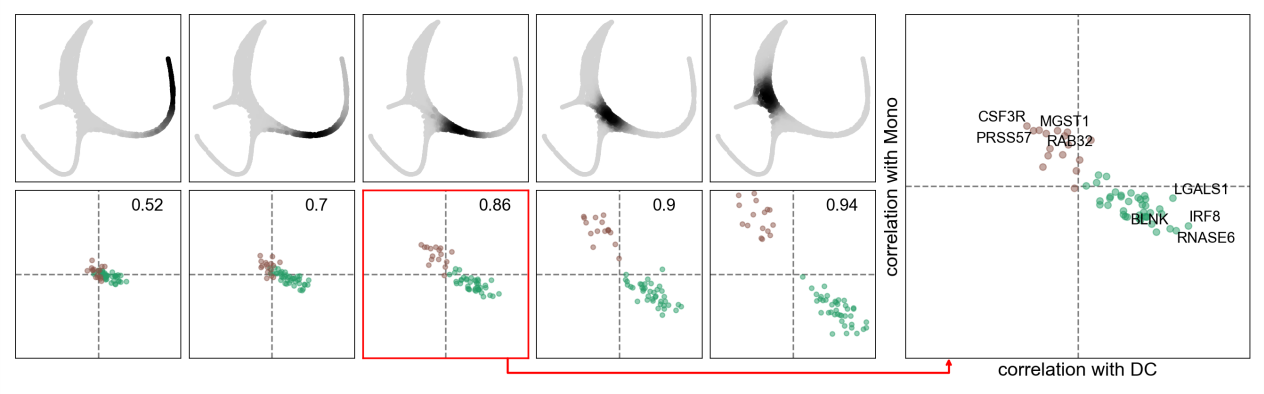
[38]:
```
scf.tl.slide_cors(adata,root_milestone="Root",milestones=["DC","Mono"])
```

```
--> added
    .uns['Root->DC<>Mono']['corAB'], gene-gene correlation modules.
```

Let's plot the results:

[39]:
```
sc.set_figure_params()
scf.pl.slide_cors(adata,root_milestone="Root",milestones=["DC","Mono"],basis="draw_graph_
↪fa",win_keep=[0,2,3,4,5],
                  focus=2,save="_H.pdf")
```

```
WARNING: saving figure to file figures/slide_cors_H.pdf
```



We can see a repulsion and mutual negative correlation prior to the bifurcation, indicating a possible competition of gene programs prior to the bifurcation.

Local trend of module correlations

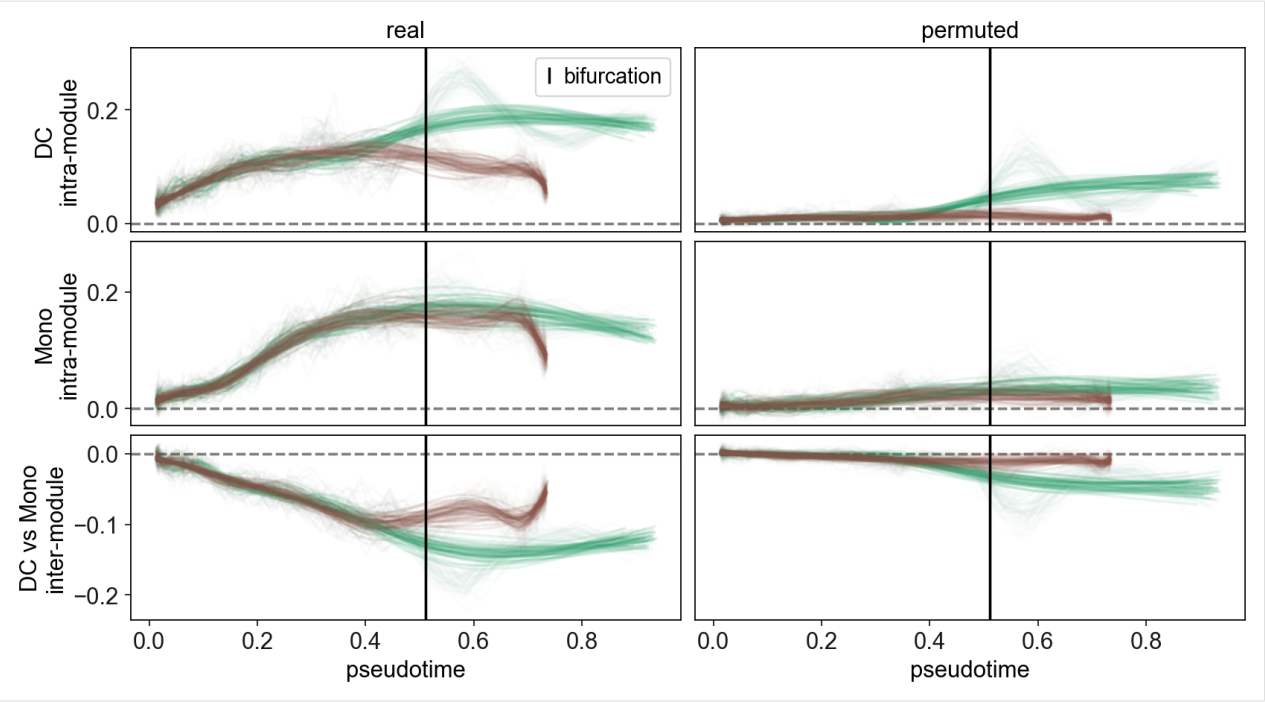
In order to investigate more precisely the competition of programs prior to bifurcation, we can compute local correlation on a sliding intersecting windows of cells:

[40]:
```
scf.tl.synchro_path(adata,root_milestone="Root",milestones=["DC","Mono"],w=100,n_map=50,
↪n_jobs=20)
```

```
computing local correlations
    multi mapping: 100%|| 50/50 [00:16<00:00,  3.05it/s]
    multi mapping permutations: 100%|| 50/50 [00:22<00:00,  2.27it/s]
    finished (0:00:38) --> added
    .uns['Root->DC<>Mono']['synchro'], mean local gene-gene correlations of all possible␣
↪gene pairs inside one module, or between the two modules.
    .obs['inter_cor Root->DC<>Mono'], GAM fit of inter-module mean local gene-gene␣
↪correlations prior to bifurcation.
```

[41]:
```
scf.pl.synchro_path(adata,root_milestone="Root",milestones=["DC","Mono"],save="_I.pdf")
```

```
WARNING: saving figure to file figures/synchro_path_I.pdf
```



Here we see that negative local correlation between the two modules is present at very early pseudotime, even from the start of the trajectory! This indicates that cells are already undergoing biasing before reaching the fork.

Formation of fate-specific modules

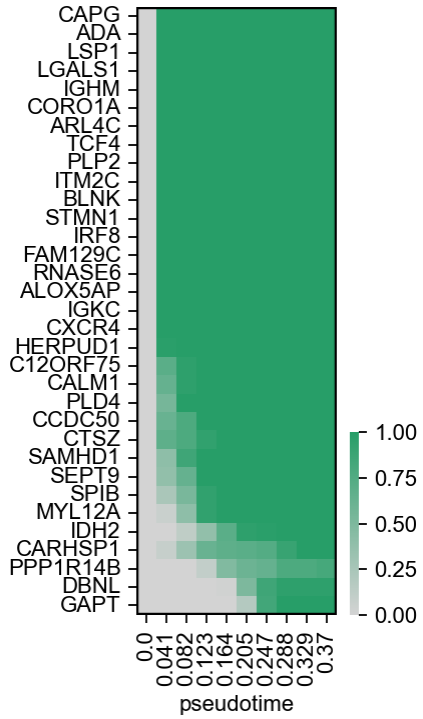
In order to study decision-making process prior to the tree-reconstructed bifurcation point, a framework is provided to identify the timing of gene inclusion into its module

[42]:
```
scf.tl.module_inclusion(adata,root_milestone="Root",milestones=["DC","Mono"],n_jobs=20,n_
↪map=50,parallel_mode="mappings")
```

```
Calculating onset of features within their own module
    multi mapping: 100%|| 50/50 [01:15<00:00,  1.50s/it]
    finished (0:01:15) --> added
    .uns['Root->DC<>Mono']['module_inclusion'], milestone specific dataframes containing␣
↪inclusion timing for each gene in each probabilistic cells projection.
    .uns['Root->DC<>Mono']['fork'] has been updated with the column 'inclusion'.
```

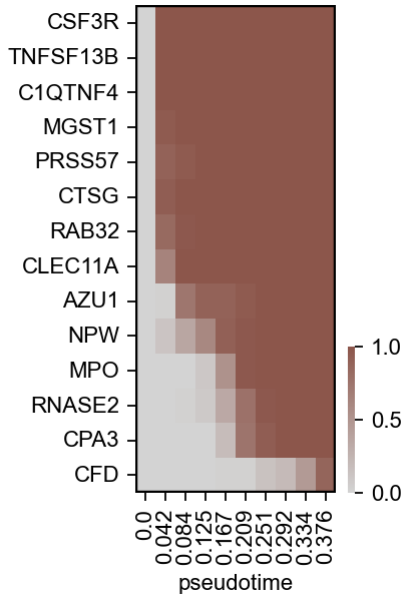
[43]:
```
sc.set_figure_params(fontsize=10)
scf.pl.module_inclusion(adata,root_milestone="Root",milestones=["DC","Mono"],
                        bins=10,branch="DC",save="_J1.pdf",figsize=(2,5))
```

```
WARNING: saving figure to file figures/module_inclusion_J1.pdf
```



[44]:
```
scf.pl.module_inclusion(adata,root_milestone="Root",milestones=["DC","Mono"],bins=10,
↪branch="Mono",
                        save="_J2.pdf",figsize=(2,4))
```

```
WARNING: saving figure to file figures/module_inclusion_J2.pdf
```



1.6.9 Generating the figure

In order to generate the figure, we need to install latex compiler, tectonic, with Arial fonts (additionally, ImageMagick to convert PDF into image):

```
conda install -c conda-forge mscorefonts imagemagick tectonic
```

[45]:
```
fname="fig2_supplementary"
path="/".join(np.array(sys.executable.split("/"))[:-1])
```

[46]:
```
%%bash -s $fname $path
cat<<EOF >$1.tex
\documentclass{article}
\usepackage[paperheight=270mm,paperwidth=210mm]{geometry}
\geometry{left=5mm,right=5mm,top=5mm,bottom=5mm,}

\usepackage{silence}
\WarningsOff*

\usepackage[labelfont=bf]{caption}

\usepackage[rgb]{xcolor}
\usepackage{fontspec}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{graphicx}
\usepackage[export]{adjustbox}

\begin{document}
\setmainfont{Arial}

\noindent
\large

\fontsize{11pt}{11pt}\selectfont

\raggedright \begin{minipage}[!ht]{0.67\textwidth}
\raggedright \begin{minipage}[t]{0.6\textwidth}
\vspace{0cm}
\textbf{A}\\

\includegraphics[width=\textwidth]{figures/A.pdf}
\end{minipage}\hfill
\raggedright \begin{minipage}[t]{0.11\textwidth}
\vspace{0cm}
\textbf{B}\\
\includegraphics[width=\textwidth]{figures/dendrogramB1.pdf}
\includegraphics[width=\textwidth]{figures/dendrogramB2.pdf}
\end{minipage}
\raggedright \begin{minipage}[t]{0.27\textwidth}
\vspace{0cm}
\textbf{C}\\
\includegraphics[width=\textwidth]{figures/C.pdf}
```

(continues on next page)

(continued from previous page)

```
\textbf{D}\\
\includegraphics[width=\textwidth]{figures/D.pdf}
\end{minipage}

\textbf{E}\\

\raggedright \begin{minipage}[!ht]{0.5\textwidth}
\includegraphics[width=\textwidth]{figures/single_trend_E1.pdf}
\end{minipage}\hfill
\raggedright \begin{minipage}[!ht]{0.5\textwidth}
\includegraphics[width=\textwidth]{figures/single_trend_E2.pdf}
\end{minipage}\hfill
\end{minipage}\hfill
\raggedright \begin{minipage}[!ht]{0.315\textwidth}
\vspace{0cm}
\begin{minipage}[t]{\textwidth}
\vspace{0cm}
\textbf{F}\\

\includegraphics[width=\textwidth]{figures/matrix_F.pdf}
\end{minipage}\hfill
\end{minipage}
\hfill


\raggedright \begin{minipage}[t]{0.22\textwidth}
\raggedright \textbf{G}\\
\includegraphics[width=\textwidth]{figures/modules_G.pdf}
\end{minipage}\hfill
\raggedright \begin{minipage}[t]{0.78\textwidth}
\raggedright \textbf{H}\\
\includegraphics[width=\textwidth]{figures/slide_cors_H.pdf}
\end{minipage}\hfill

\raggedright \begin{minipage}[t]{0.59\textwidth}
\vspace{0cm}
\textbf{I}\\

\includegraphics[width=\textwidth]{figures/synchro_path_I.pdf}
\end{minipage}\hfill
\begin{minipage}[t]{0.41\textwidth}
\vspace{0cm}
\textbf{J}\\

\raggedright \begin{minipage}[!ht]{0.47\textwidth}
\includegraphics[width=\textwidth]{figures/module_inclusion_J1.pdf}
\end{minipage}\hfill
\raggedright \begin{minipage}[!ht]{0.5\textwidth}
\includegraphics[width=\textwidth]{figures/module_inclusion_J2.pdf}
\end{minipage}\hfill
\end{minipage}
```

(continues on next page)

(continued from previous page)

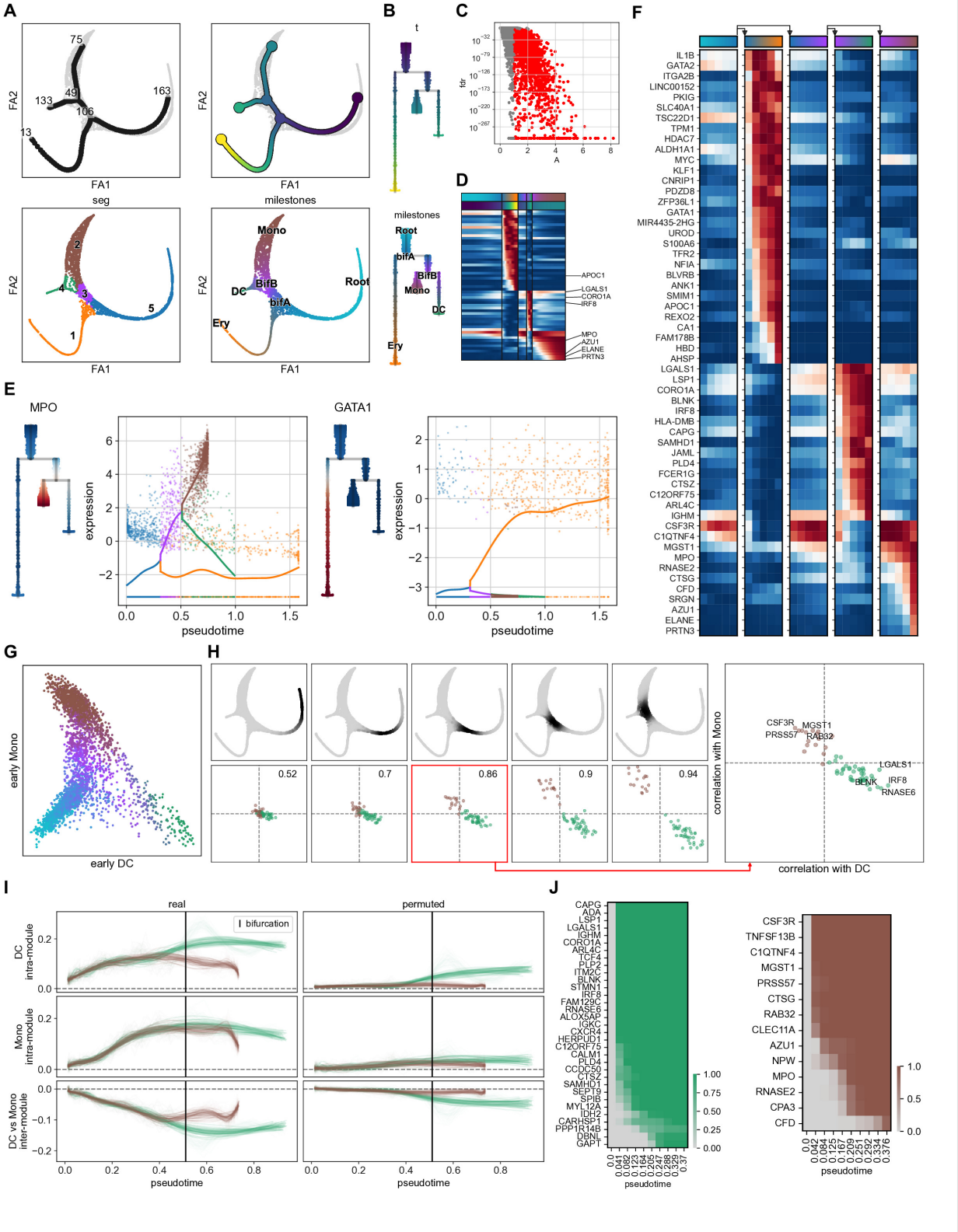
```
\hfill

\clearpage
EOF
echo "\end{document}" >> $1.tex

$2/tectonic -c minimal $1.tex

$2/identify $1.pdf

$2/convert -flatten -density 300 $1.pdf $1.jpg
```

```
warning: accessing absolute path `/usr/share/fonts/truetype/msttcorefonts/Arial_Bold.
↪ttf`; build may not be reproducible in other environments
warning: accessing absolute path `/usr/share/fonts/truetype/msttcorefonts/Arial_Bold_
↪Italic.ttf`; build may not be reproducible in other environments
warning: accessing absolute path `/usr/share/fonts/truetype/msttcorefonts/ariali.ttf`;␣
↪build may not be reproducible in other environments
warning: accessing absolute path `/usr/share/fonts/truetype/msttcorefonts/Arial.ttf`;␣
↪build may not be reproducible in other environments
warning: fig2_supplementary.tex:37: Underfull \hbox (badness 10000) in paragraph at␣
↪lines 34--37
warning: fig2_supplementary.tex:42: Underfull \hbox (badness 10000) in paragraph at␣
↪lines 40--42
warning: fig2_supplementary.tex:45: Underfull \hbox (badness 10000) in paragraph at␣
↪lines 43--45
warning: fig2_supplementary.tex:37: Underfull \hbox (badness 10000) in paragraph at␣
↪lines 34--37
warning: fig2_supplementary.tex:42: Underfull \hbox (badness 10000) in paragraph at␣
↪lines 40--42
warning: fig2_supplementary.tex:45: Underfull \hbox (badness 10000) in paragraph at␣
↪lines 43--45
warning: warnings were issued by the TeX engine; use --print and/or --keep-logs for␣
↪details.
```

```
fig2_supplementary.pdf PDF 612x792 612x792+0+0 16-bit sRGB 3655B 0.000u 0:00.000
```

