

Supplementary Materials for scFates

Louis Faure¹, Ruslan Soldatov², Peter V. Kharchenko² & Igor Adameyko¹

¹ Department of Neuroimmunology, Center for Brain Research, Medical University Vienna, 1090 Vienna, Austria

² Department of Biomedical Informatics, Harvard Medical School, Boston, MA 02115, USA.

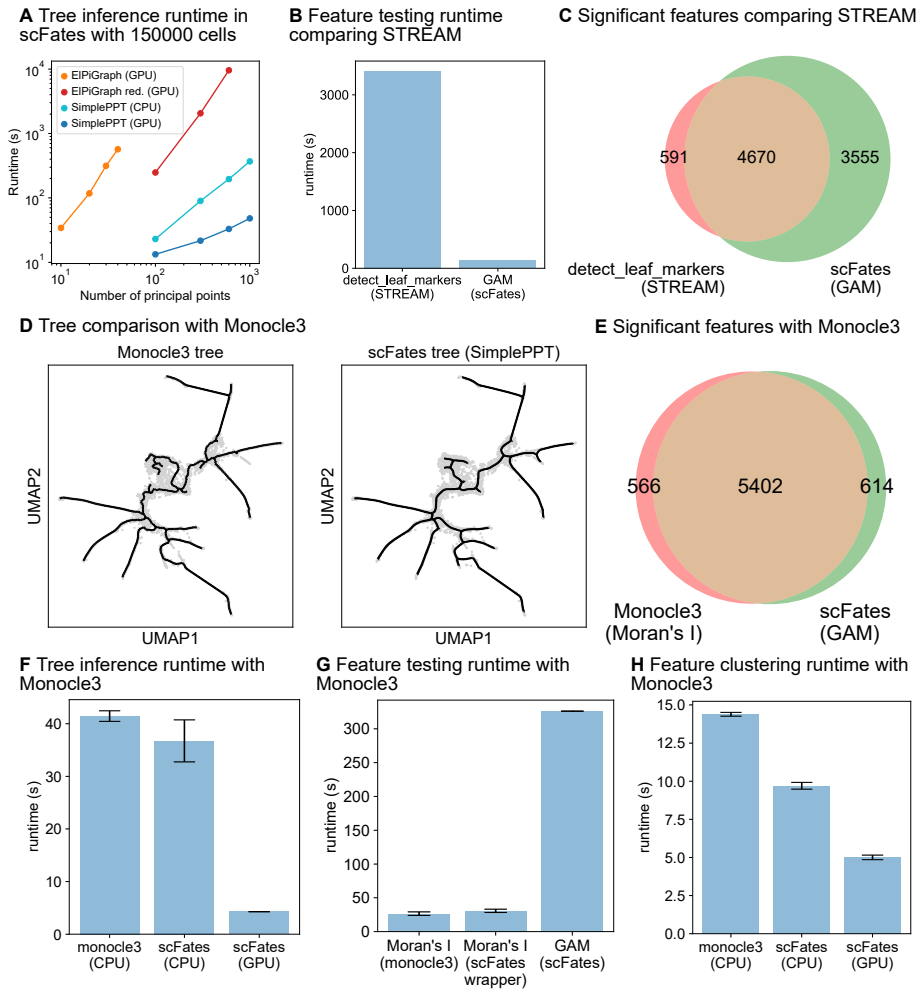


Figure 1: Benchmarks on 88 cores E5-2699v4 CPU & nvidia RTX8000 GPU. **(A)** Runtime measurements between ElPiGraph, Elpigraph reduced (one topology per grammar), SimplePPT using CPU and SimplePPT using GPU on an UMAP embedding of 150000 cells, with varying number of nodes. **(B)** Comparison of feature testing runtime between scFates and STREAM, on a dataset of 3217 cells \times 16934 genes **(C)** Overlap of significant features detected by both methods. **(D)** Comparison of trees generated by Monocle3 and scFates on the UMAP embedding of 6188 cells. **(E)** Overlap of the list of significant features detected by GAM and Moran's I test. **(F)** Runtime for the generation of the tree shown in (D), between monocle3 default tree learning, scFates's SimplePPT (CPU) and SimplePPT (GPU). **(G)** Feature testing runtime comparing Monocle3 Moran's I test, scFates GAM and scFates wrapper of Moran's I test (6188 cells \times 20222 genes). **(H)** Runtime for feature clustering (6188 cells \times 6016 genes) between Monocle3 approach, scFates CPU function (scanpy-based PCA, knn-graph and leiden clustering) and scFates GPU function (cuml-based PCA, scanpy-based knn-graph and cugraph-based leiden clustering).

