A comparison of single-cell trajectory inference methods: towards more accurate and robust tools

A comparison of single-cell trajectory inference methods: towards more accurate and robust tools

01 February, 2018

Wouter Saelens* [1] [2], Robrecht Cannoodt* [1] [2] [3], Helena Todorov[1] [2] 4 5, Yvan

Saeys[1 2]

* Equal contribution
[1] Data mining and Modelling for Biomedicine, VIB Center for Inflammation Research, Ghent, Belgium.
[2] Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Ghent, Belgium.
[3] Center for Medical Genetics, Ghent University Hospital, Ghent, Belgium.
4 CIRI, Centre International de Recherche en Infectiologie, Inserm, Lyon, France.
5 Université Claude Bernard Lyon 1, Lyon, France.

Abstract

Using single-cell transcriptomics data, it is now possible to computationally order cells along trajectories, allowing the unbiased transcriptome-wide study of cellular dynamic processes. Since 2014, more than 50 trajectory inference methods have been developed, each with its own set of methodological characteristics. This makes a direct comparison between trajectory inference methods often difficult, and as a result a comprehensive assessment of the performance and robustness of each method is still lacking. We developed a framework of data structures for trajectory inference methods in order to make their results comparable to each other and to a gold standard, when available. In this work, we compare the results from a total of 22 trajectory inference methods, on a large collection of real datasets and realistic synthetic datasets. We employ a novel parameter optimisation procedure, and We compare methods using several metrics, including accuracy of the inferred ordering, correctness of the network wiring, code quality and user friendliness. We conclude with practical guidelines for method users, and also believe our evaluation framework can be used to spearhead the development for new methods. Our evaluation pipeline is fully reproducible, can be easily extended, and is available at https://www.github.com/dynverse.

Introduction

Single-cell -omics technologies now make it possible to more accurately model biological systems than ever before. One area where single-cell data can be particularly useful is in the study of cellular dynamic processes, such as the cell cycle, cell differentiation and cell activation. When cells are sampled from a population in which cells are at different unknown points in the dynamic process, trajectory inference methods can be used to computationally order these cells along their dynamic process[1]. Because they offer an unbiased and transcriptome-wide understanding of the dynamic process, trajectories then allow the identification of

new (primed) subsets of cells, delineation the exact wiring of a differentiation tree and inference of regulatory interaction responsible for a bifurcation refs.

Dozens of trajectory inference methods have been developed over the last years, and more are being published almost every month (**Supplementary Figure 1a**). Initially, most trajectory inference fixed the trajectory structure to a linear or bifurcating structure, by either algorithmic limitations or through user parameters, and focussed on correctly ordering the cells along this fixed structure. However, several recent methods now infer both the ordering and the structure at the same time, although certain structural limitations are still imposed(**Supplementary Figure 1b**). Methods which do not impose limitations on the structure are now being developed[2,3], and it is expected that in the future methods will be able to model even more complex behavior, such as multiple dynamic processes happening at parallel in a single-cell or the integration of datasets from different patients[1,4].

Although trajectory inference methods use a variety of algorithms and subcomponents to reach a final ordering, most consist of the following three steps: (i) preprocessing (normalization, filtering of genes and/or cells), (ii) conversion to a simplified representation using dimensionality reduction, clustering or graph building and (iii) ordering the cells along the simplified representation[1]. Depending on the way the cells are ordered, some prior information about the dynamic process can be optionally used or required by the method, which can both bias the trajectory to current knowledge but also guide the method towards choosing the right trajectory among many others. Furthermore, with the ongoing strong scalability in single-cell -omics technologies, it becomes more and more important that analysis methods become scalable and user friendly .

Given this plethora of available trajectory inference methods, it is important that methods are compared, so that users can use the most optimal method available for their problem. Moreover, new methods need to be rigorously tested, so that development can focus on improving the current state-of-the-art. In this study, we therefore for the first time developed a comprehensive evaluation framework for trajectory inference methods. We test both on synthetic data, for which the gold standard is known, and real data, for which in some cases the gold standard can be extracted from expert knowledge. We include a rigorous parameter optimisation, making sure that parameters are optimized while avoiding overfitting on characteristics of specific datasets. To make our framework reusable and extendable, we make use of continuous analysis[5], which allows the developers of new methods to easily test their methods and compare them against the state-of-the-art.

Results

Method quality control

We gathered a list of 51 TI methods from literature, and selected a subset of 24 for evaluation, primarily based on their free availability and the presence of a programming interface **Table 1**. We characterized all methods in four different ways: possible trajectory structures, implementation quality, prior information and the underlying algorithm (**Figure 1**).

Table 1 Overview of current trajectory inference methods, and whether they were included in this current evaluation study

Name

Date

Maximal trajectory type

Evaluated

Monocle 1

01/04/2014

Unrooted tree

Wanderlust

24/04/2014

Undirected linear

SCUBA

30/12/2014

Unrooted tree

sincell

27/01/2015

Unrooted tree

NBOR

08/06/2015

Undirected linear

3

Waterfall
03/09/2015
Undirected linear

gpseudotime
15/09/2015
Undirected linear
3
embeddr
18/09/2015
Undirected linear

ECLAIR
12/01/2016
Unrooted tree
6
DPT
08/02/2016
Simple fork

pseudogp
05/04/2016
Undirected linear

SLICER
09/04/2016
Undirected graph

SCell
19/04/2016
Undirected linear
5

Wishbone
02/05/2016
Simple fork

TSCAN
13/05/2016
Unrooted tree

SCOUP
08/06/2016
Complex fork

DeLorean
17/06/2016
Undirected linear
7
StemID
21/06/2016
Unrooted tree

Ouija
23/06/2016
Undirected linear

Mpath
30/06/2016
Unrooted tree

cellTree
13/08/2016
Unrooted tree

WaveCrest
17/08/2016
Undirected linear
6
SCIMITAR
04/10/2016
Undirected linear
6
SCORPIUS
07/10/2016
Undirected linear

SCENT
30/10/2016
Undirected linear
4
k-branches
15/12/2016
Unrooted tree
8
SLICE
19/12/2016
Unrooted tree

topslam
13/02/2017
Undirected linear

Monocle 2
21/02/2017
Unrooted tree

Granatum
22/02/2017
Unrooted tree
5
GPfates
03/03/2017
Complex fork

MFA
15/03/2017
Complex fork

TASIC
04/04/2017
Unrooted tree
6
slingshot
19/04/2017
Unrooted tree

scTDA
01/05/2017
Undirected linear
6
UNCURL
31/05/2017
Undirected linear
9
reCAT
19/06/2017
Undirected cycle
9

MATCHER
24/06/2017
Undirected linear
9
PhenoPath
06/07/2017
Undirected linear

HopLand
12/07/2017
Undirected linear
9
SoptSC
26/07/2017
Undirected linear
1 9
PBA
30/07/2017
Complex fork
9
BGP
01/08/2017
Simple fork
9
scanpy
09/08/2017
Simple fork
9
WADDINGTON-OT
27/09/2017
Undirected graph
2 9

AGA

27/10/2017

Disconnected undirected graph

9

p-Creode

15/11/2017

Unrooted tree

9

iCpSc

30/11/2017

Undirected linear

4 9

GrandPrix

03/12/2017

Complex fork

9

Topographer

21/01/2018

Rooted tree

9

CALISTA

31/01/2018

Undirected graph

9

[1] Not free

[2] Unavailable

[3] Superseded by another method

[4] Requires data types other than expression

[5] No programming interface

[6] Unresolved errors during wrapping

[7] Too slow (requires more than one hour on a 100x100 dataset)

[8] Doesn't return an ordering

[9] Published later than 2017-05-01 to be included in the current version of the evaluation

At the level of trajectory structures, most recent methods can handle one or more splits (either bifurcations or multifurcations) in the trajectory (**Figure 1a**). However, only a handful of more recent methods can currently handle more complex graph structures which can include loops (**Supplementary Figure 1b**), one of which is included in the current evaluation (SLICER).