[47]:
```
from IPython.display import Image
Image(filename=f'{fname}.jpg')
```

[47]:


A
B
C
D
E
F
G
H
I
J
FA1
FA2
seg
milestones
t
fdr
A
Mono
Root
bifA
BifB
DC
Ery
APOC1
LGALS1
CORO1A
IRF8
MPO
AZU1
ELANE
PRTN3
MPO
GATA1
expression
pseudotime
early Mono
early DC
correlation with Mono
correlation with DC
CSF3R
MGST1
PRSS57
RAB32
LGALS1
BLNK
IRF8
RNASE6
real
permuted
bifurcation
DC intra-module
Mono intra-module
DC vs Mono inter-module

1.7 Tree operations

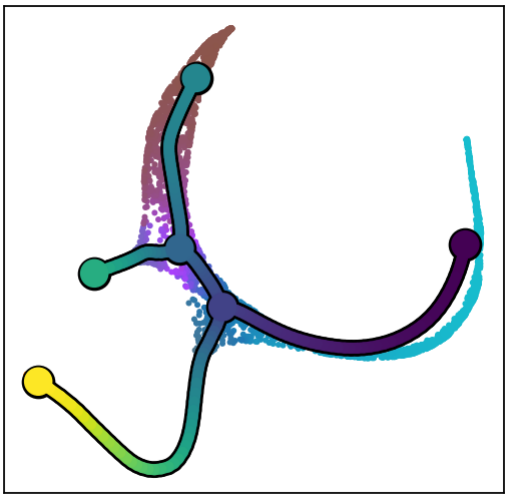
The objective of this notebook is to learn how to perform tree operations, such as subsetting and attachment.

1.7.1 Importing modules and basic settings

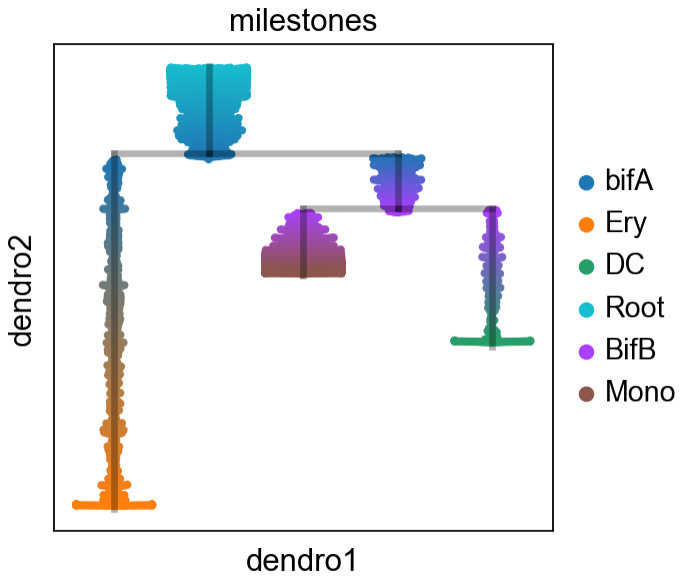
[1]:
```
import warnings
warnings.filterwarnings("ignore")
import scanpy as sc
import scFates as scf

sc.set_figure_params()
```

[2]:
```
adata = sc.read("data/empty_tree.h5ad",backup_url="https://github.com/LouisFaure/scFates_
↪notebooks/raw/main/data/empty_tree.h5ad")
```

[3]:
```
scf.pl.trajectory(adata,basis="draw_graph_fa",color_cells="milestones")
```



[4]:
```
scf.pl.dendrogram(adata,color_milestones=True,color="milestones")
```



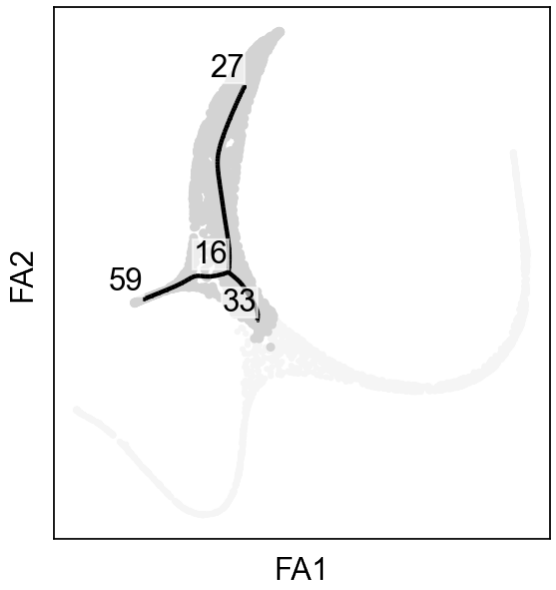
1.7.2 Subsetting of tree (extraction mode)

Subtree

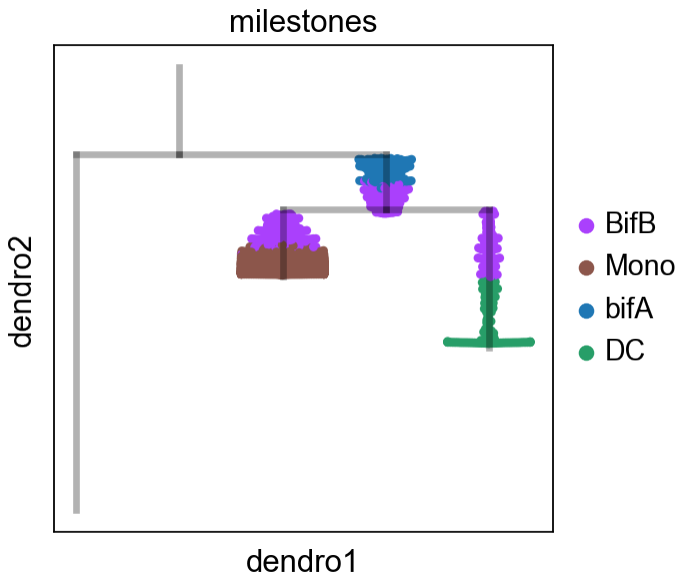
[5]:
```
adata_extracted=scf.tl.subset_tree(adata,root_milestone="bifA",milestones=["Mono","DC"],
↪copy=True)
```

```
subsetting tree
node 33 selected as a root --> added
    .uns['graph']['root'] selected root.
    .uns['graph']['pp_info'] for each PP, its distance vs root and segment assignment.
    .uns['graph']['pp_seg'] segments network information.
projecting cells onto the principal graph
    finished (0:00:01) --> added
    .obs['edge'] assigned edge.
    .obs['t'] pseudotime value.
    .obs['seg'] segment of the tree assigned.
    .obs['milestones'] milestone assigned.
    .uns['pseudotime_list'] list of cell projection from all mappings.
    finished (0:00:00) --> tree extracted
--> added
    .obs['old_milestones'], previous milestones from intial tree
```

[6]:
```
ax=sc.pl.scatter(adata,basis="draw_graph_fa",color="whitesmoke",show=False)
scf.pl.graph(adata_extracted,basis="draw_graph_fa",size_nodes=.1,ax=ax)
```



[7]:
```
scf.pl.dendrogram(adata_extracted,color="milestones")
```



Single branch

[8]:
```
adata_extracted=scf.tl.subset_tree(adata,root_milestone="bifA",milestones=["Ery"],
↪copy=True)
```

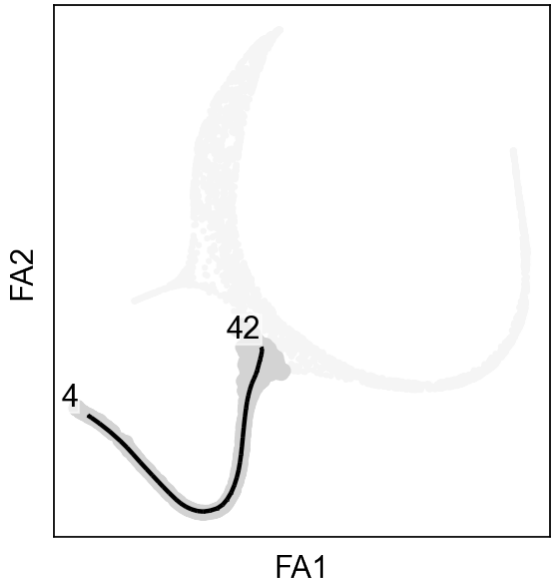
```
subsetting tree
node 42 selected as a root --> added
    .uns['graph']['root'] selected root.
    .uns['graph']['pp_info'] for each PP, its distance vs root and segment assignment.
    .uns['graph']['pp_seg'] segments network information.
```

(continues on next page)

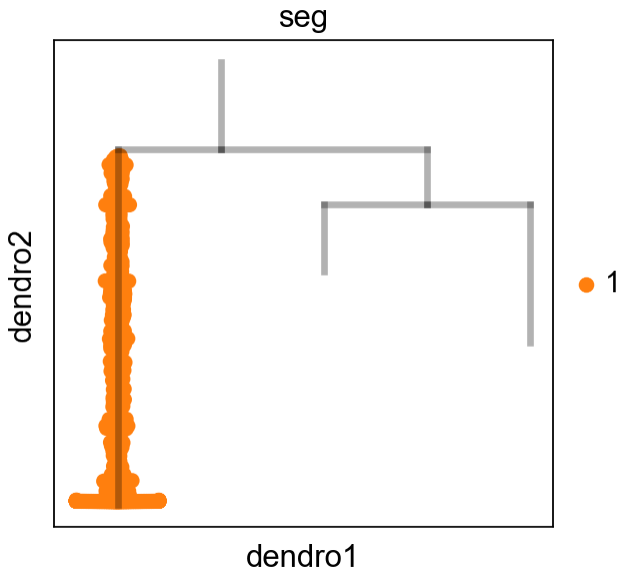
(continued from previous page)

```
projecting cells onto the principal graph
    finished (0:00:01) --> added
    .obs['edge'] assigned edge.
    .obs['t'] pseudotime value.
    .obs['seg'] segment of the tree assigned.
    .obs['milestones'] milestone assigned.
    .uns['pseudotime_list'] list of cell projection from all mappings.
    finished (0:00:00) --> tree extracted
--> added
    .obs['old_milestones'], previous milestones from intial tree
```

[9]:
```
ax=sc.pl.scatter(adata,basis="draw_graph_fa",color="whitesmoke",show=False)
scf.pl.graph(adata_extracted,basis="draw_graph_fa",size_nodes=.1,ax=ax)
```



[10]:
```
scf.pl.dendrogram(adata_extracted,color="seg")
```

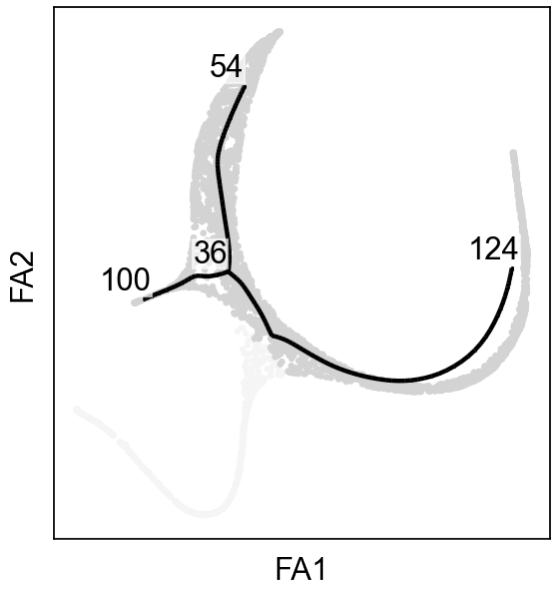


1.7.3 Subsetting of tree (substract mode)

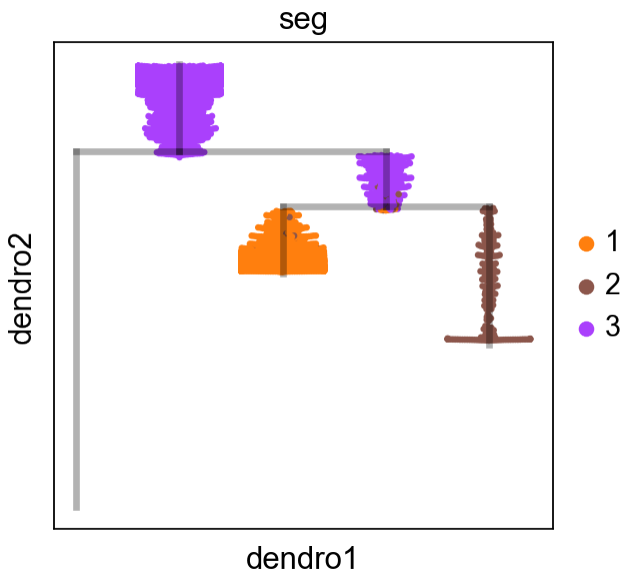
[11]:
```
adata_subtracted=scf.tl.subset_tree(adata,root_milestone="bifA",milestones=["Ery"],
                                    mode="substract",copy=True)
```

```
subsetting tree
node 124 selected as a root --> added
    .uns['graph']['root'] selected root.
    .uns['graph']['pp_info'] for each PP, its distance vs root and segment assignment.
    .uns['graph']['pp_seg'] segments network information.
projecting cells onto the principal graph
    finished (0:00:03) --> added
    .obs['edge'] assigned edge.
    .obs['t'] pseudotime value.
    .obs['seg'] segment of the tree assigned.
    .obs['milestones'] milestone assigned.
    .uns['pseudotime_list'] list of cell projection from all mappings.
    finished (0:00:00) --> tree subsetted
--> added
    .obs['old_milestones'], previous milestones from intial tree
```

[12]:
```
ax=sc.pl.scatter(adata,basis="draw_graph_fa",color="whitesmoke",show=False)
scf.pl.graph(adata_subtracted,basis="draw_graph_fa",size_nodes=.1,ax=ax)
```