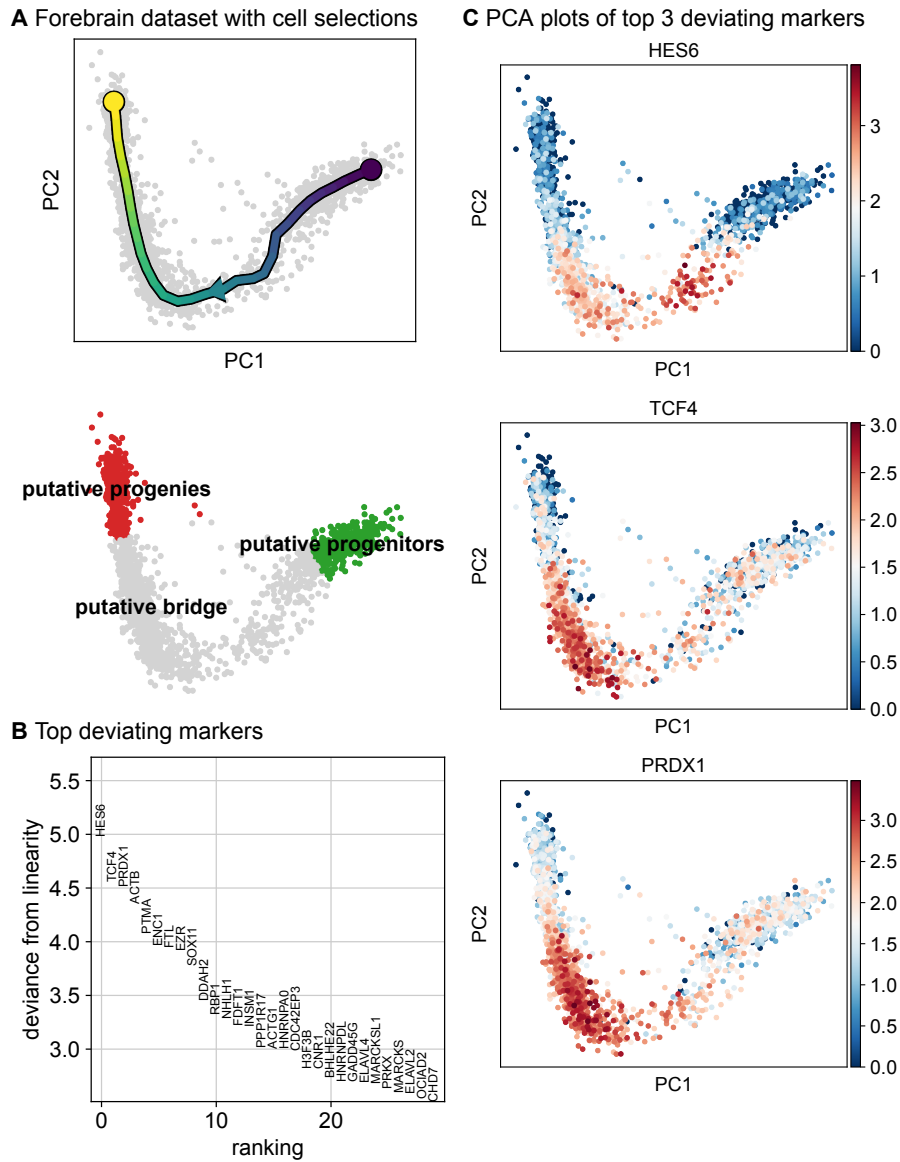


Figure 2: Example of deviation from linearity analysis. **(A)** PCA plot (2st PCs) of published forebrain dataset, containing a transition from radial glia to neurons [1], with overlaid learned curved trajectory (top). Selections of the populations for the test (bottom). **(B)** Top deviating markers, which displays in the cells of putative bridge a higher difference from the linear of mixture of putative progenies and progenitors. **(C)** Top three deviating markers expression shown on PCA plot.