While not directly related to the accuracy of the inferred trajectory, the quality of the implementation is an important first evaluation metric, because good unit testing makes sure the implementation is correct, good documentation makes it easier for potential users to apply the method on their data, and overall good code quality makes it possible for other developers to adapt the method and extend it further. We therefore looked at the implementation of each method, and assessed their quality using a transparent scoring scheme (**Supplementary Table 1**). The individual quality checks can be grouped in two ways: what aspect of the method they investigate (availability, code quality, code assurance, documentation, behaviour and the paper) or for which purpose(s) they are important (user friendliness, developer friendliness or good science). These categorisations can then be used to help current developers to improve their tool, and to guide the selection of users and developers to use these tools for their purpose. After publishing this preprint, we will contact the authors of each method, allowing them to improve their method before the final publishing of the evaluation.

Most methods fulfilling most of the basic criteria, such as free availability and basic code quality criteria (**Figure 1b**). Recent methods tend to have higher code quality (**Supplementary Figure 2**). However, several aspects are consistently lacking for the majority of the methods (**Figure 1e**) and we believe that these should receive extra attention from developers. Although these outstanding issues cover all five categories, code assurance and documentation in particular are problematic areas (**Supplementary Figure 3** and **Figure 1c**), notwithstanding studies pinpointing these as good practices[6].

The underlying algorithms can be divided into several modules, and these frequently contain soms clustering, dimensionality reduction and/or graph building steps. Interestingly, modules are frequently shared between different algorithms

(**Supplementary Figure 4**).

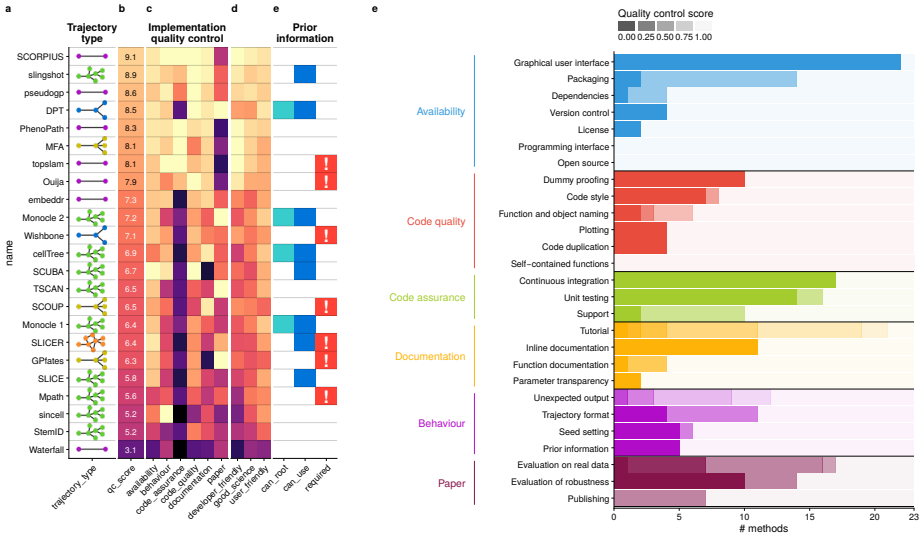


Figure 1 First characterization of each method.

Evaluation overview

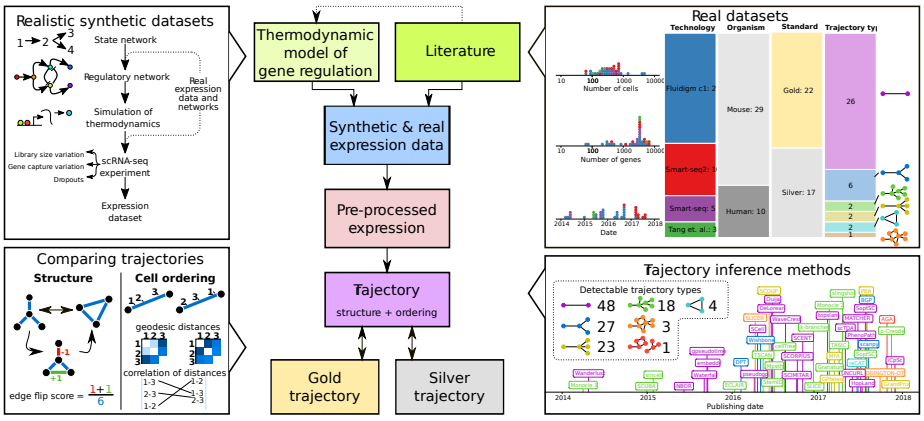


Figure 2 Overview of the evaluation

Our evaluation was structured in three main phases. In the first phase, we used toy data as a positive control for each method, assessing whether it can detect the correct trajectories in very simplistic data. Next, we generated synthetic data from a model of gene regulation, combined with a simulation of the technical variation introduced by single-cell RNA sequencing. We used this synthetic data in a parameter optimization procedure, where we used cross-validation between datasets to avoid overfitting on particular data set structures. Finally, in the third phase, we used the optimized parameters to score the performance of

each method on real data. We split the real datasets in two groups based on the type of reference trajectory available. The reference in datasets with a silver standard were defined by the author's based on the expression data itself, while the reference for gold standard was defined using external information, such as fluorescent sorting or time series.[a]

Overall method comparison

Evaluation on synthetic data

Initial evaluation.. Robustness.. Gegenereerde synthetische data. Parameter tuning voor echte data + initiële evaluatie (tot wat is de methode in staat) + robustness etc testing

Evaluation on real data

Biological relevance.. Echte data. Evaluatie (biologische relevantie van de methode)

Discussion

Methods

Trajectory types

We

Trajectory inference methods

We gathered a list of 51 trajectory inference methods **Table 1**, based on two existing lists found online[7,8] and by searching in literature for "trajectory inference" and "pseudotemporal ordering". We believe this to be the most complete list to data, but are welcome to include additional methods through e-mail or an issue on our github (https://github.com/dynverse/dynalysis). Methods were excluded from the evaluation based on several criteria: (1) Not free, (2) Unavailable, (3) Superseded by another method, (4) Requires data types other than expression, (5) No programming interface, (6) Unresolved errors during wrapping, (7) Too slow (requires more than one hour on a 100x100 dataset), (8) Doesn't return an ordering, (9) Published later than 2017-05-01 to be included in the current version of the evaluation. This resulted in the inclusion of 24 methods in the evaluation (**Table 1**).

We wrapped each method (available at https://www.github.com/dynverse/dynmethods) and postprocessed its output into a common probabilistic trajectory model. This model consists of two parts. The milestone network represents the overall network structure, and contains of edges between different milestones and the length of the edge between them. The milestone percentages contain for each cell its position between two or more milestones, and sums for each cell to one. ## Method quality control

We created a transparent scoring scheme to check the quality of each method (**Table 1**). The quality control assessed 6 categories, each looking at several aspects, which are further divided into individual items. The goal of this quality control is in the first place to stimulate the improvement of current methods, and the development of user and developer friendly new methods. The availability category checks whether the method is easily available, whether the code and dependencies can be easily installed, and how the method can be used. The code quality assesses the quality of the code both from a user perspective (function naming, dummy proofing and availability of plotting functions) and a developer perspective (consistent style and code duplication). The code assurance category is frequently overlooked, and checks for code testing, continuous integration[5] and an active support system. The documentation category checks the quality of the documentation, both externally (tutorials and function documentation) and internally (inline documentation). The behaviour category assesses the ease by which the method can be run, by looking for unexpected output files and messages, prior information and how easy the trajectory model can be extracted from the output. Finally, we also assessed the certain aspects of the study in which the method was proposed, such as publication in a peer-reviewed journal, the number of dataset on which the usefulness of the method was shown, and the scope of method evaluation in the paper.