[13]:
```
scf.pl.dendrogram(adata_subtracted,color="seg")
```

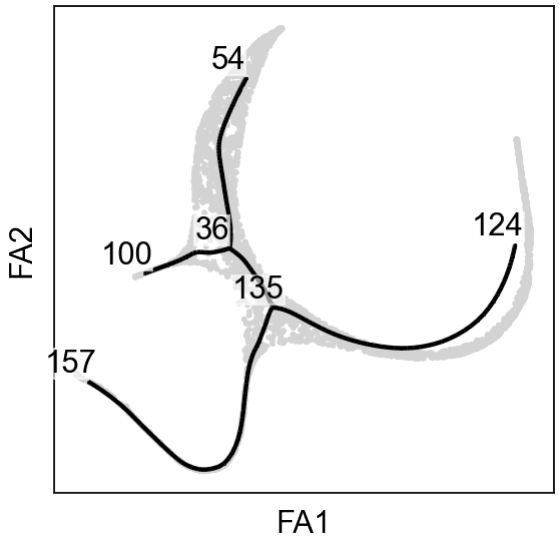


1.7.4 Tree attachment

[14]:
```
adata_attached = scf.tl.attach_tree(adata_subtracted,adata_extracted)
```

```
attaching tree
    merging
    tree refitting
    finished (0:00:02) --> datasets combined
```

[15]:
```
scf.pl.graph(adata_attached,basis="draw_graph_fa",size_nodes=.1)
```



1.8 Exploring Sigma

[1]:
```
import scFates as scf
import scanpy as sc
sc.set_figure_params()
import warnings
```

We are loading a dataset containing a simple bifurcation, already procesed with PCA, UMAP and diffusion maps

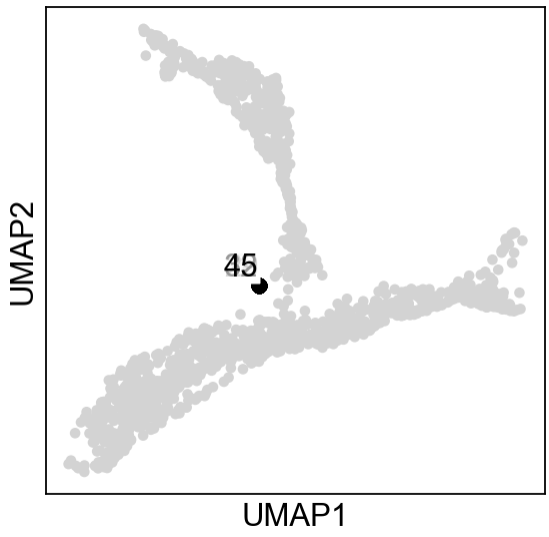
[2]:
```
adata=scf.datasets.test_adata()
```

1.8.1 Example of a bad sigma

Here we use an overly high sigma parameter to demonstrate its danger:

[3]:
```
scf.tl.tree(adata,
            Nodes=50,
            use_rep="X_pca",
            method="ppt",
            ppt_nsteps=50,
            ppt_sigma=100,
            ppt_lambda=1,
            seed=42,plot=True)
```

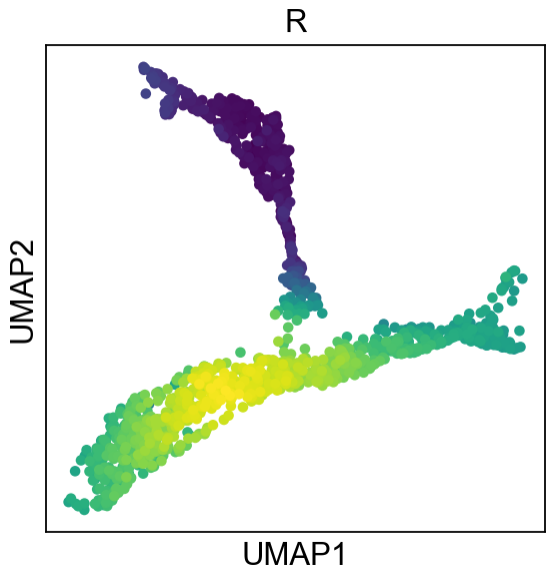
```
inferring a principal tree --> parameters used
    50 principal points, sigma = 100, lambda = 1, metric = euclidean
    fitting:   4%|         | 2/50 [00:01<00:37,  1.27it/s]
    converged
    finished (0:00:01)
```




```
--> added
    .uns['ppt'], dictionnary containing inferred tree.
    .obsm['X_R'] soft assignment of cells to principal points.
    .uns['graph']['B'] adjacency matrix of the principal points.
    .uns['graph']['F'] coordinates of principal points in representation space.
```

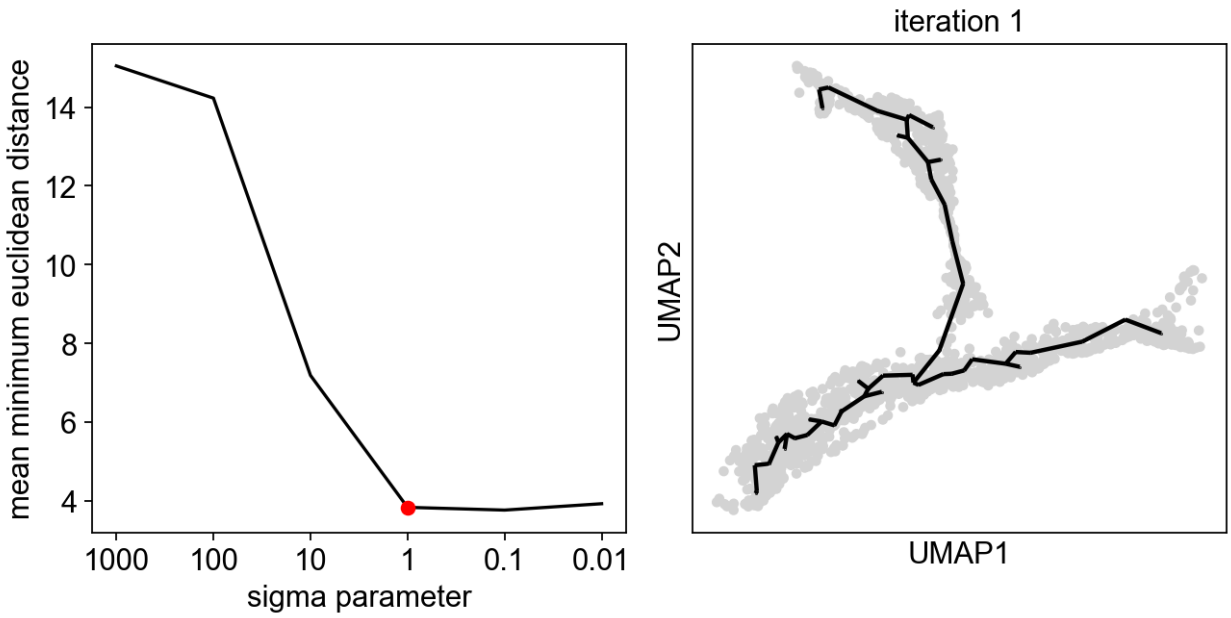
As you can see, all the nodes collapsed in the middle of the data, this is due to the fact that each node is assigned to all cells, which is not what we want. Instead, we are aiming for locallity, were each node is assigned to a subset of cells close to it:

[4]:
```
adata.obs["R"]=adata.obsm["X_R"][:,0]
sc.pl.umap(adata,color="R",colorbar_loc=None)
```



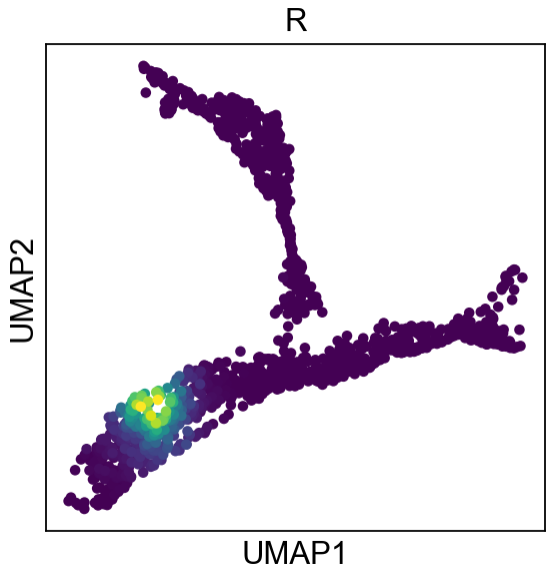
1.8.2 Exploring sigma in PC space

[5]:
```
sig=scf.tl.explore_sigma(adata,
                         Nodes=50,
                         use_rep="X_pca",
                         sigmas=[1000,100,10,1,0.1,0.01],
                         seed=42,plot=True)
```



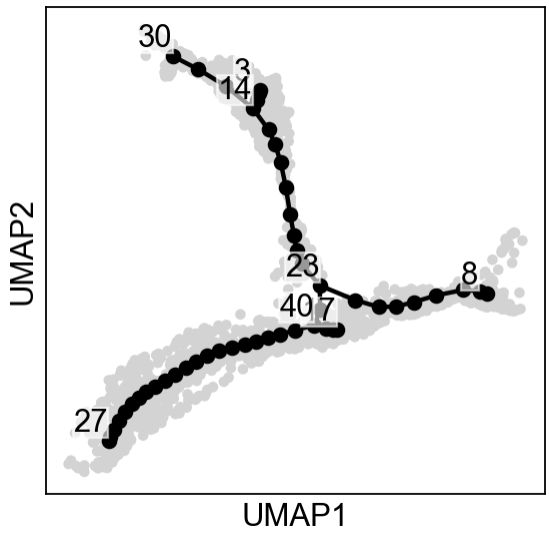
In PC space, a sigma of 1 is suggested, if we look at assigned we obtain a much better locality:

[6]:
```
adata.obs["R"]=adata.obsm["X_R"][:,0]
sc.pl.umap(adata,color="R",colorbar_loc=None)
```



[7]:
```
scf.tl.tree(adata,
            Nodes=50,
            use_rep="X_pca",
            method="ppt",
            ppt_nsteps=50,
            ppt_sigma=sig,
            ppt_lambda=100,
            seed=42,plot=True)
```

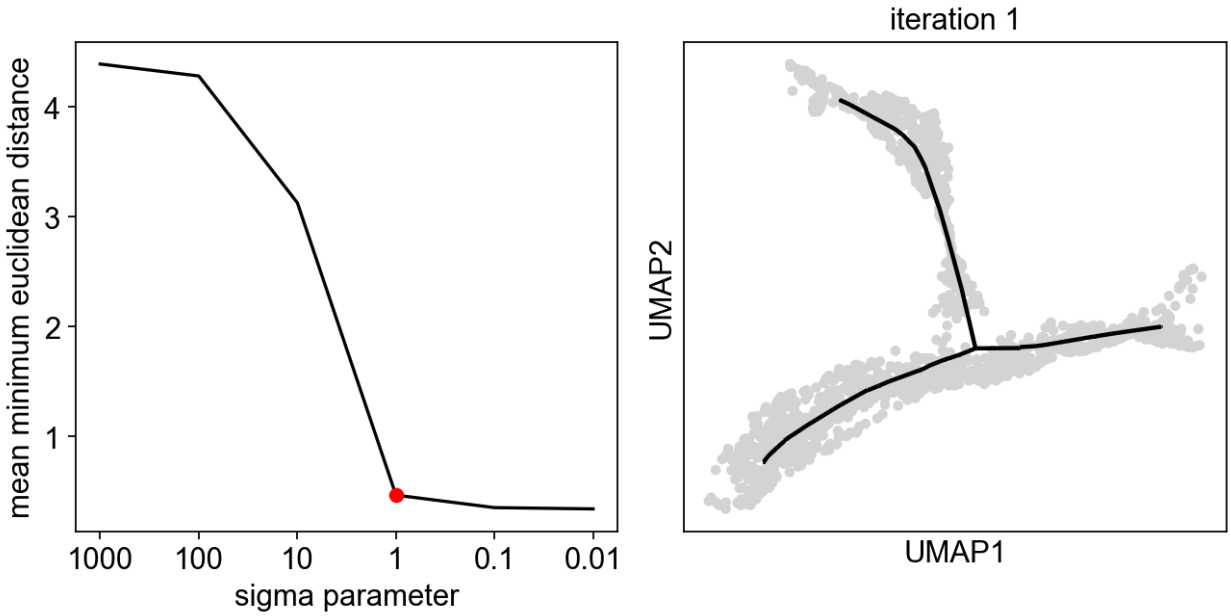
```
inferring a principal tree --> parameters used
    50 principal points, sigma = 1, lambda = 100, metric = euclidean
    fitting:  50%|     | 25/50 [00:01<00:01, 16.18it/s]
    converged
    finished (0:00:01)
```




```
--> added
    .uns['ppt'], dictionnary containing inferred tree.
    .obsm['X_R'] soft assignment of cells to principal points.
    .uns['graph']['B'] adjacency matrix of the principal points.
    .uns['graph']['F'] coordinates of principal points in representation space.
```

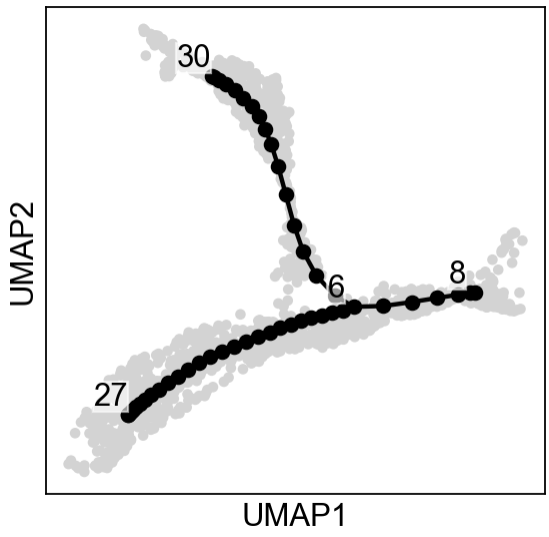
1.8.3 Exploring sigma in UMAP space

[8]:
```
sig=scf.tl.explore_sigma(adata,Nodes=50,use_rep="X_umap",seed=42,plot=True)
```



[9]:
```
scf.tl.tree(adata,
            Nodes=50,
            use_rep="X_umap",
            method="ppt",
            ppt_nsteps=50,
            ppt_sigma=sig,
            ppt_lambda=100,
            seed=42,plot=True)
```

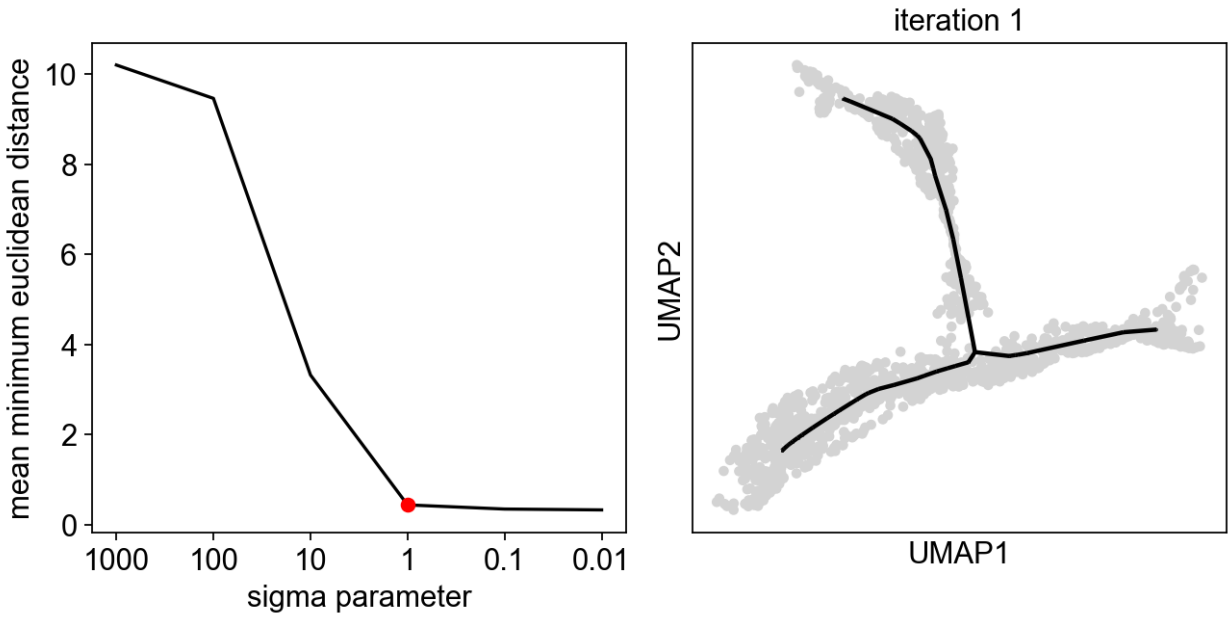
```
inferring a principal tree --> parameters used
    50 principal points, sigma = 1, lambda = 100, metric = euclidean
    fitting:  38%|      | 19/50 [00:00<00:00, 73.28it/s]
    converged
    finished (0:00:00)
```



```
--> added
    .uns['ppt'], dictionnary containing inferred tree.
    .obsm['X_R'] soft assignment of cells to principal points.
    .uns['graph']['B'] adjacency matrix of the principal points.
    .uns['graph']['F'] coordinates of principal points in representation space.
```