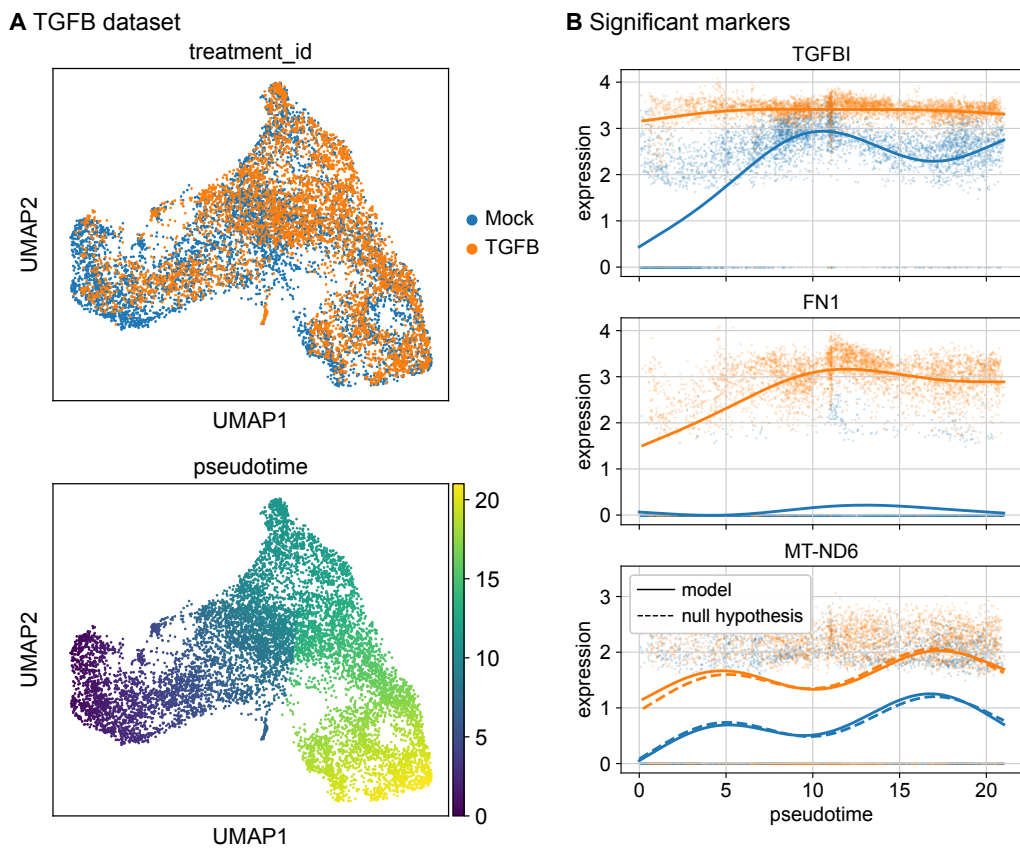


Figure 3: Example of covariate testing on a dataset integrating two experiments, focusing on epithelial to mesenchymal transition. **(A)** UMAP embeddings with the cells colored by treatment (top) and by inferred pseudotime (bottom). **(B)** Selection of markers with amplitude and trend differences (top, center) and amplitude difference only (bottom).

Reconstruction of principal tree with SimplePPT

SimplePPT is available as a separate python package named simpleppt. It consists in fitting a principal tree in M-dimensional space to a set of data points x_1, \dots, x_N . The tree is composed of principal points z_1, \dots, z_K arranged and connected in the same space. Precise description of the underlying mathematics is available in the related paper [2]. The output of the algorithm are three matrices (fig. 4): F, describing the position of the principal points in the same space as the datapoints (dimensions: $K \times M$). B, an adjacency matrix defining the principal tree (dimensions: $K \times K$). R, a soft assignment matrix linking the observed data-points to the principal points (dimensions: $K \times N$).