For each quality aspect received a weight by how frequently it was found in several papers and online sources which discuss tool quality (**Table 1**). This was to prevent make sure more important aspect, such as the open source availability of the method, to outweigh other aspects, such as the availability of a graphical user interface. For calculating the final score, we however weighed each of the six categories equally.

Real datasets

We gathered a list of real datasets by searching for "single-cell" at the Gene Expression Omnibus and selecting those datasets in which the cells are sampled from different stages in a dynamic process (**Supplementary Table 2**). The scripts to download and process these datasets are all available in our repository (http://www.github.com/dynverse/dynalysis). When available, we preferred to start from the raw counts data. These raw counts were all normalised and filtered using a common pipeline, discussed later. We determined a reference standard for every dataset using labelling provided by the author's, and clas-

sified the standards into gold and silver based on whether this labelling was determined by the expert using the expression (silver standard) or using other external information (such as FACS or the origin of the sample, gold standard) (**Supplementary Table 2**).

Synthetic datasets

Our workflow to generate synthetic data is based on the well established workflow used in the evaluation of network inference methods[9,10] and consists of four main steps: network generation, simulation, gold standard extraction and simulation of the scRNA-seq experiment. At every step, we took great care was taking to mimic real cellular regulatory networks as best as possible, while keeping the model simple and easily extendable. For every synthetic dataset, we used a random real dataset as a reference dataset (from those described earlier), making sure the number of variable genes and cells were similar.

Network generation

One of the main process involved in cellular dynamic processes is through gene regulation, where regulatory cascades and feedback loops lead to progressive changes in expression and decision making. The exact way a cell chooses a certain path during its differentiation is still an active research field, although certain models have already emerged and been tested in vivo. One driver of bifurcations seem to be mutual antagonism, where genes[11] strongly repress each other, forcing one of the two to become inactive[12]. Such mutual antagonism can be modelled and simulated[13,14]. Although such a two-gene model is simple and elegant, the reality is frequently more complex, with multiple genes (grouped into models) repressing each other[15].

Cascades? Cycle?

To simulate certain trajectory structures, we therefore designed module networks in which the cells will follow a particular trajectory structure given certain parameters (**Supplementary Figure 5**). Two module networks generated linear trajectories (linear and linear long), two generated simple forks (bifurcating and converging), one generated a complex fork (trifurcating), one generated a rooted tree (consecutive bifurcating) and two generated trajectories which simple undirected graphs (bifurcating loop and bifurcating convergence).

From these module networks we generated gene regulatory networks in two steps: generation of the main regulatory network, and addition of extra target genes from real regulatory networks. For each dataset, we used the same number of genes as were differentially expressed in the real datasets. 5% of genes were assigned to be part of the main, and were randomly distributed among all modules (with at least one gene per module). We sampled edges between

these individual genes (according to the module network) using an uniform distribution between 1 and the number of possible targets in each module. To add additional target genes to the network, we assigned every regulator from the network to a real regulator in a real network (from regulatory circuits[16]), and extracted for every regulator a local network around it using personalized pagerank (with damping factor set to 0.1).

Simulation of thermodynamics

To simulate the gene regulatory network, we used a system of differential equations similar to which has been standard in evaluations of network inference[10]. In this model, changes in gene expression (\(x_i\)) and protein expression (\(y_i\)) are modeled using ordinary differential equations[9] (ODEs):

\[\begin{align} \label{eq:mrna_ode} \frac{dx_i}{dt} &= \underbrace{m \times f(y_1, y_2, ...)}_\text{production} - \underbrace{\lambda \times x_i}_\text{degradation} \\ \label{eq:prot_ode} \frac{dy_i}{dt} &= \underbrace{r \times x_i}_\text{production} - \underbrace{\Lambda \times y_i}_\text{degradation} \end{align} \]

where \(m\), \(\lambda\), \(r\) and \(\Lambda\) represent production and degradation rates, the ratio of which determines the maximal gene and protein expression. The two types of equations are coupled because the production of protein \(y_i\) depends on the amount of gene expression \(x_i\), which in turn depends on the amount of other proteins through the activation function \(f(y_1, y_2, ...)\).

The activation function is inspired by a thermodynamic model of gene regulation, in which the promoter of a gene can be bound or unbound by a set of transcription factors, each representing a certain state of the promoter. Each state is linked with a relative activation \(\alpha_j\), a number between 0 and 1 representing the activity of the promoter at this particular state. The production rate of the gene is calculated by combining the probabilities of the promoter being in each state with the relative activation:

\[\begin{align} \label{eq:input_funcion_probabilities} f(y_1, y_2, ..., y_n) = \sum_{j \in \{0, 1, ..., n^2\}} \alpha_j \times P_j \end{align} \]

The probability of being in a state is based on the thermodynamics of transcription factor binding. When only one transcription factor is bound in a state: \[\begin{align} P_j \propto \nu = \left(\frac{y}{k}\right)^{n} \end{align} \]

Where the hill coefficient \(n\) represents the cooperativity of binding and \(k\) the transcription factor concentration at half-maximal binding. When multiple regulators are bound: \[\begin{align} P_j \propto \nu = \rho \times \prod_j \left(\frac{y_j}{k_j}\right)^{n_j} \end{align} \]

where \(\rho\) represents the cooperativity of binding between the different transcription factors.

\(P_i\) is only proportional to \(\nu\) because \(\nu\) is normalized such that \(\sum_{i} P_i = 1\).

To each differential equation, we added an additional stochastic term: \[\begin{align} \frac{dx_i}{dt} = m \times f(y_1, y_2, ...) - \lambda \times x_i &+ \eta \times \sqrt{x_i} \times \Delta W_t \\ \frac{dy_i}{dt} = r \times x_i - \Lambda \times y_i &+ \eta \times \sqrt{y_i} \times \Delta W_t \end{align} \]

with \(\Delta W_t \sim \mathcal{N}(0, h)\).

Similar to[9], we sample the different parameters from random distributions **Supplementary Table 3**.

These SDEs were simulated using the Euler–Maruyama approximation, with time-step \(h = 0.01\) and noise strength \(\eta = 8\). The total simulation time varied between 5 for linear and bifurcating datasets, 10 for consecutive bifurcating, trifurcating and converging datasets, 15 for bifurcating converging datasets and 30 for linear long, cycle and bifurcating loop datasets. The burn-in period was for each simulation 2. Each network was simulated 32 times.

Simulation of the single-cell RNA-seq experiment

For each dataset we sampled the same number of cells as were present in the reference real dataset, limited to the simulation steps after burn-in. Next, we used the Splatter package[17] to estimate the different characteristics of a real dataset, such as the distributions of average gene expression, library sizes and dropout probabilities. We used Splatter to simulate the expression levels \(\lambda_{i,j}\) of housekeeping genes \(i\) (to match the number of genes in the reference dataset) in every cell \(j\). These were combined with the expression levels of the genes simulated within a trajectory. Next, true counts were simulated using \(Y'_{i,j} \sim \text{Poi}(\lambda_{i,j})\). Finally, we simulated dropouts by setting true counts to zero by sampling from a Bernouilli distribution using a dropout probability \(\pi^D_{i,j} =\frac{1}{1+e^{-k(\text{ln}(\lambda_{i,j})-x_0)}}\).