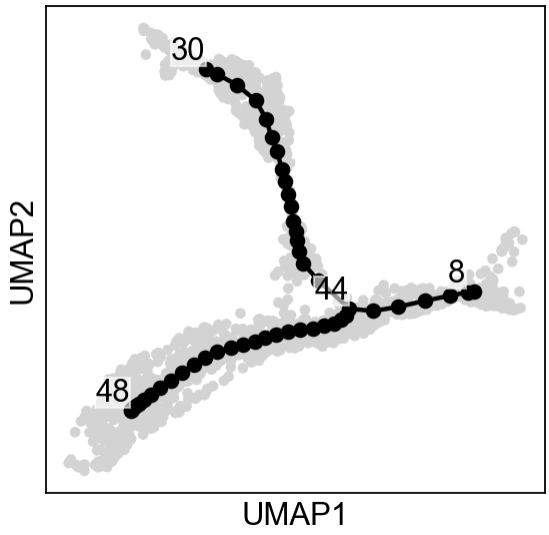
1.8.4 Exploring sigma in diffusion space

[10]:
```
sig=scf.tl.explore_sigma(adata,50,"X_diffusion",seed=42,plot=True)
```



[11]:
```
scf.tl.tree(adata,
            Nodes=50,
            use_rep="X_diffusion",
            method="ppt",
            ppt_nsteps=50,
            ppt_sigma=sig,
            ppt_lambda=100,
            plot=True,
            seed=42)
```

```
inferring a principal tree --> parameters used
    50 principal points, sigma = 1, lambda = 100, metric = euclidean
    fitting:  54%|    | 27/50 [00:00<00:00, 80.19it/s]
    converged
    finished (0:00:00)
```




```
--> added
    .uns['ppt'], dictionnary containing inferred tree.
    .obsm['X_R'] soft assignment of cells to principal points.
    .uns['graph']['B'] adjacency matrix of the principal points.
    .uns['graph']['F'] coordinates of principal points in representation space.
```

1.8.5 Neural crest dataset

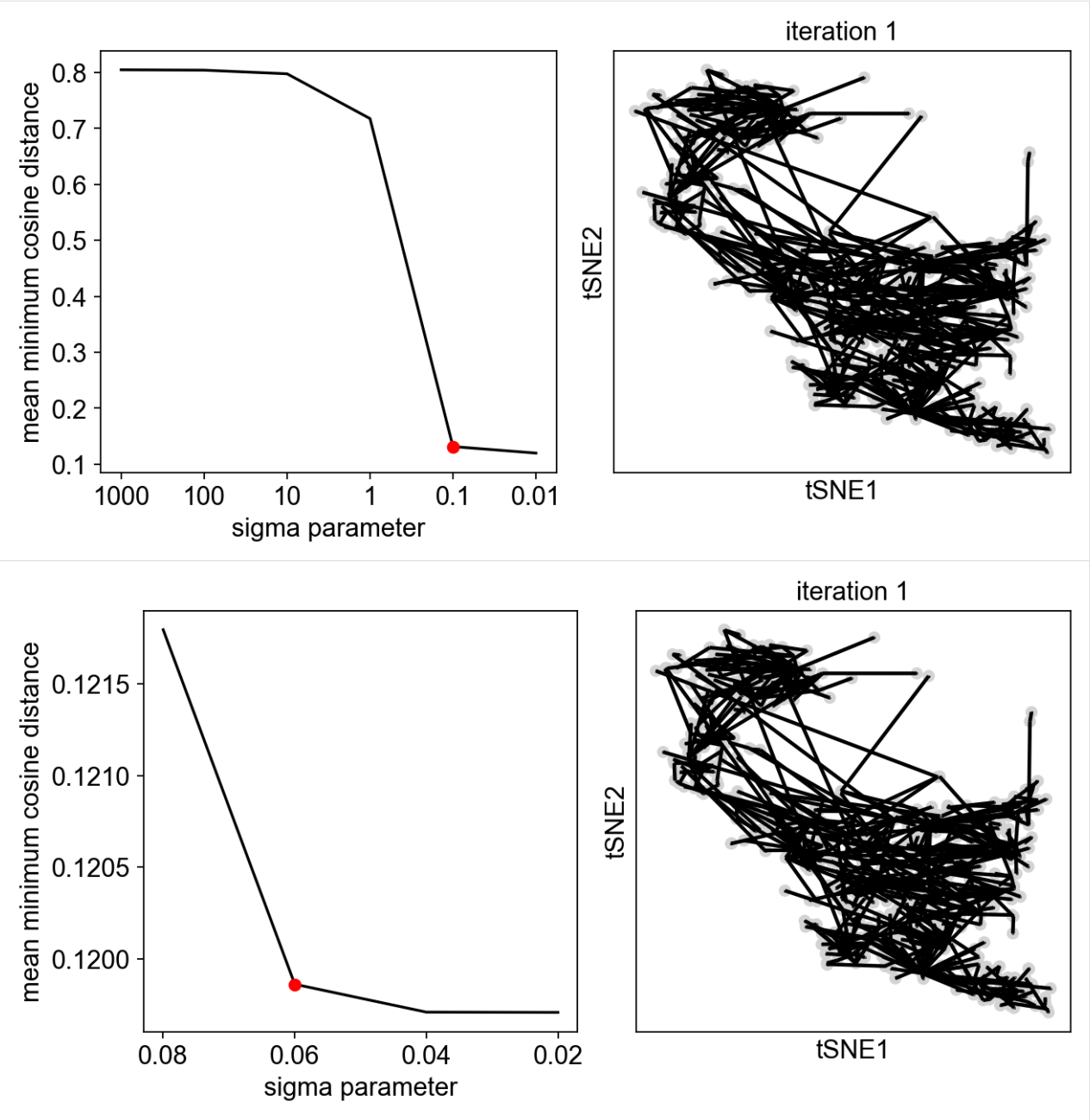
Let's reproduce the results from Soldatov et al. (2019)

[12]:
```
import scanpy as sc
adata=scf.datasets.neucrest19()
```

```
  0%|          | 0.00/9.34M [00:00<?, ?B/s]
```

We can perform two rounds of exploration, to get a finer selection of the sigma value:

[13]:
```
sig=scf.tl.explore_sigma(adata,690,use_rep="X",weight_rep="weights",nsteps=1,metric=
↪"cosine",seed=42,plot=True,
                  second_round=True)
```



Since we are doing only one iteration, the resulting graph can look messy, but our aim is to avoid the collapse of the principal points!

[14]:
```
scf.tl.tree(adata,Nodes=adata.shape[0],use_rep="X",weight_rep="weights",
            ppt_metric="cosine",method="ppt",device="gpu",
            ppt_sigma=sig,ppt_lambda=600,seed=42)
```

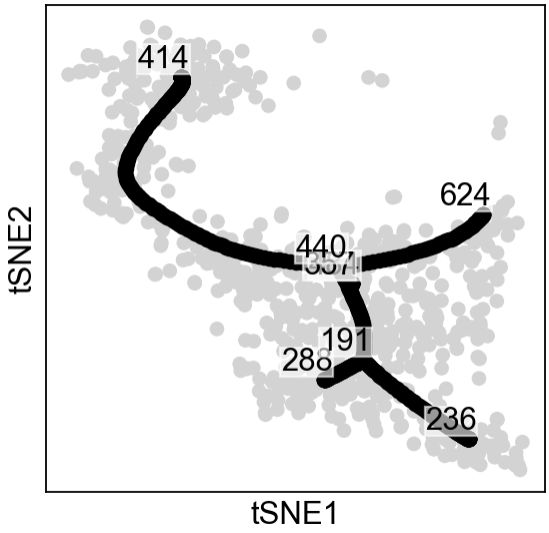
```
inferring a principal tree --> parameters used
    690 principal points, sigma = 0.06, lambda = 600, metric = cosine
    fitting: 100%|| 50/50 [00:12<00:00,  4.01it/s]
    inference not converged (error: 0.00861262787058733)
    finished (0:00:16) --> added
```

(continues on next page)

(continued from previous page)

```
    .uns['ppt'], dictionnary containing inferred tree.
    .obsm['X_R'] soft assignment of cells to principal points.
    .uns['graph']['B'] adjacency matrix of the principal points.
    .uns['graph']['F'] coordinates of principal points in representation space.
```

[15]:
```
scf.pl.graph(adata)
```



1.9 Conversion from CellRank pipeline

The aim of this notebook is to convert resulting analysis from CellRank into a principal tree that can be used by scFates

CellRank aims at identifying fate potentials by considering single cell dynamics as a Markov process (see Lange et al., biorxiv, 2021). This is a great tool for finding the macrostates such as the "tips" of our trajectories, thanks to its powerful probabilistic approach. Here we propose to extend it by converting the fate probabilities into a principal tree, allowing easier interpretation of what is happening "in between" (early biases, bifurcations).

1.9.1 Setting up environment modules and basic settings

Generating the environment

The following needs to be run in the command-line

```
conda create -n scFates -c conda-forge -c r python=3.8 r-mgcv rpy2=3.4.2 -y
conda activate scFates

# Install new jupyter server
conda install -c conda-forge jupyter

# Or add to an existing jupyter server
conda install -c conda-forge ipykernel
python -m ipykernel install --user --name scFates --display-name "scFates"
```

(continues on next page)

(continued from previous page)

```
# Install scFates
pip install scFates
```

Required additional packages

[1]:
```
import sys
!{sys.executable} -m pip -q install scvelo cellrank
```

Loading modules and settings

[2]:
```
import scvelo as scv
import scanpy as sc
import cellrank as cr
import numpy as np

scv.settings.verbosity = 3
scv.settings.set_figure_params('scvelo')
cr.settings.verbosity = 2
```

[3]:
```
import warnings
warnings.simplefilter("ignore", category = UserWarning)
warnings.simplefilter("ignore", category = FutureWarning)
```

1.9.2 Reproduction of CellRank notebook

Here we run a compressed version of the CellRank notebook which reproduces figure 2 of their paper.

[4]:
```
adata = cr.datasets.pancreas()
adata.var_names.name=None
adata.raw = adata # We want to keep all the genes for testing on the resulting tree
scv.pp.filter_genes(adata,min_shared_counts=20)
scv.pp.filter_and_normalize(adata, min_shared_counts=20, n_top_genes=2000)
sc.tl.pca(adata)
sc.pp.neighbors(adata, n_pcs=30, n_neighbors=30)
scv.pp.moments(adata, n_pcs=None, n_neighbors=None)
scv.tl.recover_dynamics(adata, n_jobs=20)
scv.tl.velocity(adata, mode='dynamical')
scv.tl.velocity_graph(adata)

scv.tl.latent_time(adata)
```

```
Filtered out 22024 genes that are detected 20 counts (shared).
Normalized count data: X, spliced, unspliced.
Extracted 2000 highly variable genes.
Logarithmized X.
computing moments based on connectivities
    finished (0:00:00) --> added
```

(continues on next page)

(continued from previous page)

```
    'Ms' and 'Mu', moments of un/spliced abundances (adata.layers)
recovering dynamics (using 20/88 cores)
WARNING: Unable to create progress bar. Consider installing `tqdm` as `pip install tqdm`␣
↪and `ipywidgets` as `pip install ipywidgets`,
or disable the progress bar using `show_progress_bar=False`.
    finished (0:00:38) --> added
    'fit_pars', fitted parameters for splicing dynamics (adata.var)
computing velocities
    finished (0:00:02) --> added
    'velocity', velocity vectors for each individual cell (adata.layers)
computing velocity graph (using 1/88 cores)
    finished (0:00:04) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
computing terminal states
WARNING: Uncertain or fuzzy root cell identification. Please verify.
    identified 1 region 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 using root_cells as prior
    finished (0:00:00) --> added
    'latent_time', shared time (adata.obs)
```

[5]:
```
from cellrank.tl.estimators import GPCCA
weight_connectivities=0.2
mode="stochastic"
n_jobs=40
softmax_scale=None

kernel = cr.tl.transition_matrix(adata,
                                 weight_connectivities=weight_connectivities,
                                 mode=mode,
                                 n_jobs=n_jobs,
                                 softmax_scale=softmax_scale)
g_fwd = GPCCA(kernel)
g_fwd.compute_schur(n_components=20)
n_states = 12
g_fwd.compute_macrostates(cluster_key='clusters', n_states=n_states)

g_fwd.set_terminal_states_from_macrostates(names=['Alpha', 'Beta', 'Epsilon', 'Delta'])
g_fwd.compute_absorption_probabilities()
cr.pl.lineages(adata)
```

```
WARNING: Unable to detect a method for Hessian computation. If using predefined␣
↪functions, consider installing `jax` as `pip install jax jaxlib`
Defaulting to `mode='monte_carlo'` and `n_samples=1000`
Computing transition matrix based on logits using `'monte_carlo'` mode
```

```
/tmp/ipykernel_1233105/1773602998.py:7: DeprecationWarning: `cellrank.tl.transition_
↪matrix` will be removed in version `2.0`. Please use the `cellrank.kernels` or␣
↪`cellrank.estimators` interface instead.
  kernel = cr.tl.transition_matrix(adata,
```

```
Estimating `softmax_scale` using `'deterministic'` mode
```

```
100%|| 2531/2531 [00:05<00:00, 429.55cell/s]
```

```
Setting `softmax_scale=3.7951`
```

```

100%|| 2531/2531 [00:20<00:00, 122.11sample/s]
```

```
    Finish (0:00:27)
Using a connectivity kernel with weight `0.2`
Computing transition matrix based on `adata.obsp['connectivities']`
    Finish (0:00:00)
```

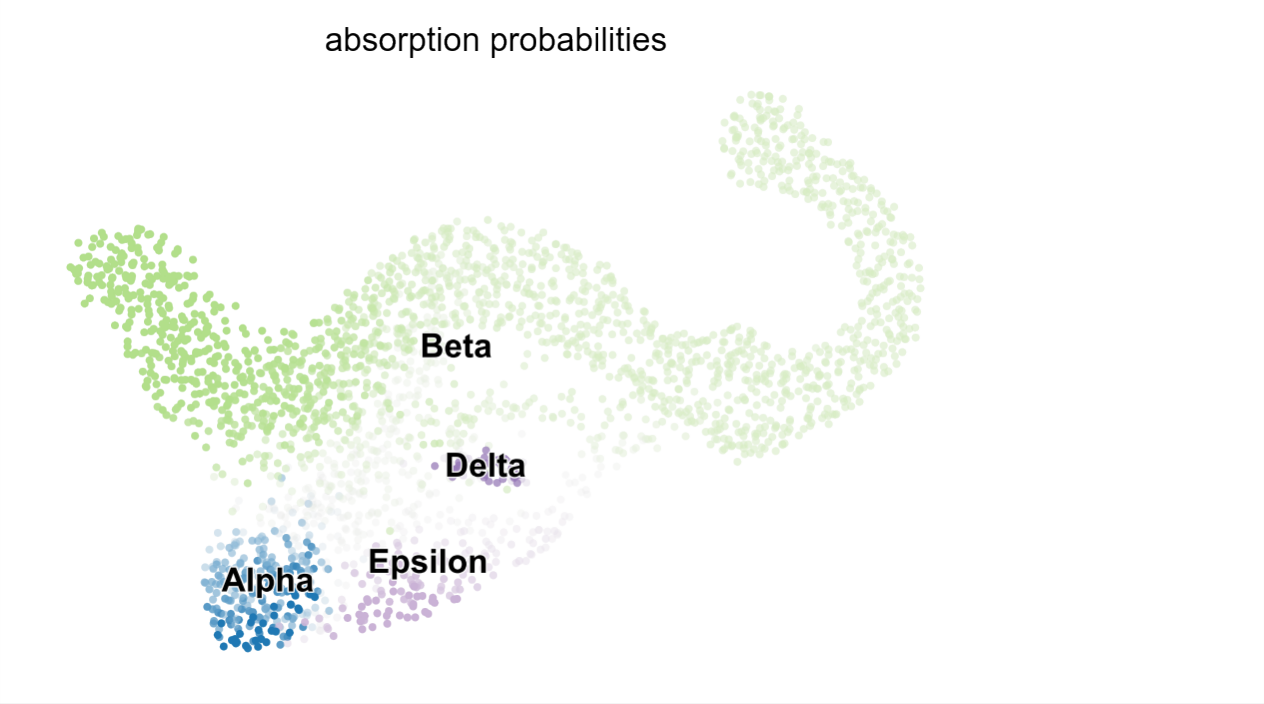
```

```

```
WARNING: Unable to import `petsc4py` or `slepc4py`. Using `method='brandts'`
WARNING: For `method='brandts'`, dense matrix is required. Densifying
Computing Schur decomposition
When computing macrostates, choose a number of states NOT in `[6, 9, 15, 17]`
Adding `adata.uns['eigendecomposition_fwd']`
       `.schur_vectors`
       `.schur_matrix`
       `.eigendecomposition`
    Finish (0:00:07)
Computing `12` macrostates
Adding `.macrostates`
       `.macrostates_memberships`
       `.coarse_T`
       `.coarse_initial_distribution
       `.coarse_stationary_distribution`
       `.schur_vectors`
       `.schur_matrix`
       `.eigendecomposition`
    Finish (0:00:18)
Adding `adata.obs['terminal_states']`
       `adata.obs['terminal_states_probs']`
       `.terminal_states`
       `.terminal_states_probabilities`
       `.terminal_states_memberships
    Finish`
Computing absorption probabilities
WARNING: Unable to import petsc4py. For installation, please refer to: https://petsc4py.
↪readthedocs.io/en/stable/install.html.
Defaulting to `'gmres'` solver.
```

```
100%|| 4/4 [00:00<00:00, 38.09/s]
```

```
Adding `adata.obsm['to_terminal_states']`
       `.absorption_probabilities`
    Finish (0:00:00)
```

```