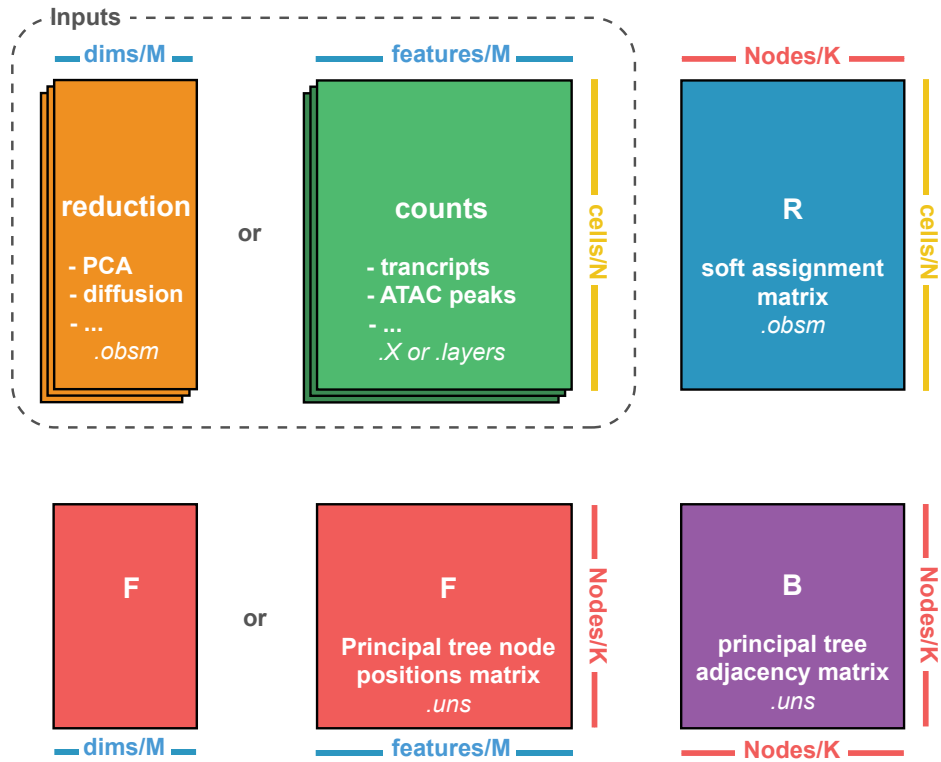


Figure 4: Schematic of the data inputs (in dashed lines) and outputs that scFates uses for tree learning. Text in italic shows the data location in anndata format

Briefly, the algorithm starts with the initialization of a matrix F. These initial positions can be manually given or randomly sampled by choosing K cells as being the initial principal points. F is then fixed, and R and B matrices are solved. B is solved by obtaining the minimum spanning tree from principal point pairwise distances. R is obtained by solving the following constrained optimization problem:

$$\min_{R \in B_r} \sum_{i=1}^N \sum_{k=1}^K r_{ik} (\|x_i + z_k\|^2 + \sigma r_{ik} \log r_{ik}) \quad (1)$$

Which is in other words consisting in computing pairwise distance matrix P between principal points and

data points. P is transformed following a similar step applied in diffusion maps algorithm [3]:

$$R = \exp\left(-\frac{P}{\sigma}\right) \quad (2)$$

The resulting transformation is then row-normalized, leading to a matrix with each data point i being assigned to a principal point k , with the following characteristics:

$$0 > r_{ik} > 1 \quad \text{and} \quad \sum_{k=1}^K r_{ik} = 1 \quad (3)$$

σ in equation (2) is the sigma parameter, which acts as a gaussian kernel to control the locality of the assignments between principal points and data points. For example, a sigma too high will collapse all the principal points into the “middle” of all data points.

Once R and B are calculated, these are fixed to calculate F , by solving the following:

$$F = XR(\lambda L + \Lambda)^{-1} \quad (4)$$

Where L is the Laplacian matrix, calculated by subtracting the diagonal of the degree matrix of B by the matrix B . Λ is the diagonal matrix of the row-sums of R . λ is the parameter lambda which constrains the total length of the tree.

These two main steps are repeated until a defined number of steps or convergence, the latter being defined as the difference between the current F and previous F matrices is below a defined threshold. SimplePPT algorithm can be used by calling the following function:

```
scFates.tl.tree(adata, Nodes, method="ppt", ppt_sigma, ppt_lambda)
```

Reconstruction of principal tree with ELPiGraph

ELPiGraph relies on elastic principal graphs to learn a tree in reduced space. Elastic principal graphs are based on minimization of the energy potential containing three parts:

$$U = MSE + \lambda U_E + \mu U_R \quad (5)$$

MSE is the mean squared error of data approximation, U_E is the sum of squared edge lengths, U_R is a term minimizing the deviation of the principal graph embedding from harmonicity (generalization of linearity). λ , μ are coefficients of regularization. The structure of the graph is generated by application of a sequence of graph transformations, using operations from predefined graph grammar. The simplest

graph grammar "Bisect an edge", "Add a node to a node" leads to construction of a principal tree. A more detailed explanation of the algorithm can be found in the related publication [4]. The output of the algorithm is a principal tree, curve or circle, composed of nodes and a list of edges connecting them together.

To make EIPiGraph output compatible with scFates, the node edge list is converted into the B adjacency matrix, the nodes positions are used as F matrix, and R is generated by assigning cells to their closest principal point, generating a hard assignment matrix, with the following characteristics:

$$r_{ik} \in \{0, 1\} \quad \text{and} \quad \sum_{k=1}^K r_{ik} = 1 \quad (6)$$

EIPiGraph algorithm can be used by calling the following functions:

```
scFates.tl.tree(adata, Nodes, method="epg", epg_lambda, epg_mu)
scFates.tl.curve(adata, Nodes, epg_lambda, epg_mu)
scFates.tl.circle(adata, Nodes, epg_lambda, epg_mu)
```