Gold standard extraction

Because each cellular simulations follows the trajectory it its own speed, knowing the exact position of a cell within the trajectory structure is not straightforward. Furthermore, the speed by which simulated cells made a decision between two or more alternative paths was highly variable sfig?. To estimate a cell's position during a simulation within the trajectory structure, we therefore used the known progression of the modules, given in **Supplementary Figure 5c**, as a backbone. We smoothed the expression in each simulation using a rolling mean with a window of 50 time steps, and then calculated the average module expression along the simulation. We used dynamic time warping, implemented in the dtw

R package[18,19], with an open end to align a simulation to all possible module progressions, and then picked the alignment which minimised the normalised distance between the simulation and the backbone. In case of cyclical trajectory structures, the number of possible milestones a backbone could progress through was limited to 20.

Common normalisation pipeline

We used a standard single-cell RNA-seq preprocessing pipeline which uses the scran and scater Bioconductor packages[20]. The advantages of this pipeline is that it works both with and without spike-ins, and includes a harsh cell filtering which looks at abnormalities in library sizes, mitochondrial gene expression, and number of genes expressed using median absolute deviations (set to 3). We required that a gene has to be expressed in at least 5% of the cells, and that it should have an average expression higher than 0.05. Furthermore, we used the pipeline to select the most highly variable genes, using a false discovery rate of 5% and a biological component higher than 0.5. As a final filter, we removed both all zero genes and all zero cells until convergence.

Evaluation metrics

We used two different metrics to assess the performance of each TI method, one looking at the accuracy of the cellular ordering, and one looking at the accuracy of the detected trajectory structure.

Acknowledgements

References

1. Cannoodt, R., Saelens, W. & Saeys, Y. Computational methods for trajectory inference from single-cell transcriptomics. *European Journal of Immunology* **46,** 2496–2506 (2016).

2. Welch, J. D., Hartemink, A. J. & Prins, J. F. SLICER: Inferring branched, nonlinear cellular trajectories from single cell RNA-seq data. *Genome Biology* **17,** 106 (2016).

3. Wolf, F. A. *et al.* Graph abstraction reconciles clustering with trajectory inference through a topology preserving map of single cells. *bioRxiv* 208819 (2017). doi:10.1101/208819

4. Tanay, A. & Regev, A. Scaling single-cell genomics from phenomenology to mechanism. *Nature* **541,** nature21350 (2017).

5. Beaulieu-Jones, B. K. & Greene, C. S. Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology* **35,** nbt.3780 (2017).

6. Wilson, G. *et al.* Best Practices for Scientific Computing. *PLOS Biology* **12,** e1001745 (2014).

7. Davis, S. Awesome-single-cell: List of software packages for single-cell data analysis, including RNA-seq, ATAC-seq, etc. (2018).

8. Gitter, A. Single-cell-pseudotime: An overview of algorithms for estimating pseudotime in single-cell RNA-seq data. (2018).

9. Schaffter, T., Marbach, D. & Floreano, D. GeneNetWeaver: In silico benchmark generation and performance profiling of network inference methods. *Bioinformatics (Oxford, England)* **27,** 2263–2270 (2011).

10. Marbach, D. *et al.* Wisdom of crowds for robust gene network inference. *Nature methods* **9,** 796–804 (2012).

11. Xu, H. *et al.* Regulation of bifurcating B cell trajectories by mutual antagonism between transcription factors IRF4 and IRF8. *Nature Immunology* **16,** 1274–1281 (2015).

12. Graf, T. & Enver, T. Forcing cells to change lineages. *Nature* **462,** 587 (2009).

13. Wang, J., Zhang, K., Xu, L. & Wang, E. Quantifying the Waddington landscape and biological paths for development and differentiation. *Proceedings of the National Academy of Sciences* **108,** 8257–8262 (2011).

14. Ferrell, J. E. Bistability, Bifurcations, and Waddington's Epigenetic Landscape. *Current Biology* **22,** R458–R466 (2012).

15. Yosef, N. *et al.* Dynamic regulatory network controlling Th17 cell differentiation. *Nature* **496,** 461–468 (2013).

16. Marbach, D. *et al.* Tissue-specific regulatory circuits reveal variable modular perturbations across complex diseases. *Nature Methods* **13,** 366 (2016).

17. Zappia, L., Phipson, B. & Oshlack, A. Splatter: Simulation of single-cell RNA sequencing data. *Genome Biology* **18,** 174 (2017).

18. Giorgino, T. Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package | Giorgino | Journal of Statistical Software. *Journal of Statistical Software* **31,** (2009).

19. Tormene, P., Giorgino, T., Quaglini, S. & Stefanelli, M. Matching incomplete time series with dynamic time warping: An algorithm and an application to post-stroke rehabilitation. *Artificial Intelligence in Medicine* **45,** 11–34 (2009).

20. Lun, A. T., McCarthy, D. J. & Marioni, J. C. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Research* **5,** 2122 (2016).

21. Lee, J. Rpackages: R package development - the Leek group way! (2017).

22. Taschuk, M. & Wilson, G. Ten simple rules for making research software more robust. *PLOS Computational Biology* **13,** e1005412 (2017).

23. Wickham, H. *R Packages: Organize, Test, Document, and Share Your Code.* ("O'Reilly Media, Inc.", 2015).

24. Artaza, H. *et al.* Top 10 metrics for life science software good practices. *F1000Research* **5,** 2000 (2016).

25. Silva, L. B., Jimenez, R. C., Blomberg, N. & Luis Oliveira, J. General guidelines for biomedical software development. *F1000Research* **6,** (2017).

26. Jiménez, R. C. *et al.* Four simple recommendations to encourage best practices in research software. *F1000Research* **6,** (2017).