```



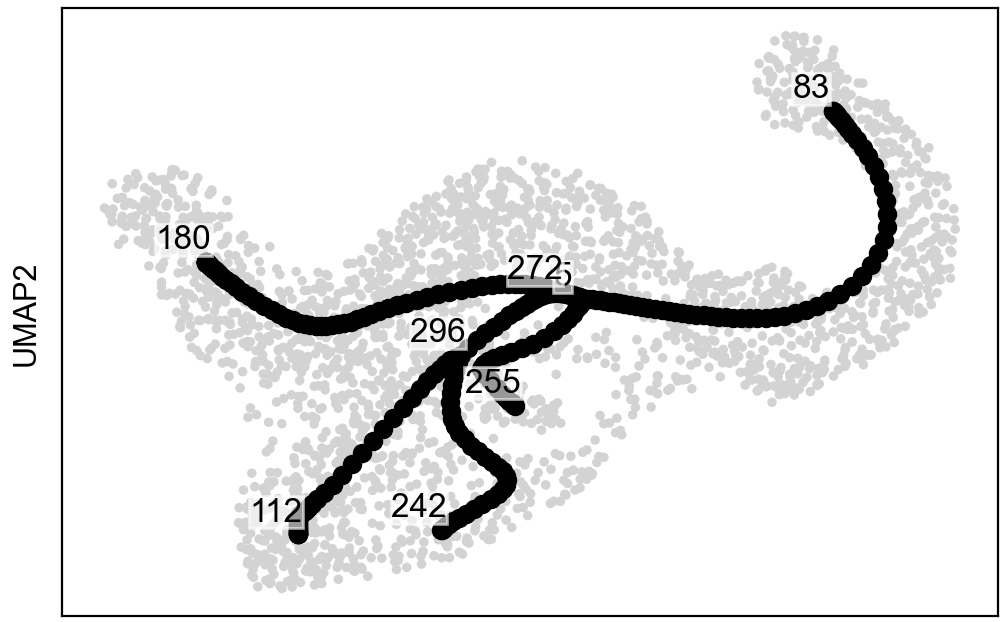
1.9.3 Converting Cellrank output into a principal tree

[6]:
```
warnings.simplefilter("ignore", category = DeprecationWarning)
import scFates as scf
```

[7]:
```
scf.tl.cellrank_to_tree(adata,time="latent_time",Nodes=300,seed=1)
```

```
Converting CellRank results to a principal tree --> with .obsm['X_fates'], created by␣
↪combining:
    .obsm['X_fate_simplex_fwd'] (from cr.pl.circular_projection) and adata.obs['latent_
↪time']

Solving TSP for `4` states
inferring a principal tree inferring a principal tree --> parameters used
    300 principal points, sigma = 0.1, lambda = 100, metric = euclidean
    fitting:  54%|    | 27/50 [00:02<00:02,  9.58it/s]
    converged
    finished (0:00:02) --> added
    .uns['ppt'], dictionnary containing inferred tree.
    .obsm['X_R'] soft assignment of cells to principal points.
    .uns['graph']['B'] adjacency matrix of the principal points.
    .uns['graph']['F'] coordinates of principal points in representation space.

finished (0:00:02)     .obsm['X_fates'] representation used for fitting the tree.
    .uns['graph']['pp_info'].time has been updated with latent_time
    .uns['graph']['pp_seg'].d has been updated with latent_time
```

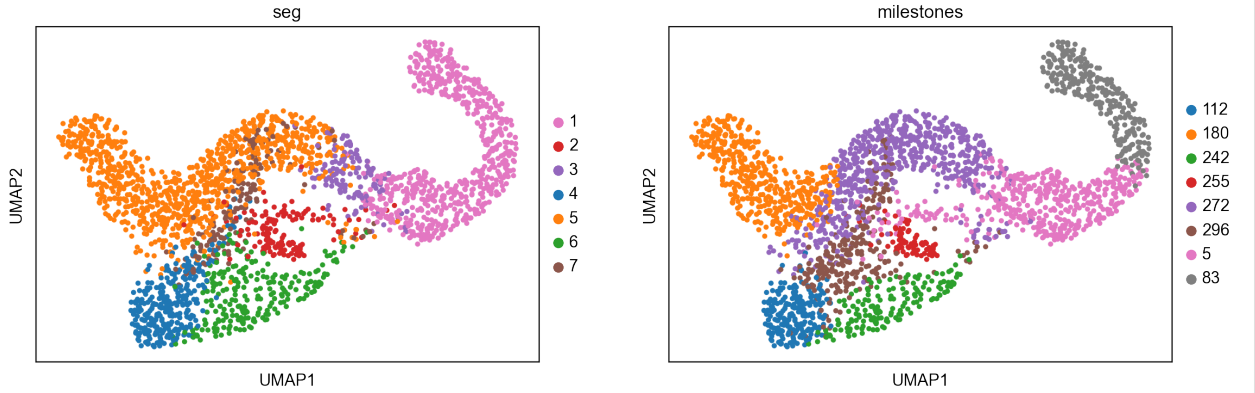
[8]:
```
scf.pl.graph(adata)
```



[9]:
```
scf.tl.root(adata,83)
scf.tl.pseudotime(adata,n_jobs=20,n_map=100,seed=42)
```

```
node 83 selected as a root --> added
    .uns['graph']['root'] selected root.
    .uns['graph']['pp_info'] for each PP, its distance vs root and segment assignment.
    .uns['graph']['pp_seg'] segments network information.
projecting cells onto the principal graph
    mappings: 100%|| 100/100 [00:36<00:00,  2.75it/s]
    finished (0:00:38) --> added
    .obs['edge'] assigned edge.
    .obs['t'] pseudotime value.
    .obs['seg'] segment of the tree assigned.
    .obs['milestones'] milestone assigned.
    .uns['pseudotime_list'] list of cell projection from all mappings.
```

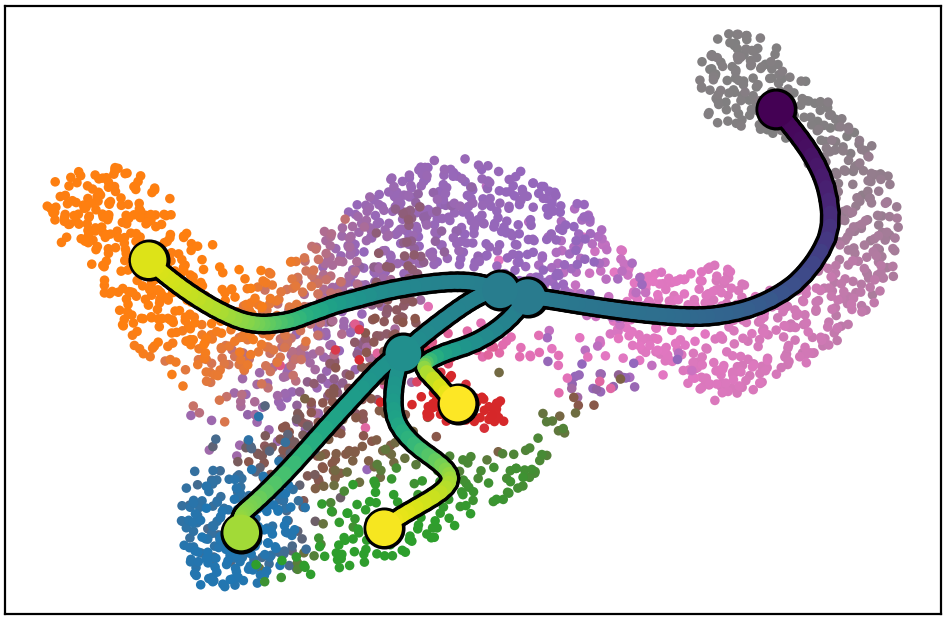
[10]:
```
sc.pl.umap(adata,color=["seg","milestones"])
```



[11]:
```
# to avoid overlapping of labels, we set some intermediate states to empty strings of␣
↪varying length
dct={"112":"Alpha","180":"Beta","5":"Ngn3 high EP","242":"Epsilon","255":"Delta","272":"␣
↪ ","296":" ","83":"Ngn3 low EP"}
```

[12]:
```
scf.tl.rename_milestones(adata,dct)
```

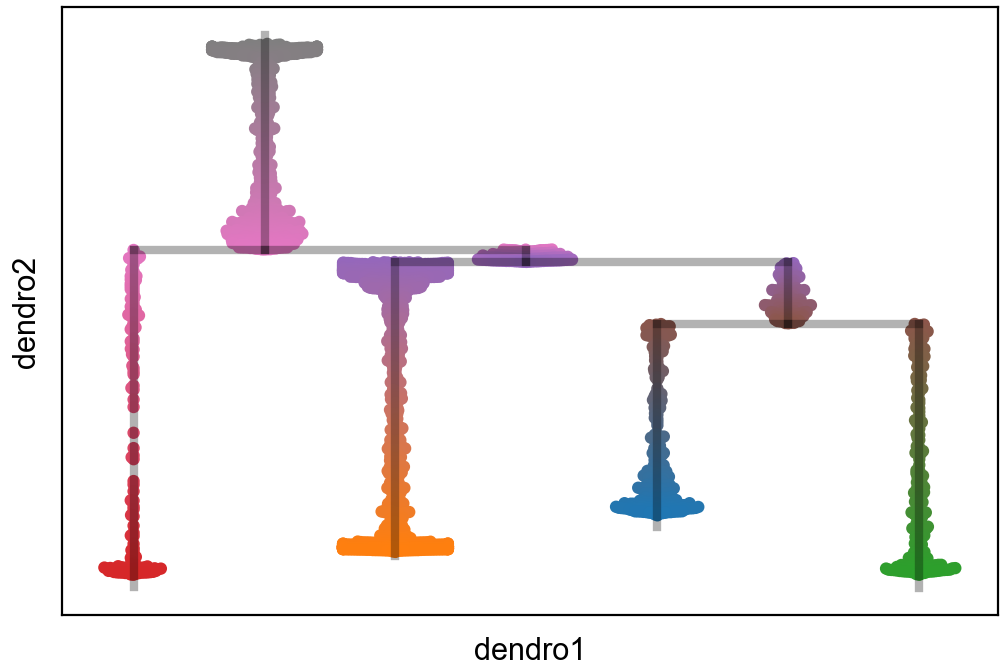
[13]:
```
scf.pl.trajectory(adata,color_cells="milestones")
```



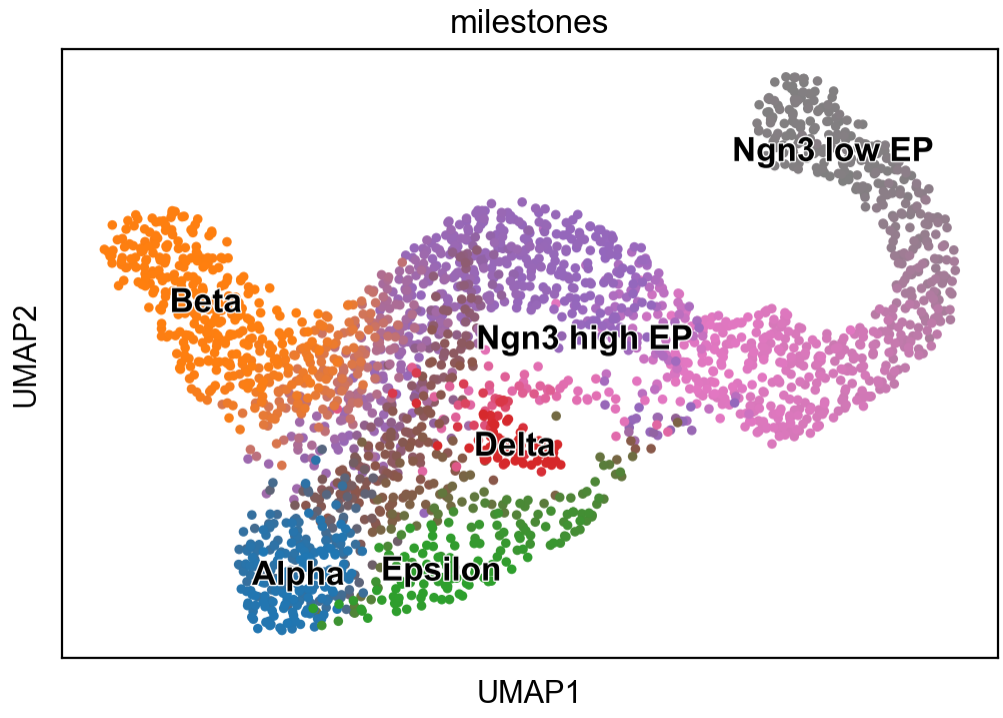
[14]:
```
scf.tl.dendrogram(adata)
```

```
Generating dendrogram of tree
    segment : 100%|| 7/7 [00:01<00:00,  4.07it/s]
    finished (0:00:01) --> added
    .obsm['X_dendro'], new embedding generated.
    .uns['dendro_segments'] tree segments used for plotting.
```

[15]:
```
scf.pl.dendrogram(adata,color_milestones=True)
```



[16]:
```
scf.pl.milestones(adata,annotate=True)
```



1.9.4 How the conversion is performed

Note

The following section is not needed for your own analysis pipeline, it is rather an explanation of what is going on when running the function `scf.tl.cellrank_to_tree`

From the scvelo/CellRank pipeline, we used the latent time estimate:

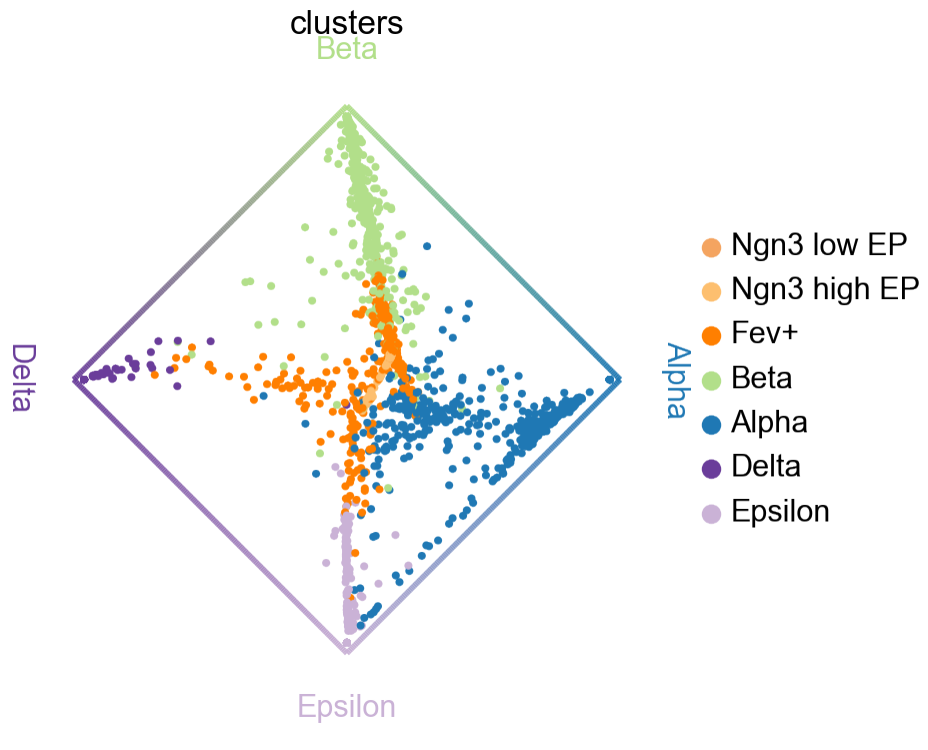
[17]:
```
scv.tl.latent_time(adata)
```

```
computing latent time using root_cells as prior
    finished (0:00:00) --> added
    'latent_time', shared time (adata.obs)
```