Reconstruction of principal tree from CellRank reduction with SimplePPT

scFates can construct a principal tree on CellRank output to allow better interpretation of bifurcations. To do so, fate absorption probabilities are combined into a 2D simplex using circular posteriori projection [5]. The calculated CellRank pseudotime is concatenated into the 2D simplex to generate a 3D space, on which SimplePPT will be run. Conversion from CellRank output can be performed by calling the following function:

```
scFates.tl.cellrank_to_tree(adata, time, Nodes)
```

Mapping of cells on the tree

Assigning a pseudotime value to cells requires first to assign that value to principal points. To do so, a root node is selected, either in a manual manner, or according to the value of a measurement (expression of marker, or other cell-wise observation). Pseudotime of a given node is calculated as the distance on the principal tree from the root to that point, in the distance metric used for tree learning. The root is selected with the following function:

```
scFates.tl.root(adata, root)
```

To map a cell onto a given tree, the first closest principal point $pp1$ is selected by using the maximum assignment value from R. The closest principal point to $pp1$, $pp2$, is also selected. The cell is then projected onto the edge connecting $pp1$ and $pp2$, by calculating a weighted average of both principal points distances to the root, using the probabilities given by R. This weighted average is considered as the pseudotime value of the cell (fig. 5 A & B.).

To perform probabilistic mapping of the cells onto the tree, we can take advantage of the fact that the soft assignment matrix R represents natural probabilistic measure as seen in (3) to assign cells onto principal points. The first principal point $pp1$ is sampled according to the probabilities given by R, then the same process is performed as described in the previous paragraph. This approach is not possible if the tree has been calculated using EIPiGraph algorithm, due to the hard assignment matrix it generates (6). Instead, the pseudotime calculation is performed by EIPiGraph. Additionally, a function is proposed to convert the hard assignment matrix into a soft one, by applying one iteration of SimpePPT algorithm using the tree generated by EIPiGraph

Cell mapping assigns to each cell a pseudotime value, a segment and a milestone. The latter, inspired by dynverse [6], highlights the forks and tips. Cell mapping is performed by the following function:

```
scFates.tl.pseudotime(adata)
```

Testing for features associated with the tree

Feature testing is carried out using generalized additive models (GAM) on normalized counts using gaussian errors with 'linear' link [7]. Feature expression is modelled as a function of pseudotime in a branch-specific manner, using cubic spline regression $exp_i \sim t_i$ for each branch independently. This tree-dependent model is then compared with the unconstrained model $exp_i \sim 1$ using F-test. P-values are then corrected for multiple testing, a feature is considered significant by applying a threshold on FDR values (fig. 5 C.). This association test is performed by the following function:

```
scFates.tl.test_association(adata)
```

Fitting, visualisation and clustering of features

Significant features can be fitted to obtain smoothed values along the branches, using the same GAM model as used in the previous section. This is performed using the following function:

```
scFates.tl.fit(adata)
```

Fitted features can be subsequently visualised in different manners, as overview heatmap (fig. 5 D.), single trends (fig. 5 E.) or matrix like plot (fig. 5 F.). Additionally, fitted features can be clustered using the following approach: By default, PCA is performed on the transposed fitted features matrix, a nearest neighbor graph is generated from PC space with a distance metric of choice, then leiden algorithm is run to detect clusters. When run in CPU mode, PCA, nearest neighbor graph and leiden clustering are all performed using scanpy python package. When run in GPU mode, PCA is performed by cuml python package, knn graph is generated by scanpy and leiden clustering is performed using cugraph python package. Clustering can be done using the following function:

```
scFates.tl.cluster(adata)
```

Covariate analysis

Features from curved trajectories or segments from trees containing cells from two or more conditions (e.g. perturbation, disease sample) can be tested for differential expression. First, for each covariates, features are tested for significant change along the pseudotime as explained in the previous section using the following function:

```
scFates.tl.test_association_covariate(adata, group_key)
```

The union of significant feature from each test are then tested for amplitude using the following GAM model:

$$g_i \sim s(\text{pseudotime}) + s(\text{pseudotime}) : \text{Covariate} + \text{Covariate} \quad (7)$$

Where $s(\cdot)$ denotes the penalized regression spline function and $s(\text{pseudotime}) : \text{Covariate}$ denotes interaction between the smoothed pseudotime and covariate terms. From this interaction term, p-values can be extracted and then corrected for multiple testing. Features can also be tested for trend differences, comparing the model described in (7) to the following reduced one:

$$g_i \sim s(\text{pseudotime}) + \text{Covariate} \quad (8)$$

Comparison between the two model is performed with ANOVA and p-values are corrected for multiple testing. The amplitude and trend tests can be performed using the following function:

```
scFates.tl.test_covariate(adata)
```