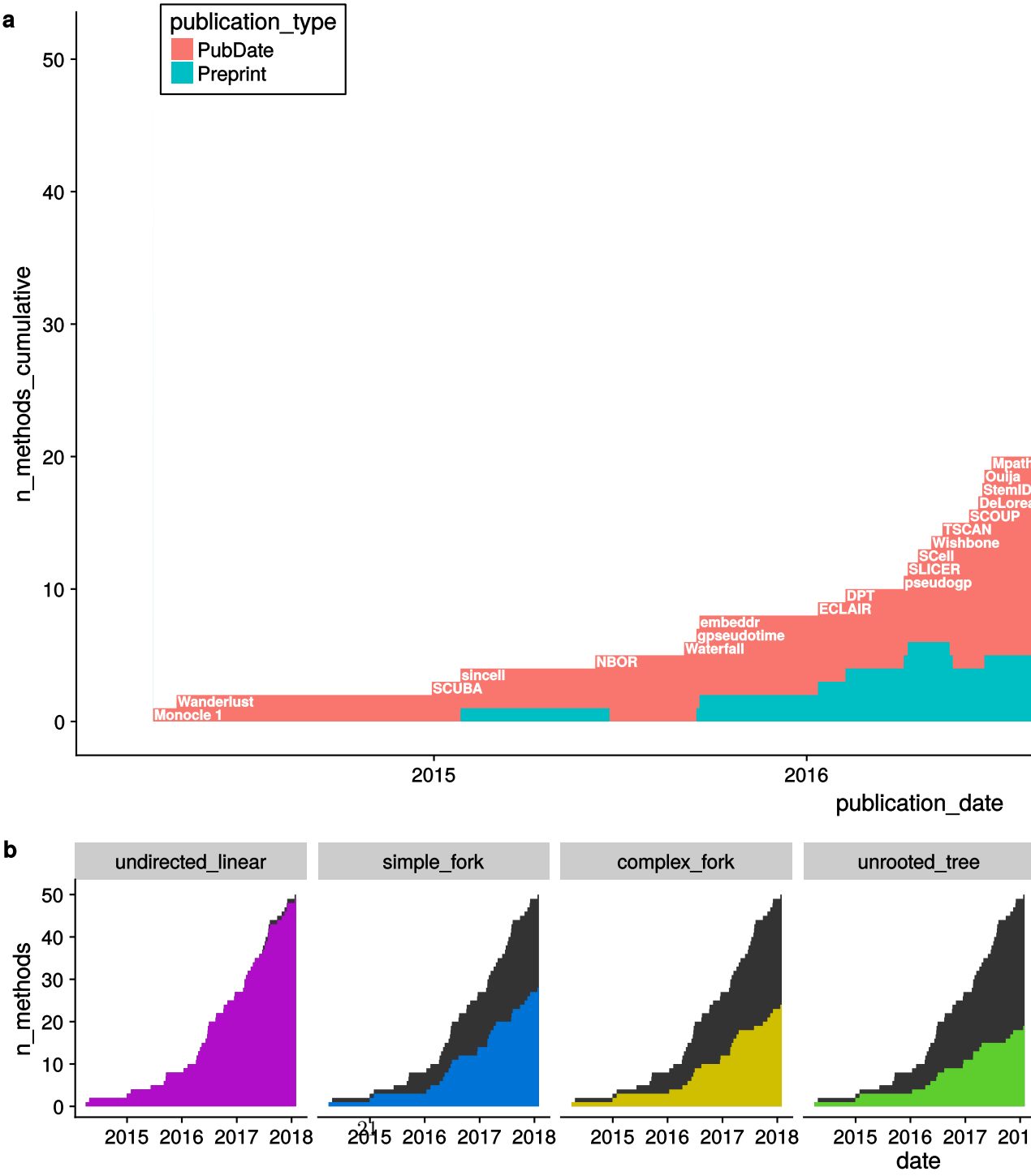
27. Karimzadeh, M. & Hoffman, M. M. Top considerations for creating bioinformatics software documentation. *Briefings in Bioinformatics* doi:10.1093/bib/bbw134

28. Anderson, A. Writing Great Scientific Code. (2016).

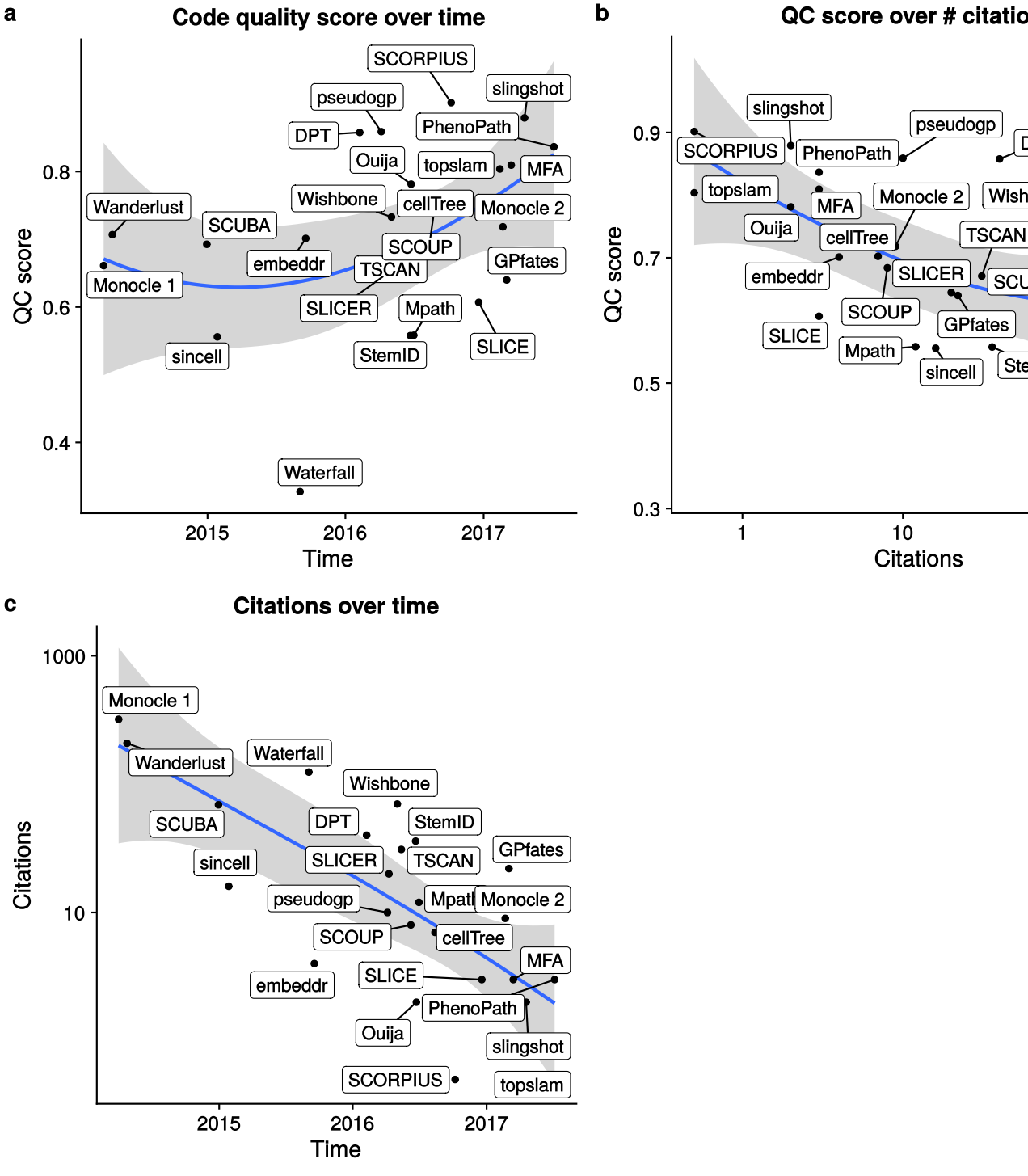
29. jfp. Green dice are loaded (welcome to p-hacking) (IT Best Kept Secret Is Optimization). (2016).