And a projection of the fate probabilities:

[18]:
```
cr.pl.circular_projection(adata,"clusters",legend_loc="right")
```

```
Solving TSP for `4` states
```



This projection generated by`cr.pl.circular_projection` can be found under:

[19]:
```
adata.obsm["X_fate_simplex_fwd"]
```

[19]:
```
array([[ 0.17996252,  0.13099097],
       [ 0.27331319, -0.11478642],
       [ 0.17542952,  0.09987978],
       ...,
       [ 0.15714817,  0.19592445],
```

(continues on next page)

(continued from previous page)

```
       [ 0.83724649, -0.12325167],
       [ 0.00368501, -0.80083918]])
```

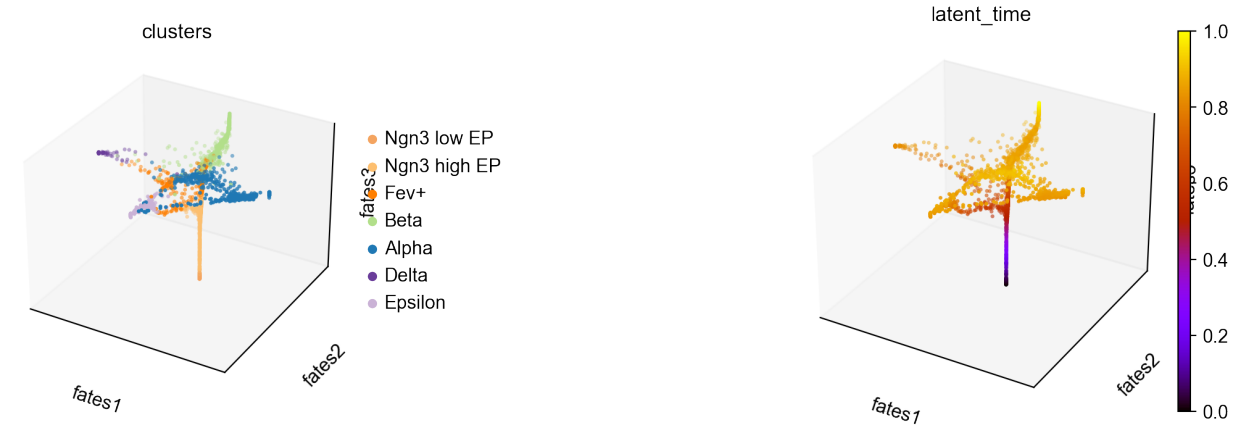
if we add a third dimension using our latent time or any pseudotime measurement, we obtain the following

[20]:
```
adata.obsm["X_fates"]=np.concatenate([adata.obsm["X_fate_simplex_fwd"],
                                      adata.obs["latent_time"].values.reshape(-1,1)],
↪axis=1)
```

[21]:
```
adata.obsm["X_fates"]
```

[21]:
```
array([[ 0.17996252,  0.13099097,  0.8600946 ],
       [ 0.27331319, -0.11478642,  0.88931465],
       [ 0.17542952,  0.09987978,  0.65183399],
       ...,
       [ 0.15714817,  0.19592445,  0.82558667],
       [ 0.83724649, -0.12325167,  0.85034744],
       [ 0.00368501, -0.80083918,  0.83946749]])
```

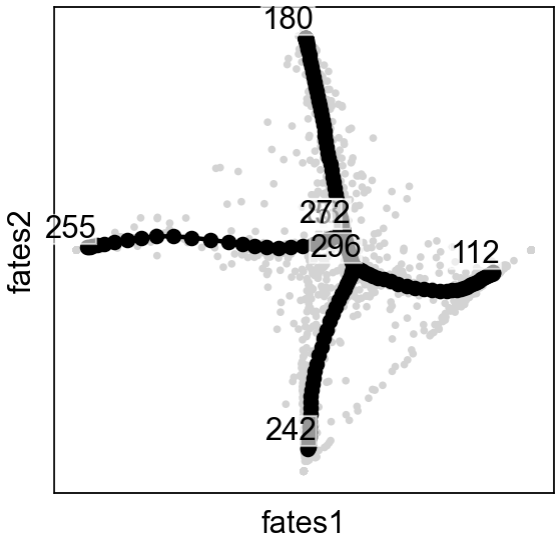
[22]:
```
sc.pl.embedding(adata,basis="fates",projection="3d",color=["clusters","latent_time"],
↪cmap="gnuplot")
```

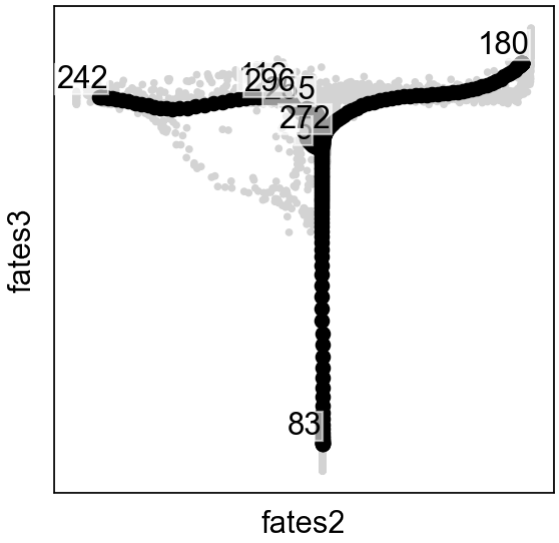


We then fitted a principal tree in this 3d space! Given that each dimension is between 0 and 1, scFates should be able to easily capture all the tips without having to tweak the parameters.

[23]:
```
sc.set_figure_params()
scf.pl.graph(adata,basis="fates")
scf.pl.graph(adata,basis="fates",dimensions=[1,2])
```

```
/home/lfaure/miniconda3/envs/scFates/lib/python3.8/site-packages/scanpy/_settings.py:447:
↪ DeprecationWarning: `set_matplotlib_formats` is deprecated since IPython 7.23,␣
↪directly use `matplotlib_inline.backend_inline.set_matplotlib_formats()`
  IPython.display.set_matplotlib_formats(*ipython_format)
```





What about the case of two terminal states only?

No simplex representation can be generated, in such case, one column from `adata.obsm['to_terminal_states']` is used as 2d representation to be combined with the time estimate.

What about multiple initial states?

We could combine `adata.obsm['X_fate_simplex_fwd']` or `adata.obsm['to_terminal_states']` with `adata.obsm['X_fate_simplex_bwd']` or `adata.obsm['from_intial_states']` into a 4d representation, this will be implemented and tested in a future version.

1.9.5 Downstream analysis

Test and fitting associated genes

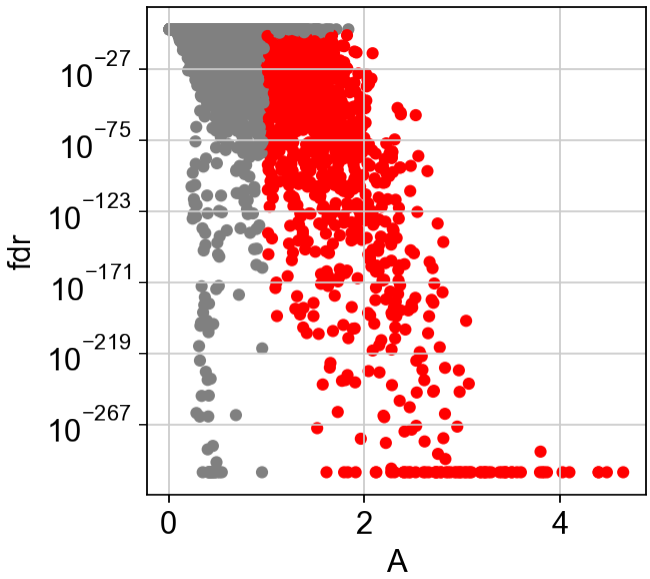
[24]:
```
adata=adata.raw.to_adata()
```

[25]:
```
sc.pp.filter_genes(adata,min_cells=3)
sc.pp.normalize_total(adata,target_sum=1e6)
sc.pp.log1p(adata,base=10)
```

[26]:
```
scf.tl.test_association(adata,n_jobs=20)
```

```
test features for association with the trajectory
    single mapping : 100%|| 14939/14939 [02:34<00:00, 96.67it/s]
    found 1545 significant features (0:02:34) --> added
    .var['p_val'] values from statistical test.
    .var['fdr'] corrected values from multiple testing.
    .var['st'] proportion of mapping in which feature is significant.
    .var['A'] amplitue of change of tested feature.
    .var['signi'] feature is significantly changing along pseudotime.
    .uns['stat_assoc_list'] list of fitted features on the graph for all mappings.
```

[27]:
```
scf.pl.test_association(adata)
```



[28]:
```
scf.tl.fit(adata,n_jobs=20)
```

```
fit features associated with the trajectory
    single mapping : 100%|| 1545/1545 [00:45<00:00, 34.11it/s]
    finished (adata subsetted to keep only fitted features!) (0:00:46) --> added
    .layers['fitted'], fitted features on the trajectory for all mappings.
    .raw, unfiltered data.
```

Identification of branch specific genes

[29]:
```
root='Ngn3 low EP'
miles=['Alpha','Epsilon','Beta','Delta']
```

[30]:
```
scf.tl.test_fork(adata,root_milestone=root,milestones=miles,n_jobs=20,rescale=True)
```

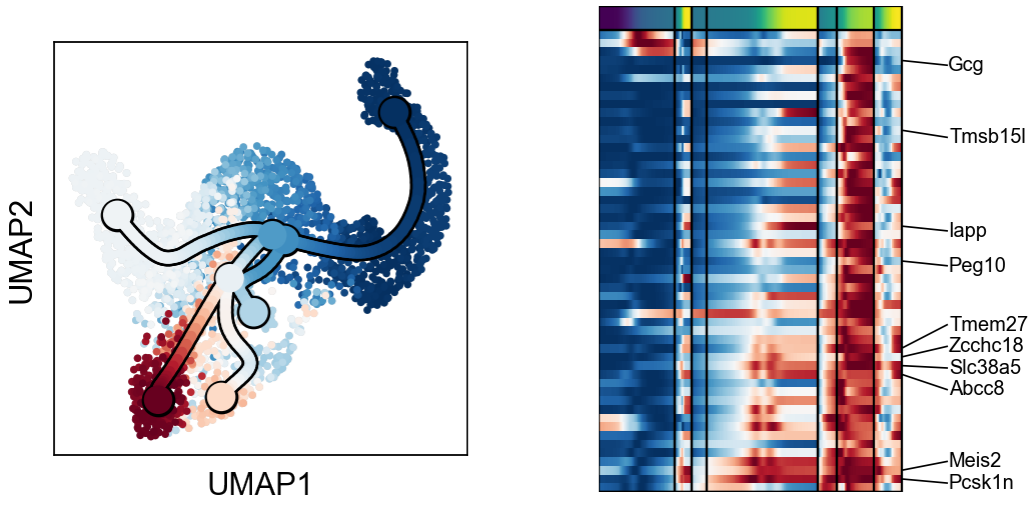
```
testing fork
    single mapping
    Differential expression: 100%|| 1545/1545 [00:25<00:00, 61.35it/s]
    test for upregulation for each leave vs root
    upreg Alpha: 100%|| 397/397 [00:00<00:00, 657.44it/s]
    upreg Epsilon: 100%|| 480/480 [00:00<00:00, 496.55it/s]
    upreg Beta: 100%|| 383/383 [00:00<00:00, 591.96it/s]
    upreg Delta: 100%|| 285/285 [00:00<00:00, 882.93it/s]
    finished (0:00:28) --> added
    .uns['Ngn3 low EP->Alpha<>Epsilon<>Beta<>Delta']['fork'], DataFrame with fork test␣
↪results.
```

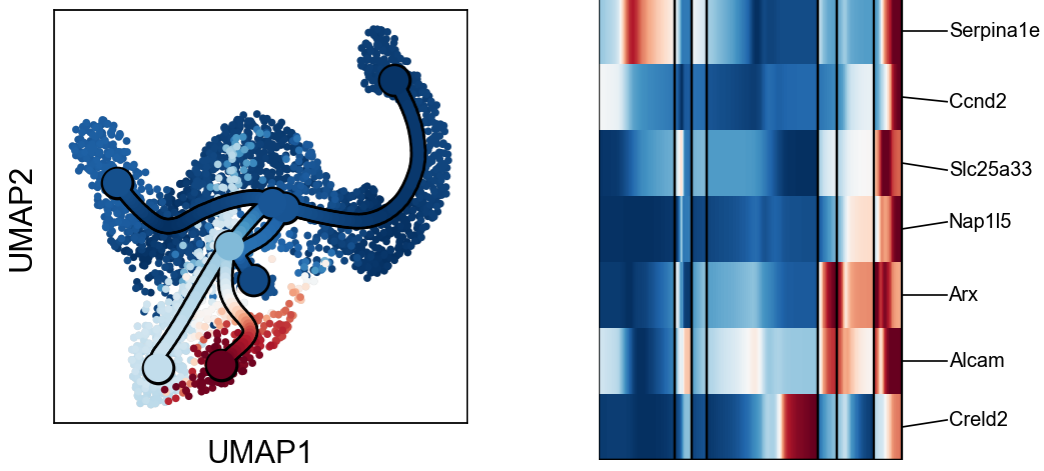
[31]:
```
scf.tl.branch_specific(adata,root_milestone=root,milestones=miles,effect=.3)
```

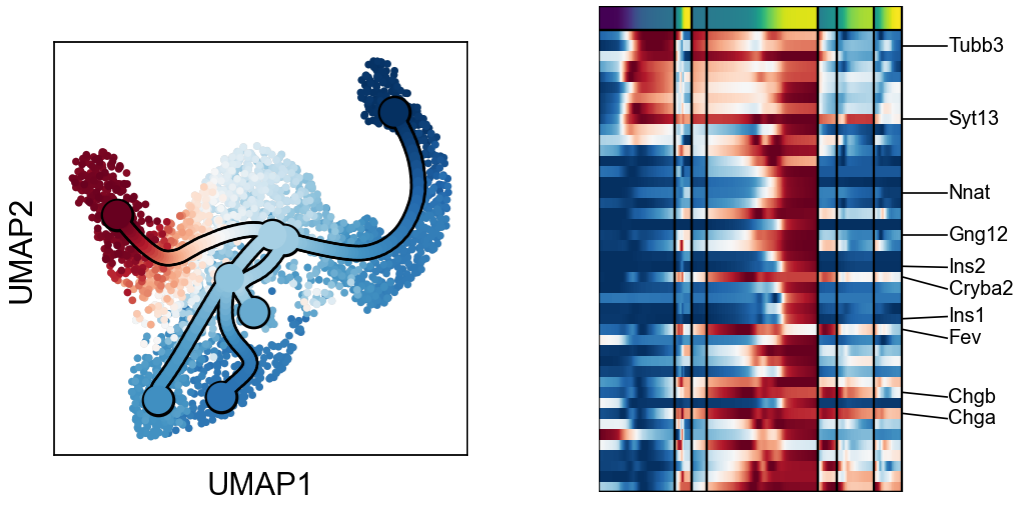
```
    branch specific features: Alpha: 53, Beta: 44, Delta: 8, Epsilon: 7
    finished --> updated
    .uns['Ngn3 low EP->Alpha<>Epsilon<>Beta<>Delta']['fork'], DataFrame updated with␣
↪additionnal 'branch' column.
```