Linearity deviation test

To find features that specifically characterize the transitions (B) but not the cell population before (A) or after (C) the transitions, a test is performed to check whether feature expression in the transition (B) could be explained by a linear mixture of expressions of that gene in populations A and C. The feature expression profile of each cell in B was modelled as a linear combination of mean gene expression profiles in A and C with the `lm` function in R without the intercept term:

$$lm(B \sim meanA + meanC - 1) \quad (9)$$

For each gene in each cell in bridge B, the magnitude of the residuals not explained by the model is calculated. Mean residuals across all cells in bridge B are calculated and normalized to the standard deviation of the expression of a given gene. The obtained normalized mean residual values can be used to prioritize the genes with distinctive expression patterns in the bridge population. The test can be run with the following function:

```
scFates.tl.linearity_deviation(adata)
```

Bifurcation analysis

Branch specific features are first detected via amplitude testing (`tl.test_fork`) using the following GAM model:

$$g_i \sim s(pseudotime) + s(pseudotime) : Branch + Branch \quad (10)$$

From $s(pseudotime) : Branch$ interaction term, p-values are extracted and then corrected for multiple testing. Then, each significant feature is tested for its upregulation along the path from progenitor to derivative branch, using the linear model $g_i \sim pseudotime$. This first steps are performed using the following function:

```
scFates.tl.test_fork(adata,root_milestone,milestones)
```

Differentially expressed features are then assigned between two post-bifurcation branches with defined differences in expression and FDR cutoffs (fig. 5 D.). This is done via:

```
scFates.tl.branch_specific(adata,root_milestone,milestones,effect)
```

Finally, DE genes can be separated between early and late genes module, according to their pseudotime activation (fig. 5 G.). A gene is considered early if its pseudotime activation is before the pseudotime of the bifurcation. There are two ways to estimate the pseudotime of activation of the DE features. The first approach estimates it by separating the cells into n pseudotime bins (10 bins by default), and by calculating the relative expression rate at a specific bin:

$$r(b_t) = \frac{f(b_{t+1}) - f(b_{t-1})}{\max f - \min f} \quad (11)$$

Where $f(b)$ is the mean fitted expression at a specific bin, if the rate is higher than a defined threshold, the gene is considered to activate at the pseudotime value of the related bin. This approach can be used by calling:

```
scFates.tl.activation(adata,root_milestone,milestones)
```

The second approach uses the linear model $g_i \sim pseudotime$ in the progenitor branch only to estimate if the feature displays an upward trend before the fork. While this method does not output an estimated activation pseudotime, it is a robust alternative to separate early and late genes in more challenging dataset. This approach can be used by calling:

```
scFates.tl.activation_lm(adata,root_milestone,milestones)
```

Local compositional heterogeneity analysis

To study local heterogeneity along a pseudotime trajectory and near a bifurcation, cell composition around a given pseudotime t is approximated by n cells closest to t . To account for cell assignment uncertainty, the probability of a cell being close to a pseudotime point t can also be estimated as a fraction of cases the cell is close to t across a number of sampled cell pseudotime mappings. The related function is:

```
scFates.tl.slide_cells(adata,root_milestone,milestones,win)
```

Local gene-gene correlation reflects coordination of genes around a given pseudotime t , and is defined as a weighted gene-gene Pearson correlation with cells weighted according to the probability of being close to the pseudotime t . Local correlation of a gene g with a module, is assessed as a mean local correlation of that gene with the other genes comprising the module. Similarly, intra-module and inter-module correlations are taken as the mean local gene-gene correlations of all possible gene pairs inside one module, or between the two modules, respectively. The repulsive score, reflecting antagonistic expression

of competing modules at a given developmental stage, is defined as the difference between inter- and intra- module correlations. If genes inside the module correlate much stronger than they do with the competing module, such repulsive score will be close to -1, otherwise it will be closer to 0 (fig. 5 H.). Such calculation can be performed using the following function:

```
scFates.tl.slide_cors(adata,root_milestone,milestones)
```

Such function is also provided with a sliding window of overlapping cells over pseudotime, allowing to extract trends of intra- and inter- module correlations over the trajectory (fig. 5 I.). To ensure that the observed correlations do not arise due to the induction/repression trends of the considered genes, a control model can be employed, where the expression levels of each gene are permuted in windows of n_{perm} cells along the pseudotime axis. The related function for such analysis are:

```
scFates.tl.synchro_path(adata,root_milestone,milestones)
```