Supplementary Material

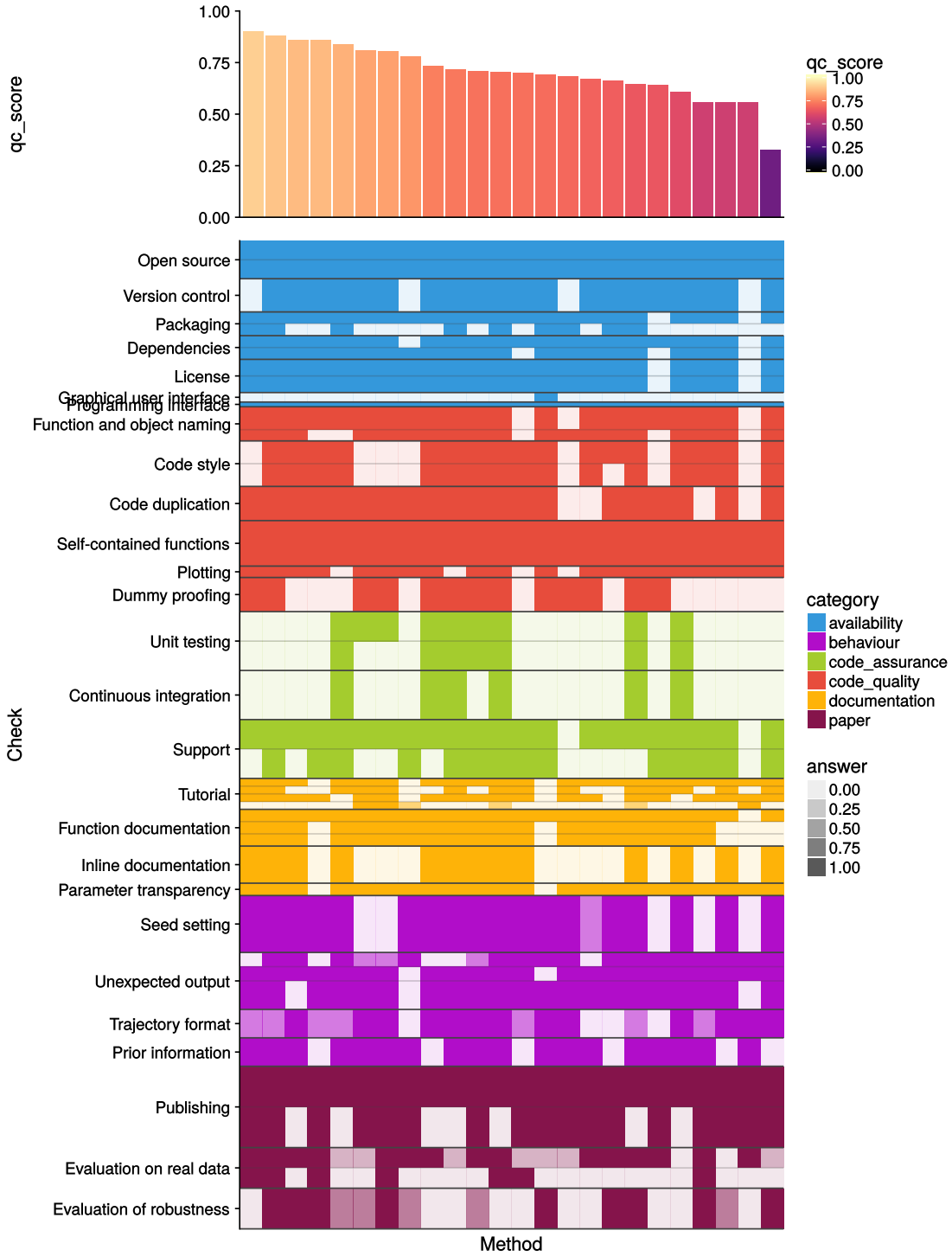
Supplementary Figures



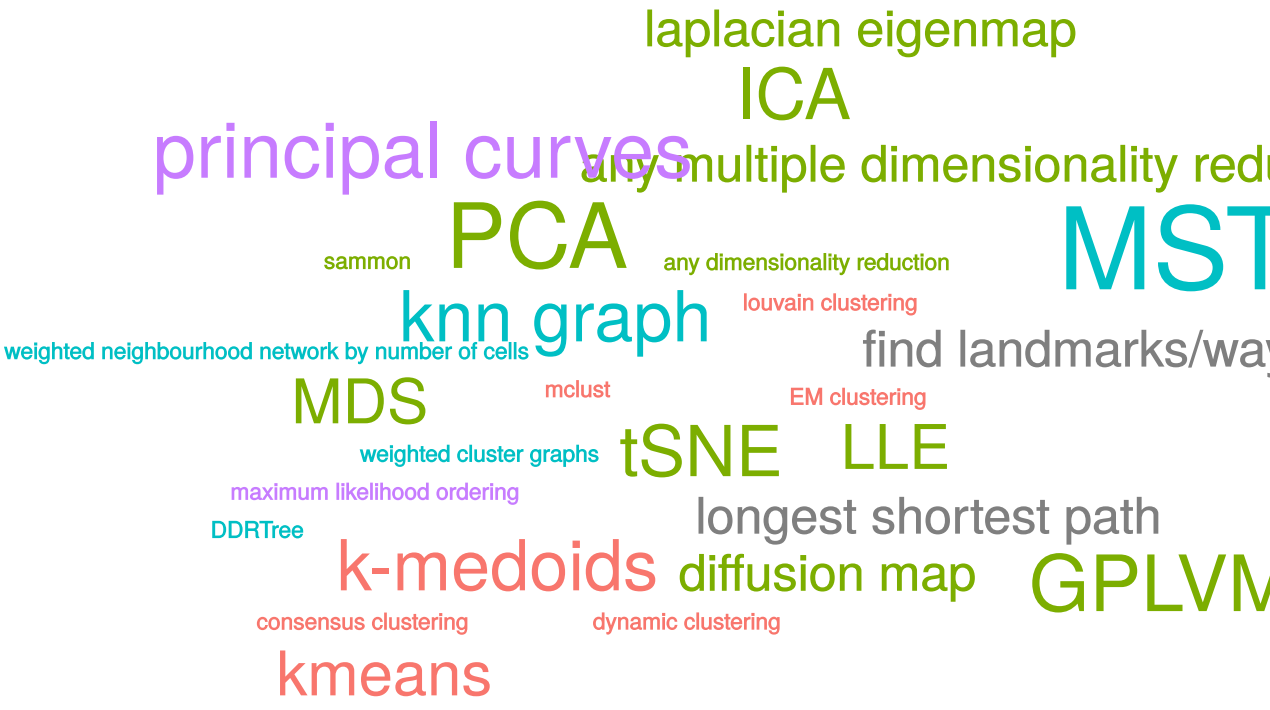
Supplementary Figure 1 New TI methods over time. **a** Number of methods published or in preprint. **b** Number of methods which can handle a particular type of trajectory.


a
Code quality score over time
QC score
Time
0.8
0.6
0.4
2015
2016
2017
SCORPIUS
slingshot
pseudogp
PhenoPath
DPT
Ouija
topslam
MFA
Wishbone
cellTree
Monocle 2
Wanderlust
SCUBA
SCOUP
embeddr
TSCAN
GPfates
Monocle 1
SLICER
Mpath
SLICE
sincell
StemID
Waterfall
b
QC score over # citatio
QC score
Citations
0.9
0.7
0.5
0.3
1
10
slingshot
pseudogp
SCORPIUS
PhenoPath
topslam
MFA
Monocle 2
Ouija
cellTree
TSCAN
embeddr
SLICER
SCOUP
GPfates
SLICE
Mpath
sincell
c
Citations over time
Citations
Time
1000
10
2015
2016
2017
Monocle 1
Waterfall
Wanderlust
Wishbone
SCUBA
DPT
StemID
sincell
SLICER
TSCAN
GPfates
pseudogp
Mpath
Monocle 2
SCOUP
cellTree
MFA
embeddr
SLICE
PhenoPath
Ouija
slingshot
SCORPIUS
topslam