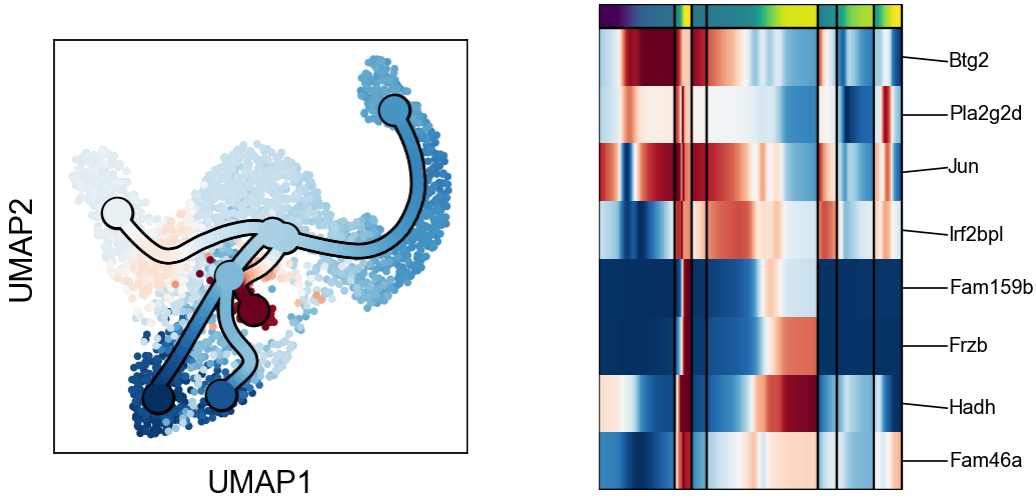
[32]:
```
adata.obs_names.name=None
```

[33]:
```
from matplotlib import MatplotlibDeprecationWarning
warnings.simplefilter("ignore", category = MatplotlibDeprecationWarning)
scf.pl.trends(adata,root_milestone=root,milestones=miles,branch=miles[0])
scf.pl.trends(adata,root_milestone=root,milestones=miles,branch=miles[1])
scf.pl.trends(adata,root_milestone=root,milestones=miles,branch=miles[2])
scf.pl.trends(adata,root_milestone=root,milestones=miles,branch=miles[3])
```




UMAP2
UMAP1
Serpina1e
Ccnd2
Slc25a33
Nap1l5
Arx
Alcam
Creld2


UMAP2
UMAP1
Tubb3
Syt13
Nnat
Gng12
Ins2
Cryba2
Ins1
Fev
Chgb
Chga

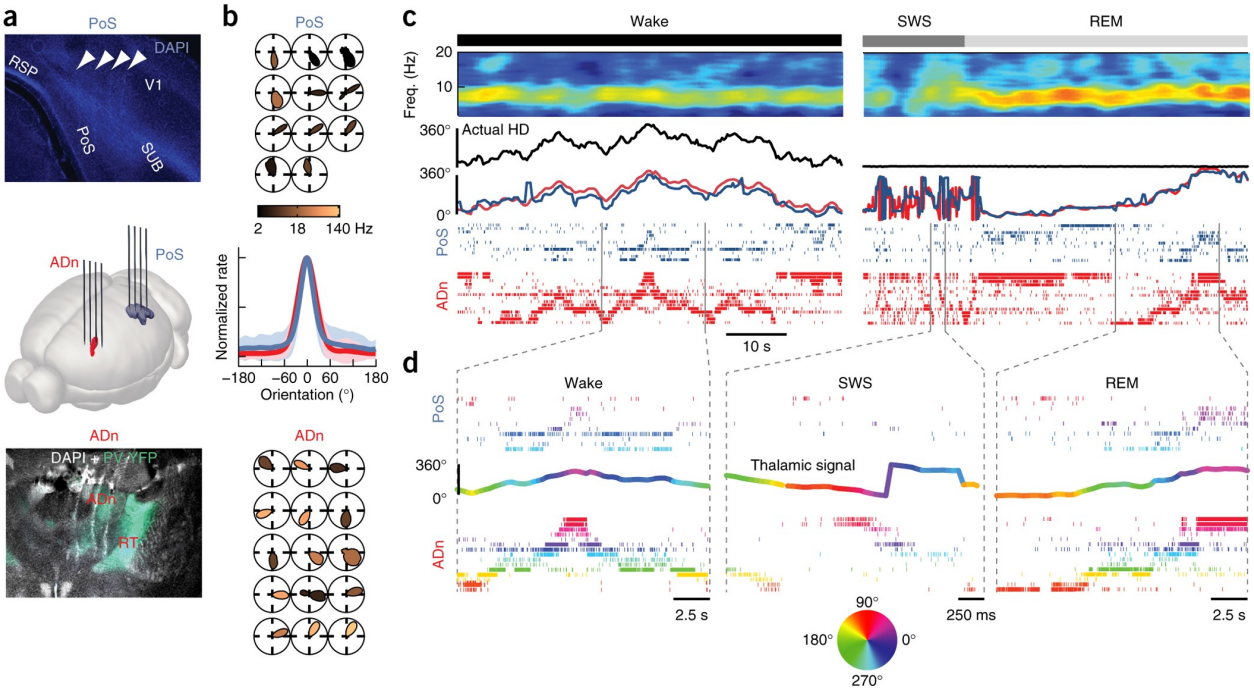

UMAP2
UMAP1
Btg2
Pla2g2d
Jun
Irf2bpl
Fam159b
Frzb
Hadh
Fam46a

1.10 Beyond scRNAseq: neuronal recordings

In this tutorial we are showing the applicability of scFates on other high-dimensional datasets, such as single neuronal acitvity recordings, in order to extract population dynamics.

1.10.1 The dataset and preprocessing

In this example we will use data from Peyrache et al, which is a study on mammalian head direction cell recordings. Here is the figure 1 that summaries the location and data recorded:



The data used for this tutoral includes recordings from 22 neurons located in the anterodorsal thalamic nucleus (ADn) of a single mouse that was awake and foraging in an open environment, with measured head angles.

Preprocessing

Following Chaudhuri et al. method, spike times were converted into time-varying rates. Firing rates were estimated by convolving the spike times with a Gaussian kernel of standard deviation 100 ms. The rates were then replaced by their square root to stabilize the variance. For each timepoint is the related measured angle.

1.10.2 Load libraries and data

[1]:
```
import pandas as pd
import numpy as np
import palantir
import anndata
import warnings
# ignore all future warnings
warnings.filterwarnings("ignore")
```

(continues on next page)

(continued from previous page)

```
import scFates as scf
import scanpy as sc
import scanpy.external as sce
import seaborn as sns
import matplotlib.colors as col
import seaborn
seaborn.reset_orig()
%matplotlib inline

N = 256
flat_huslmap = col.ListedColormap(sns.color_palette('husl',N))

angle_u = 'https://github.com/LouisFaure/scFates_notebooks/raw/main/data/angles.csv.gz'
counts_u = 'https://github.com/LouisFaure/scFates_notebooks/raw/main/data/counts.csv.gz'
```

```
findfont: Font family ['Raleway'] not found. Falling back to DejaVu Sans.
```

[2]:
```
counts=pd.read_csv(counts_u,header=None)
counts.index=counts.index.astype(str)
counts.columns=counts.columns.astype(str)
counts.shape
```

[2]:
```
(22841, 22)
```

[3]:
```
adata=anndata.AnnData(counts)
adata.obs['angles']=np.degrees(pd.read_csv(angle_u, header=None).values)
```

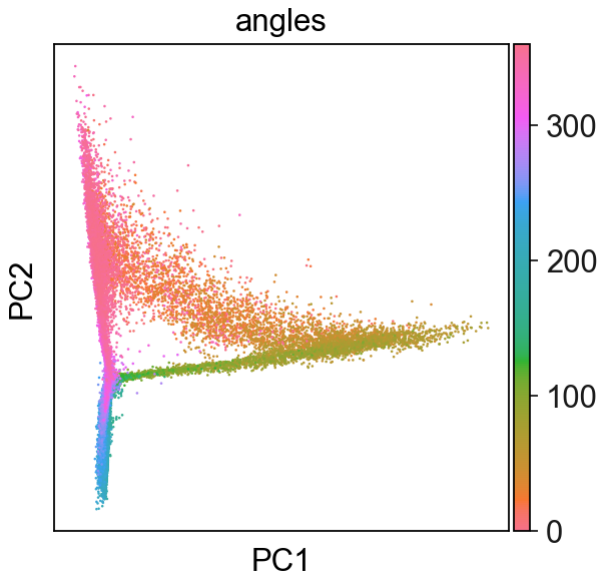
[4]:
```
adata.obs['time']=np.arange(0,adata.shape[0]*0.05,0.05)
```

1.10.3 Dimensionality reduction

PCA

[5]:
```
sc.pp.pca(adata)
```

[6]:
```
sc.set_figure_params()
sc.pl.pca(adata,color="angles",cmap=flat_huslmap)
```



Multi-scale diffusion maps

[7]:
```
sce.tl.palantir(adata)
adata
```

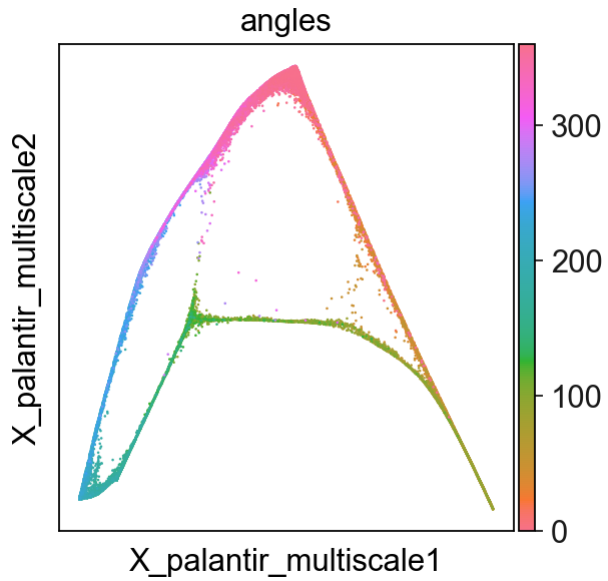
```
Determing nearest neighbor graph...
```

[7]:
```
AnnData object with n_obs × n_vars = 22841 × 22
    obs: 'angles', 'time'
    uns: 'pca', 'palantir_EigenValues'
    obsm: 'X_pca', 'X_palantir_diff_comp', 'X_palantir_multiscale'
    varm: 'PCs'
    layers: 'palantir_imp'
    obsp: 'palantir_diff_op'
```

[8]:
```
adata.obsm["X_palantir_multiscale"].shape
```

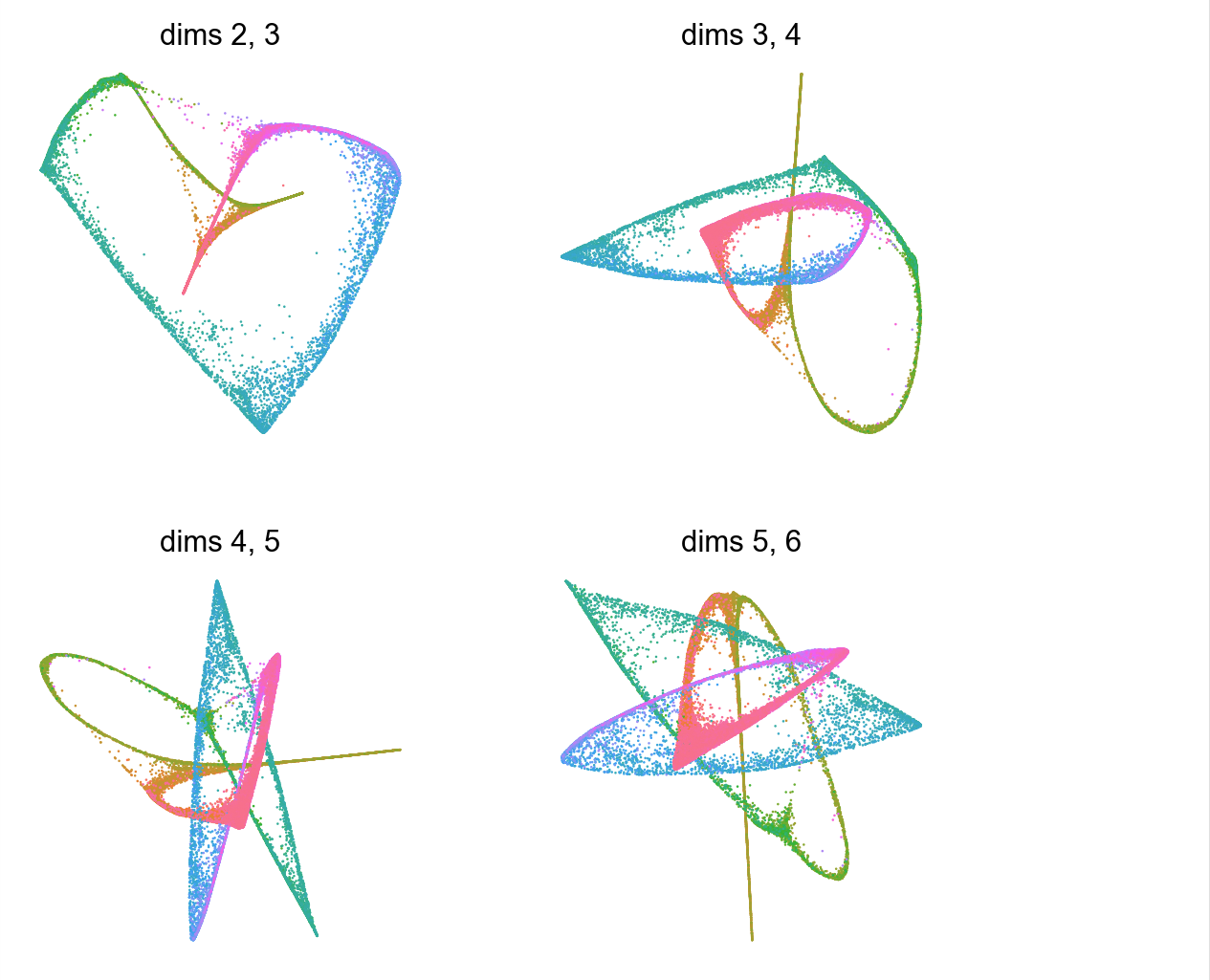
[8]:
```
(22841, 7)
```

[9]:
```
sc.pl.embedding(adata,basis='X_palantir_multiscale',color='angles',cmap=flat_huslmap)
```



Show several dimensions of multiscale diffusion space

[10]:
```
import matplotlib.pyplot as plt
fig, axs = plt.subplots(2,2,figsize=(8,8))
axs=axs.ravel()
for d in np.arange(2,6):
    sc.pl.embedding(adata,basis='X_palantir_multiscale',
                    title="dims %d, %d"%(d,d+1),
                    dimensions=(int(d),int(d+1)),
                    color='angles',cmap=flat_huslmap,
                    show=False,ax=axs[d-2],frameon=False)

    # complex way to remove the side colorbar
    axs[d-2].set_box_aspect(aspect=1)
    fig = axs[d-2].get_gridspec().figure
    cbar = np.argwhere(
        ["colorbar" in a.get_label() for a in fig.get_axes()]
    ).ravel()
    if len(cbar) > 0:
        fig.get_axes()[cbar[0]].remove()
```

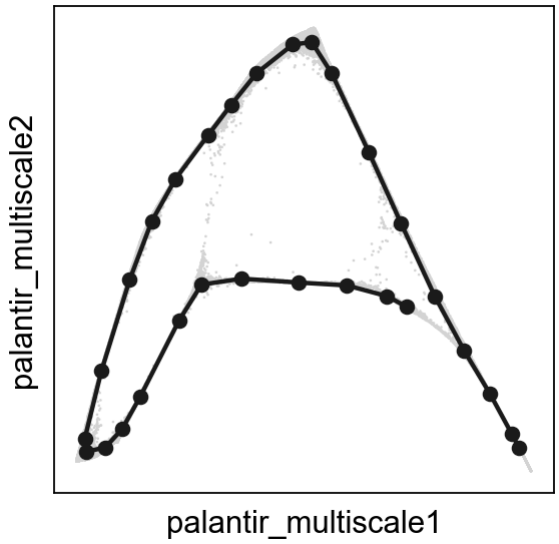


1.10.4 Principal circle fitting

[11]:
```
scf.tl.circle(adata,Nodes=30,use_rep="X_palantir_multiscale",device="gpu")
```

```
inferring a principal circle --> parameters used
    30 principal points, mu = 0.1, lambda = 0.01
    there are 1 non assigned nodes
    finished (0:00:16) --> added
    .uns['epg'] dictionnary containing inferred elastic circle generated from elpigraph.
    .obsm['X_R'] hard assignment of cells to principal points.
    .uns['graph']['B'] adjacency matrix of the principal points.
    .uns['graph']['F'], coordinates of principal points in representation space.
```