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. The approach is based on two assumptions. First, the lineage bias at a given developmental stage is manifested as coordinated expression of a subset of fate-specific genes. Second, if a gene is involved in a coordinated expression with a subset of fate-specific genes at a given developmental stage, then it will continue to be involved at later stages. In other words, the genes are progressively included into a coordinately expressed fate-specific module during differentiation, and are never excluded from the module once they join it. With these assumptions, one can infer the time of gene inclusion into a coordinated fate-specific module using a step-wise indicator function $I_g(t)$ for each fate-specific gene g :

$$I_g(t) = \begin{cases} 0 & \text{if } t < \text{time of gene inclusion} \\ 1 & \text{if } t \geq \text{time of gene inclusion} \end{cases} \quad (12)$$

This approach assesses the correlation of a gene with the time-dependent module at different time points, and identifies an inclusion time of a gene as a time at which such correlation becomes significant. Given some inclusion function $I_g(t)$, the correlation of a gene g at time point t with time-dependent module $M_t = \{g' | I_{g'}(t) = 1\}$ is:

$$cor(g, M_t|t) = \frac{\sum_{g' \in M_t} cor(g, g'|t)}{|M_t|} \quad (13)$$

where g' is a gene from the fate-specific module M_t . $cor(g, M_t|t)$ is approximated by the Pearson correlation among a defined number of cells closest to time point t . To assess the statistical significance of $cor(g, M_t|t)$ expression of each gene can be locally permuted among a defined number of cells from which an empirical p-value can be extracted. As a result, a time-series of p-values $P_g(t)$ is generated for each gene. The time of inclusion is identified as a change-point switching from insignificant to significant p-values. More specifically, a new step function $I_g(t)$ that maximizes probability is used:

$$L = P(P_g(t_1), \dots, P_g(t_n)|I_g(\cdot)) = \prod_i P(P_g(t_i)|I_g(t_i)) \quad (14)$$

Where $P_g(t_i|1) = \frac{\alpha}{1-e^{-\alpha}}e^{-\alpha P_g(t_i)}$, $P_g(t_i|0) = 1$, and α is a parameter that regulates the stringency of the desired significance level. The log likelihood can be rewritten as:

$$\log L = \sum_{t_i \geq \text{time of inclusion}} \left[-\alpha P_g(t_i) + \log \frac{\alpha}{1 - e^{-\alpha}} \right] \quad (15)$$

Iterative optimization is used to identify functions $\{I_g(\cdot)\}$, initializing them as $I_g(\cdot) = 1$. At each step, given $\{I_g(\cdot)\}$, for each gene g we calculated $cor(g, M_t|t)$, then estimated $P_g(t)$, and then updated $I_g(t)$. In practice, the method converges in less than 10 iterations. All steps described in that section are performed by the following single function (fig. 5 J):

```
scFates.tl.module_inclusion(adata,root_milestone,milestones)
```

Data reproducibility

The whole figure 5 can be fully reproduced by following the related notebook in the documentation: https://scfates.readthedocs.io/en/latest/Tree_Analysis_Bone_marrow_fates.html. Benchmarks with STREAM and Monocle3 are included in the following github repository: https://github.com/LouisFaure/scFates_notebooks/tree/main/benchmarks.

References

- [1] Gioele La Manno et al. “RNA velocity of single cells”. In: *Nature* 2018 560:7719–560 (7719 Aug. 2018), pp. 494–498.
- [2] Qi Mao et al. “SimplePPT: A simple principal tree algorithm”. In: 2015.
- [3] Ronald R. Coifman and Stéphane Lafon. “Diffusion maps”. In: *Applied and Computational Harmonic Analysis* 21 (1 July 2006), pp. 5–30.

- [4] Luca Albergante et al. “Robust and scalable learning of complex intrinsic dataset geometry via ElPiGraph”. In: *Entropy* 22 (3 Mar. 2020), p. 296.
- [5] Lars Velten et al. “Human haematopoietic stem cell lineage commitment is a continuous process”. In: *Nature Cell Biology* 2017 19:4 19 (4 Mar. 2017), pp. 271–281.
- [6] Wouter Saelens et al. “A comparison of single-cell trajectory inference methods”. In: *Nature biotechnology* 37.5 (2019), pp. 547–554.
- [7] Trevor Hastie and Robert Tibshirani. *Generalized additive models*. Wiley Online Library, 1990.
- [8] Manu Setty et al. “Characterization of cell fate probabilities in single-cell data with Palantir”. In: *Nature Biotechnology* 37 (4 2019), pp. 451–460.