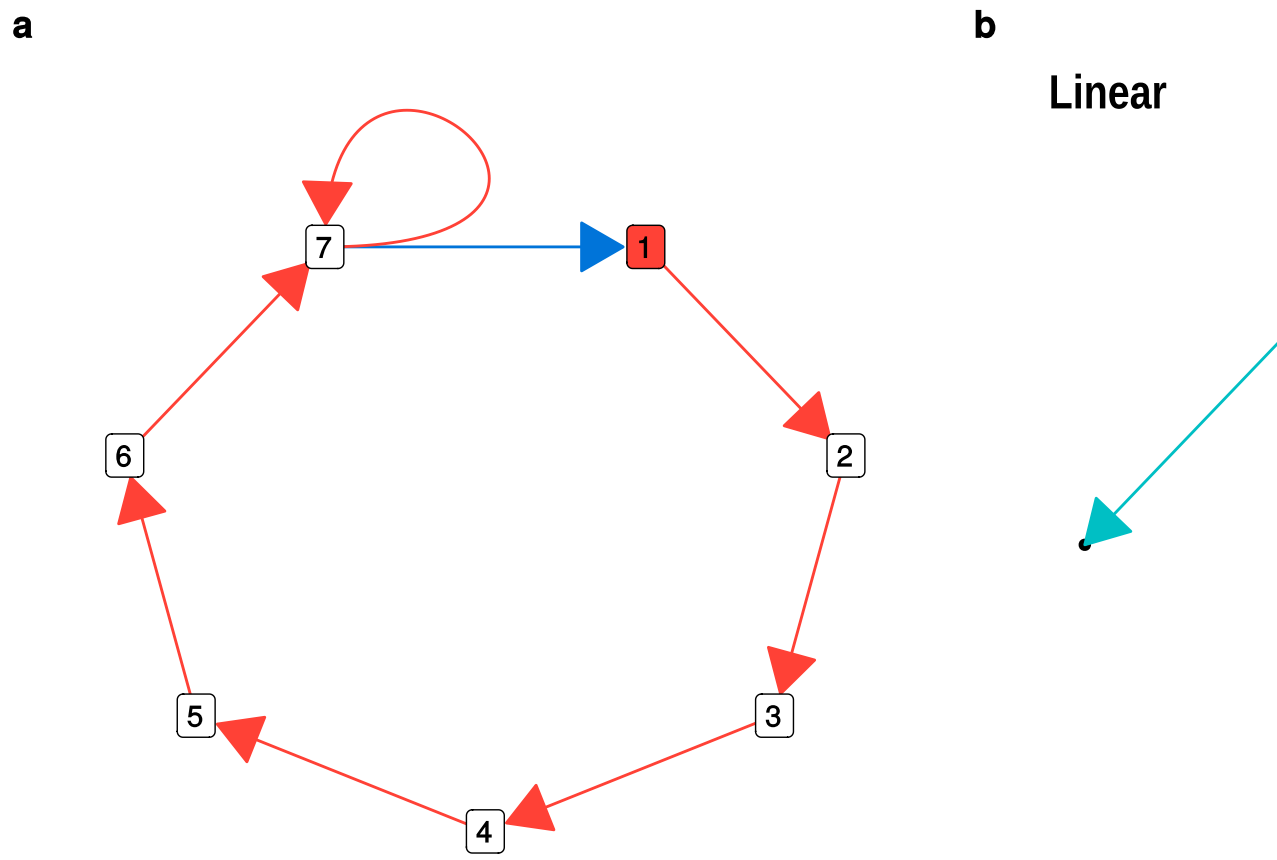
Supplementary Figure 2 Quality control scores and number of citations over time


qc_score
1.00
0.75
0.50
0.25
0.00
qc_score
1.00
0.75
0.50
0.25
0.00
Open source
Version control
Packaging
Dependencies
License
Graphical user interface
Programming interface
Function and object naming
Code style
Code duplication
Self-contained functions
Plotting
Dummy proofing
Unit testing
Continuous integration
Support
Tutorial
Function documentation
Inline documentation
Parameter transparency
Seed setting
Unexpected output
Trajectory format
Prior information
Publishing
Evaluation on real data
Evaluation of robustness
Check
Method
category
availability
behaviour
code_assurance
code_quality
documentation
paper
answer
0.00
0.25
0.50
0.75
1.00

Supplementary Figure 3 Caption


laplacian eigenmap
ICA
principal curves
any multiple dimensionality redu
PCA
MST
sammon
any dimensionality reduction
knn graph
louvain clustering
weighted neighbourhood network by number of cells
find landmarks/way
MDS
mclust
EM clustering
weighted cluster graphs
tSNE
LLE
maximum likelihood ordering
longest shortest path
DDRTree
k-medoids
diffusion map
GPLVM
consensus clustering
dynamic clustering
kmeans

Supplementary Figure 4 TI methods frequently share steps



Supplementary Figure 5 Module networks, milestone networks and the module dynamics for each of the different types of synthetic data

Supplementary Tables

Supplementary Table 1

Aspect

Weight

References

Item

Item weight

Developer friendly

User friendly

Good science

Availability

Is the code freely available?

open_source 8

open_source[6,21–26]

Code is freely available

0.50

Tool can be run on a freely available platform

0.50

Is the code version controlled?

version_control 7

version_control[6,21–25]

Code is available on a public version controlled repository, such as Github

1.00

Is the code contained into an easy to install package? Is it discoverable?

packaging 5

packaging[21,25,26]

The code is provided as a "package", exposing functionality through functions or shell commands

0.50

The code can be easily installed through a repository such as CRAN, Bioconductor, PyPI, CPAN, debian packages, …

0.50

Are dependencies clearly stated and available?

dependencies 5

dependencies[22–24,27]

Dependencies are clearly stated in the tutorial or in the code

0.50

Dependencies are automatically installed

0.50

Is the license of the code clear? Does this license permit academic use?

license 7

license[21–26]

License of the code is clear

0.50

License allows academic use

0.50

Does the tool have a graphical user interface?

gui 2

gui[25]

The tool can be run using a graphical user interface, either locally or on a web server (in contrast to a programming interface)

1.00

Does the tool have a programming interface?

programming_interface 1

programming_interface

The tool can be run through the command line or through a programming language (in contrast to a GUI)

1.00

Code quality

Do the functions and objects have a consistent naming?

naming 3

naming[6,23]

Functions have well chosen names

0.67

Arguments have well chosen names

0.33

Do the functions have a consistent style?

style 4

style[6,23,24]

Code is styled consistently

0.50

Code following (basic) good practices in the programming language of choice

0.50

Is code frequently duplicated?

duplication 3

duplication[6,23]

Duplicated code is minimal

1.00

Does the code expose certain steps in the method as self-contained functions or commands?

pure_functions 4

pure_functions[22,25,28]

The method is exposed to the user as self-contained functions or commands

1.00

Does the package allow plotting of (intermediate) results?

plotting 1

plotting

Plotting functions are provided for the final and/or intermediate results

1.00

Does the package include dummy proofing?

dummy_proofing 3

dummy_proofing[21,27]

Package contains dummy proofing, i.e. testing whether the parameters and data supplied by the user make sense and are useful

1.00

Code assurance

Does the package include some testing?

testing 6

testing[6,21,23,25,28]

Code contains unit tests

0.50

Tests are run automatically using functionality from the programming language

0.50

Does the package include some continuous integration?

continuous_integration 5

continuous_integration[5,23–25]

Continuous integration, for example on Travis CI

1.00

Does the package include a system to ask for support?

support 6

support[6,23–26]

There is a support ticket system, for example on Github

0.50

Tickets are closed within a reasonable time frame

0.50

Documentation

Is there a tutorial available for the method? Does this tutorial show everything the user needs?

tutorial 5

tutorial[23,25–27]

A tutorial or vignette is available

0.25

The tutorial has example results

0.25

The tutorial has real example data
0.25

The use of the method is shown with several (1=0, 2=0.5, >2=1) datasets
0.25

Is the purpose and usage of each function documented?
documentation 6
documentation[6,22,23,25,27]
The purpose and usage of functions is documented
0.33

The parameters of functions are documented
0.33

The output of functions is documented
0.33

Is the code documented inline?
inline_documentation 6
inline_documentation[6,22,23,25,27]
Inline documentation is present in the code, to make it understandable
1.00

Are all important parameters available to the user?
parameter_transparency 2
parameter_transparency[22]
All important parameters are exposed to the user
1.00

Behaviour

Are no seeds set during the execution of the method?

seed_setting 2

seed_setting[29]

No seeds (1), some seeds (0.5) or a lot of seeds (0) are set

1.00

Is unexpected output generated by the method?

unexpected_output 2

unexpected_output[24]

No unexpected output messages

0.25

No unexpected files, folders or plots are generated

0.25

No unexpected warnings during runtime or compilation are generated

0.50

Was postprocessing necessary to get the output of the method into a useful format?

format 1

format

The postprocessing is minimal (1), moderate (0.5) or extensive (0)

1.00

Does the method require prior information?

prior_information 1

prior_information

Prior information is required

1.00

Paper

Is the method published in a peer-reviewed journal?

publishing 2

publishing[27]

The method is published

0.50

The paper is published in a peer-reviewed journal

0.50

Is the methods usefulness shown in the paper?

evaluation 1

evaluation

The paper shows the method's usefulness on several (1), one (0.25) or no datasets.