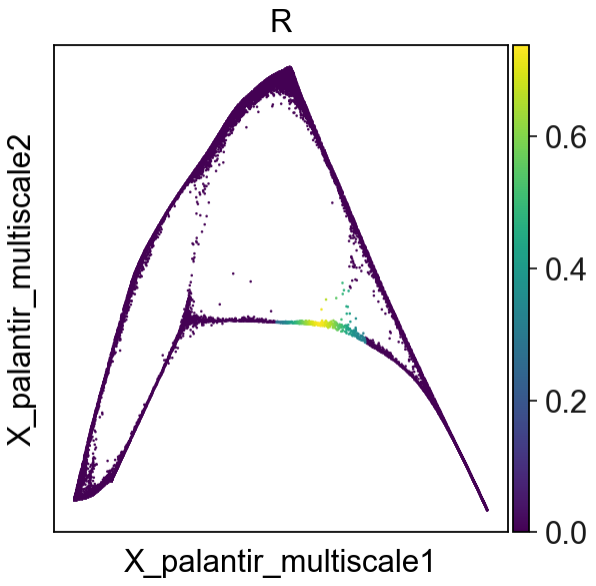
[12]:
```
scf.pl.graph(adata,basis='palantir_multiscale')
```



Converting to soft assigment matrix

ElPiGraph lead to a hard assignment matrix R, rendering it impossible to perform multiple mappings to account for variability, as the assignment to nodes are either 0 or 1:

[13]:
```
adata.obs["R"]=adata.obsm["X_R"][:,25]
sc.pl.embedding(adata,basis='X_palantir_multiscale',color='R')
```

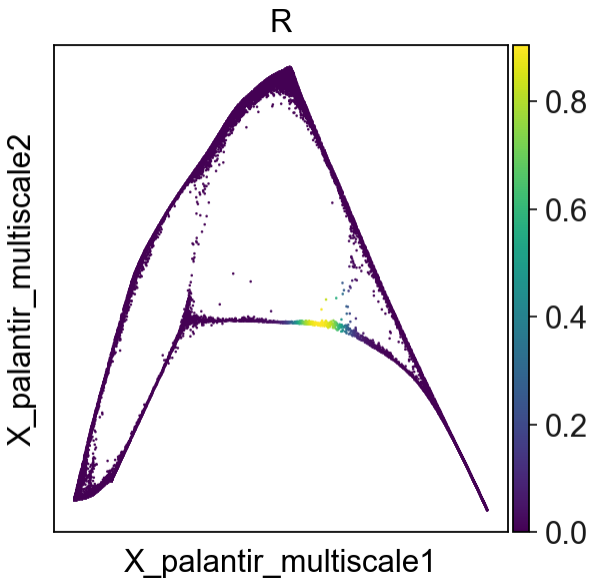


To alleviate this limitation, we can apply one iteration of SimplePPT algorithm, generating a soft assignment matrix R, with continuous values between 0 and 1:

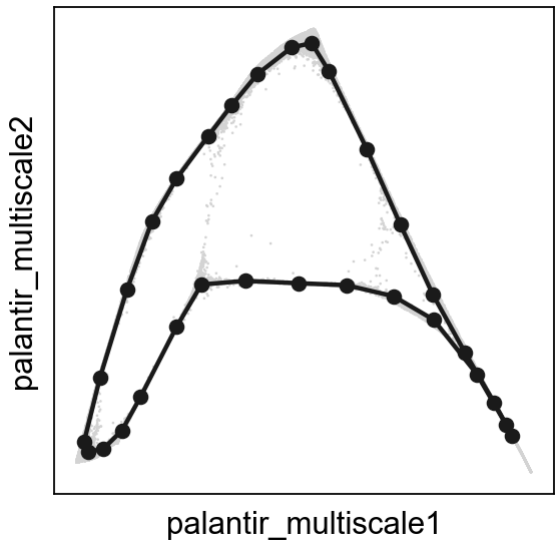
[14]:
```
scf.tl.convert_to_soft(adata,sigma=1,lam=100)
```

```
Converting R into soft assignment matrix
    finished (0:00:00) --> updated
    .obsm['X_R'] converted soft assignment of cells to principal points.
    .uns['graph']['F'] coordinates of principal points in representation space.
```

[15]:
```
adata.obs["R"]=adata.obsm["X_R"][:,25]
sc.pl.embedding(adata,basis='X_palantir_multiscale',color='R')
```



[16]:
```
scf.pl.graph(adata,basis='palantir_multiscale')
```



1.10.5 Projecting activities over pseudotime

Select root node characterized by activity at minimum angle

[17]:
```
scf.tl.root(adata,root="angles",min_val=True)
```

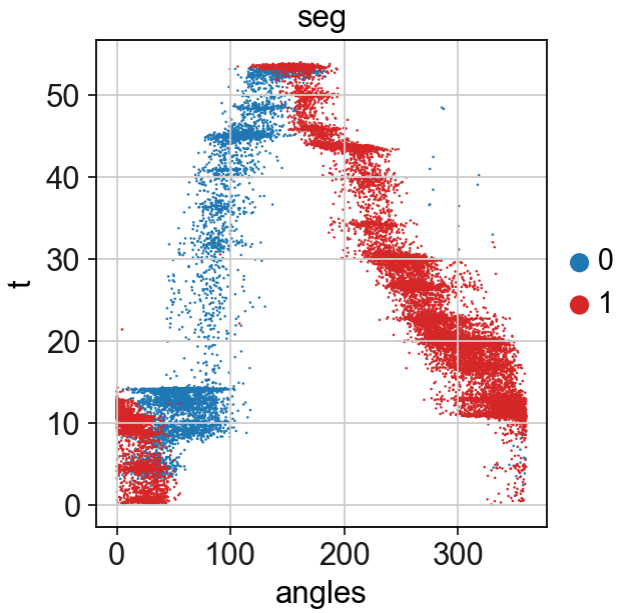
```
automatic root selection using angles values
node 27 selected as a root --> added
    .uns['graph']['root'] selected root.
    .uns['graph']['pp_info'] for each PP, its distance vs root and segment assignment.
    .uns['graph']['pp_seg'] segments network information.
```

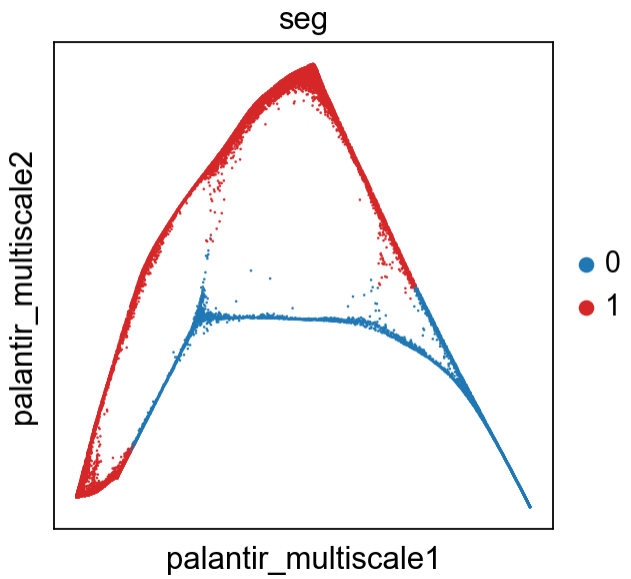
To enable multiple mappings, the circle is considered as a trajectory composed of two segments from the selected root and clostest node converging to the furthest node.

[18]:
```
scf.tl.pseudotime(adata,n_jobs=50,n_map=50,seed=42)
```

```
projecting cells onto the principal graph
    mappings: 100%|| 50/50 [00:58<00:00,  1.16s/it]
    finished (0:01:13) --> added
    .obs['edge'] assigned edge.
    .obs['t'] pseudotime value.
    .obs['seg'] segment of the tree assigned.
    .obs['milestones'] milestone assigned.
    .uns['pseudotime_list'] list of cell projection from all mappings.
```

[19]:
```
sc.pl.scatter(adata,x='angles',y='t',color="seg",palette=["tab:blue","Tab:red"])
sc.pl.embedding(adata,color="seg",basis='palantir_multiscale')
```





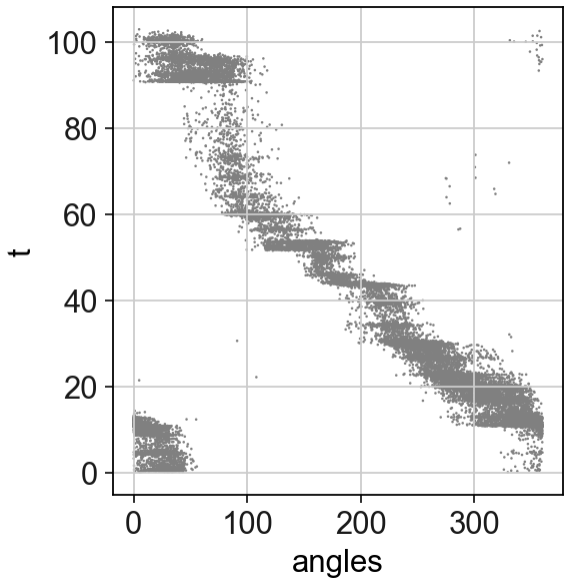
Unroll the circle to get a pseudotime value for all parts of the cicle

To obtain a pseudotime value that differs between the two segments, one can unroll the circle into a single curved trajectory starting from the root and ending to its closest node.

[20]:
```
scf.tl.unroll_circle(adata)
```

```
--> updated
    .obs['t'] assigned pseudotime value.
    .obs['seg'] assigned segment of the tree.
    .uns['graph']['root'] selected root.
    .uns['graph']['pp_info'] for each PP, its distance vs root and segment assignment.
    .uns['graph']['pp_seg'] segments network information.
```

[21]:
```
sc.pl.scatter(adata,x='angles',y='t')
```



GAM fit of single neuron activities

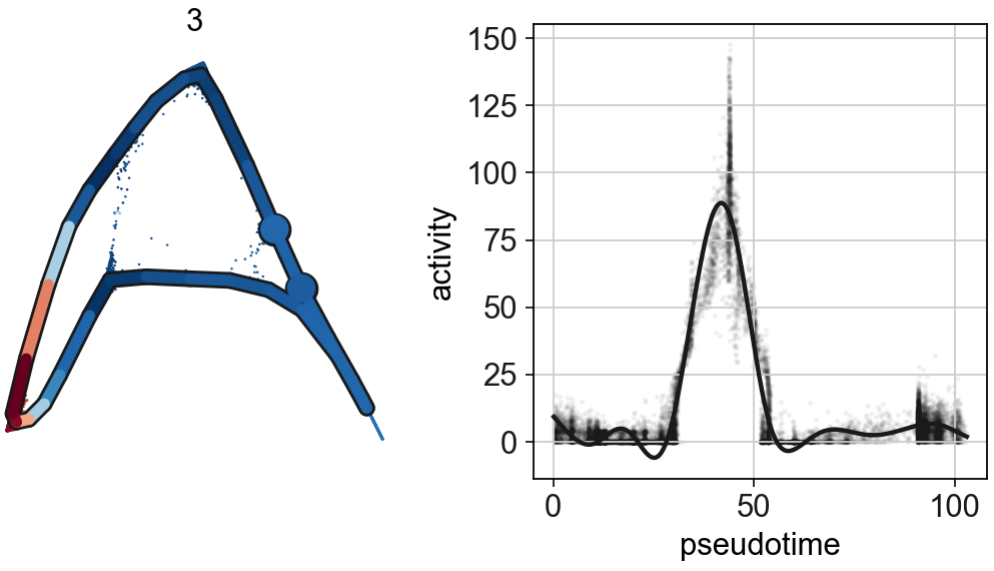
[22]:
```
scf.tl.fit(adata,features=adata.var_names,n_jobs=10)
```

```
fit features associated with the trajectory
    single mapping : 100%|| 22/22 [00:25<00:00,  1.15s/it]
    finished (adata subsetted to keep only fitted features!) (0:00:25) --> added
    .layers['fitted'], fitted features on the trajectory for all mappings.
    .raw, unfiltered data.
```

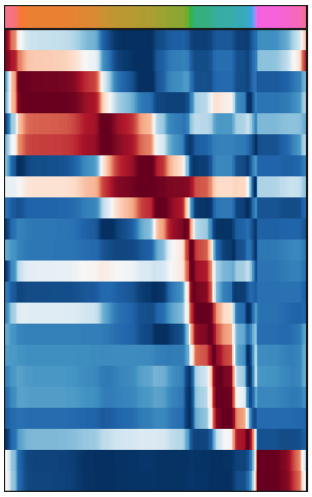
[23]:
```
adata.uns["graph"]["milestones"]
```

[23]:
```
{'10': 10, '24': 24, '27': 27}
```

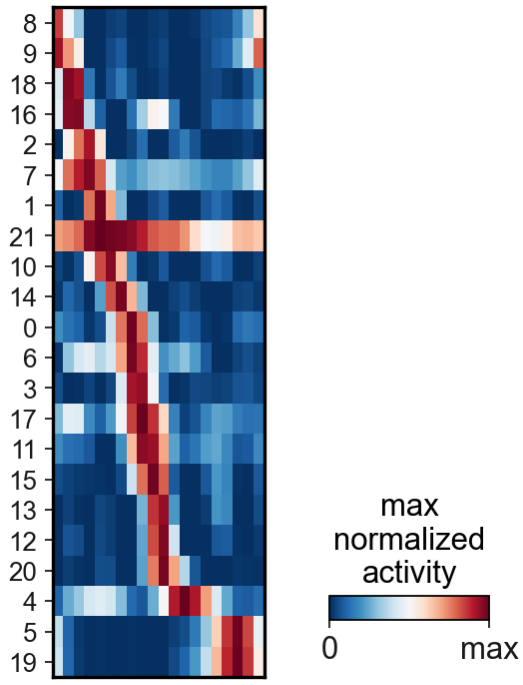
[24]:
```
scf.pl.single_trend(adata,'3',basis='palantir_multiscale',frameon=False,
                    color_exp="k",alpha_expr=.04,ylab='activity')
```



[25]:
```
g=scf.pl.trends(adata,plot_emb=False,highlight_features=[],ordering="max",
                fig_heigth=6,pseudo_cmap=flat_huslmap,ord_thre=.95,return_genes=True)
```



[26]:
```
scf.pl.matrix(adata,features=g,nbins=20,
              cmap="RdBu_r",annot_top=False,
              colorbar_title="max\nnormalized\nactivity")
```


8
9
18
16
2
7
1
21
10
14
0
6
3
17
11
15
13
12
20
4
5
19
max
normalized
activity
0
max

BIBLIOGRAPHY

[Soldatov19] Soldatov *et al.* (2019), *Spatiotemporal structure of cell fate decisions in murine neural crest*, Science.

[Wolf18] Wolf *et al.* (2018), *Scanpy: large-scale single-cell gene expression data analysis*, Genome Biology.

[Setty19] Setty *et al.* (2019), *Characterization of cell fate probabilities in single-cell data with Palantir*, Nature Biotechnology.

[Albergante20] Albergante *et al.* (2019), *Robust and Scalable Learning of Complex Intrinsic Dataset Geometry via ElPiGraph*, Entropy.

[Faure20] Faure *et al.* (2020), *Single cell RNA sequencing identifies early diversity of sensory neurons forming via bi-potential intermediates*, Nature Communications.

[Mao15] Mao *et al.* (2015), *SimplePPT: A simple principal tree algorithm*, SIAM International Conference on Data Mining.

[Bargaje17] Bargaje *et al. (2017)*, *Cell population structure prior to bifurcation predicts efficiency of directed differentiation in human induced pluripotent cells*, PNAS.

[Lange22] Lange *et al. (2022)*, *CellRank for directed single-cell fate mapping*, Nature Methods.

[Kameneva21] Kameneva *et al. (2021) Single-cell transcriptomics of human embryos identifies multiple sympathoblast lineages with potential implications for neuroblastoma origin*, Nature Genetics.

[Ji22] Ji *et al. (2022) A statistical framework for differential pseudotime analysis with multiple single-cell RNA-seq samples*, biorxiv.

PYTHON MODULE INDEX

S

INDEX

U