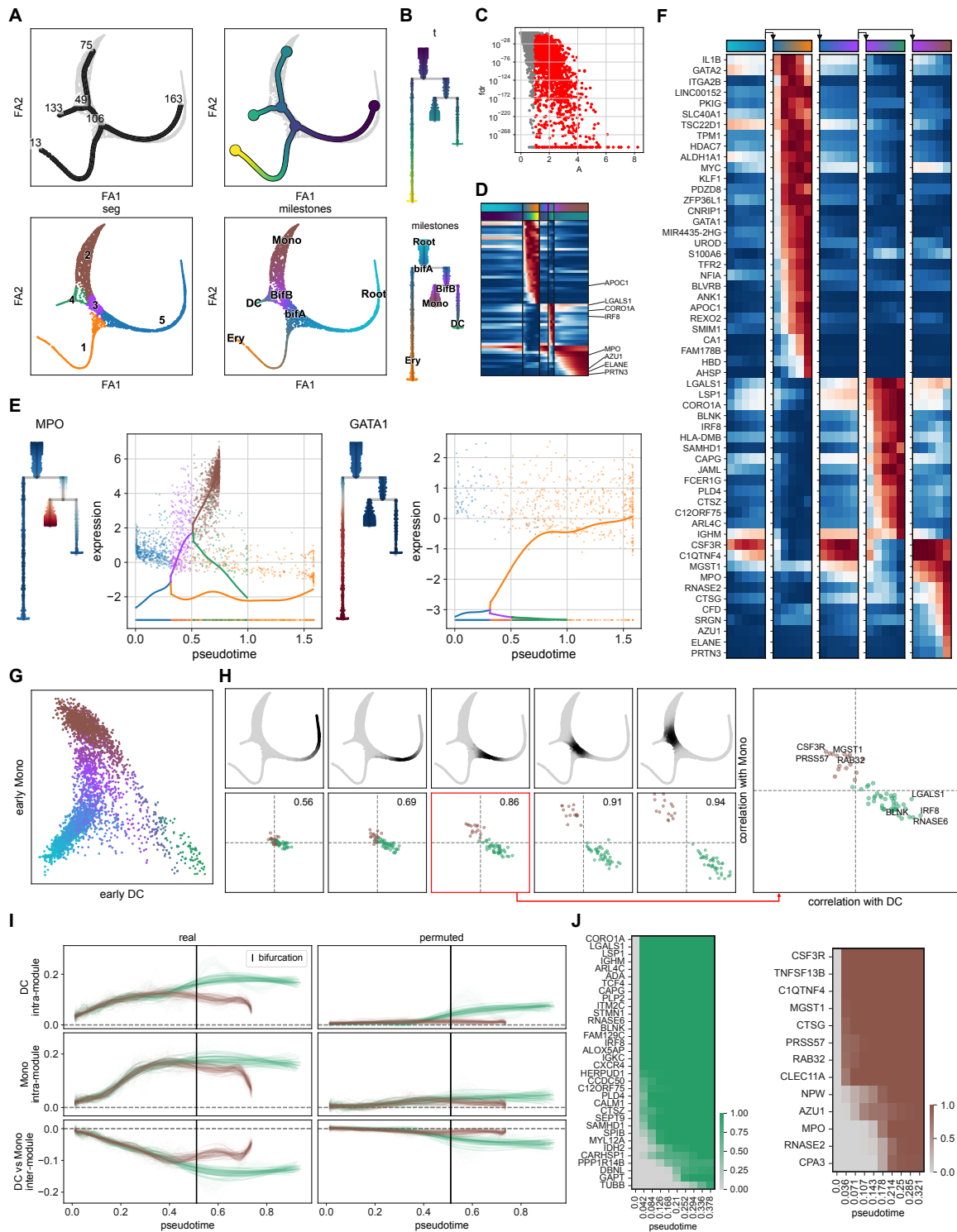


Figure 5: scFates analysis plot examples on bone marrow dataset used in Palantir study [8]. All panels have been assembled without any modification from python output: **A**) Representation of the tree before (top left) and after (top right) root selection and pseudotime calculation. Cells are assigned a segment (lower left) and a milestone (lower right). **B**) Dendrogram representation of the tree. **C**) Detection of significantly associated genes with the the tree. **D**) Top DE branch specific marker expressions showed as heatmap of cells aligned with pseudotime. **E**) Single trend plot from two known markers. **F**) Same data as shown in D, this time as a matrix plot for summarized trends over bins of pseudotime, per segment. **G**) Early gene module mean expression over cells for a bifurcating trajectory. **H**) Correlation and repulsion analysis of two competing early gene module along non-intersecting windows of cells. **I**) intra- and inter-module correlation analysis over sliding window of cells. **J**) module inclusion analysis.