0.50

The paper quantifies the accuracy of the method on real data given a gold or silver standard trajectory

0.50

Does the paper assess method robustness?

robustness 1

robustness

Method robustness (to eg. noise, subsampling, parameter changes, stability) is assessed in one (0.5) or several (1) ways

1.00

Supplementary Table 2 Real datasets used in this study

Date	Gse	Organism	Technology	Trajectory type	Standard determination	Number of cells	Number of genes	
GSE52529 2014-03-23	GSE52529	GSE52529 mouse	GSE52529 fluidigm c1	Directed linear	timeseries	291	3582	
GSE52583 2014-04-08	GSE52583	GSE52583 mouse	GSE52583 fluidigm c1	Multifurcation	clustering(hierarchical)	65	457	
GSE48968 2014-06-11	GSE48968	GSE48968 mouse	GSE48968 smart-seq	Directed linear	timeseries	541		

3630
Directed linear
timeseries
408
3387
Directed linear
timeseries
435
3604
E-MTAB-2805 2014-12-29
E-MTAB-2805
E-MTAB-2805 mouse
E-MTAB-2805 fluidigm c1
Directed cycle
FACS
264
5310
GSE63818 2015-06-03
GSE63818
GSE63818 human
GSE63818 Tang et. al.
Bifurcation
timeseries,sample_origin
272
5708
Directed linear
timeseries,sample_origin
166
3457
Directed linear
timeseries,sample_origin
101

3290
GSE64016 2015-06-24
GSE64016
GSE64016 human
GSE64016 fluidigm c1
Directed cycle
clustering(oscope)
222
3725
GSE60781 2015-07-16
GSE60781
GSE60781 mouse
GSE60781 fluidigm c1
Directed linear
FACS
238
1845
E-MTAB-3929 2015-09-25
E-MTAB-3929
E-MTAB-3929 human
E-MTAB-3929 fluidigm c1
Directed linear
timeseries
1299
3447
GSE59114 2015-10-01
GSE59114
GSE59114 mouse
GSE59114 smart-seq
Directed linear
FACS
873

2863
Directed linear
FACS
493
2406
GSE71982 2015-10-15
GSE71982
GSE71982 mouse
GSE71982 fluidigm c1
Directed acyclic graph
clustering
158
1737
Directed linear
clustering
117
912
Directed linear
clustering
85
228
Directed linear
clustering
85
313
GSE74596 2016-04-18
GSE74596
GSE74596 mouse
GSE74596 fluidigm c1
Bifurcation
FACS
197

3982
GSE67310 2016-06-08
GSE67310
GSE67310 mouse
GSE67310 fluidigm c1
Bifurcation
timeseries,clustering(PCA,hierarchical)
355
3301
GSE75330 2016-06-10
GSE75330
GSE75330 mouse
GSE75330 fluidigm c1
Directed linear
clustering(BackSpinV2)
3694
65
Multifurcation
clustering(BackSpinV2)
4959
74
GSE85066 2016-08-02
GSE85066
GSE85066 human
GSE85066 fluidigm c1
Rooted tree
FACS
501
3523
GSE70240,GSE70243,GSE70244,GSE70236 2016-08-26
GSE70240,GSE70243,GSE70244,GSE70236
GSE70240,GSE70243,GSE70244,GSE70236 mouse

GSE70240,GSE70243,GSE70244,GSE70236 fluidigm c1

Directed linear

FACS

318

2702

GSE70240,GSE70244,GSE70236 2016-08-26

GSE70240,GSE70244,GSE70236

GSE70240,GSE70244,GSE70236 mouse

GSE70240,GSE70244,GSE70236 fluidigm c1

Rooted tree

FACS,clustering(ICGS)

376

2967

GSE79363 2016-09-14

GSE79363

GSE79363 mouse

GSE79363 smart-seq2

Bifurcation

FACS

60

1849

GSE67602 2016-09-20

GSE67602

GSE67602 mouse

GSE67602 fluidigm c1

Directed linear

clustering(AP)

749

460

Directed linear

clustering(AP)

699

353
Directed linear
clustering(AP)
346
298
GSE90860 2017-03-08
GSE90860
GSE90860 mouse
GSE90860 fluidigm c1
Bifurcation
timeseries,clustering(PCA,hierarchical)
213
4872
GSE86146 2017-03-31
GSE86146
GSE86146 human
GSE86146 smart-seq2
Directed linear
clustering(DBclust)
659
3630
Directed linear
clustering(DBclust)
629
4691
Directed linear
timeseries(sample_origin)
659
3630
Directed linear
timeseries(sample_origin)
671

4459
GSE87375 2017-05-02
GSE87375
GSE87375 mouse
GSE87375 smart-seq2
Directed linear
timeseries
322
6763
Directed linear
timeseries
563
6372
GSE90047 2017-07-14
GSE90047
GSE90047 mouse
GSE90047 smart-seq2
Bifurcation
timeseries,clustering(PCA,hierarchical)
503
1295
GSE99951 2017-07-31
GSE99951
GSE99951 mouse
GSE99951 smart-seq2
Directed linear
timeseries
192
413
Directed linear
timeseries
456

3585

Supplementary Table 3 Distributions from which each parameter in the thermodynamic model was sampled

$r = \mathcal{U}(10, 200)$

$d = \mathcal{U}(2, 8)$

$p = \mathcal{U}(2, 8)$

$q = \mathcal{U}(1, 5)$

$a0 = \begin{cases}1 & \text{if } e \in \{1\} \text{ or } e = \emptyset \\ 0 & \text{if } e \in \{-1\} \\ 0.5 & \text{otherwise}\end{cases}$

$a_i = \begin{cases}1 & \text{if } e_i \in \{-1\} \\ 0 & \text{otherwise}\end{cases}$

$s = \mathcal{U}(1, 20)$

$k = y_{max} / (2 * s)$

$c = \mathcal{U}(1, 4)$

where

$$y_{max} = r/d * p/q$$

Colophon

This report was generated on 2018-02-01 16:27:34 using R version 3.4.3 (2017-11-30) and the following packages:

package	*	version	date	source
acepack		1.4.1	2016-10-29	CRAN (R 3.4.1)
AnnotationDbi		1.40.0	2018-01-11	cran (**???**)
ape		5.0	2017-10-30	cran (**???**)
assertthat		0.2.0	2017-04-11	CRAN (R 3.4.1)
backports		1.1.2	2017-12-13	cran (**???**)
base	*	3.4.3	2017-12-01	local
base64enc		0.1-3	2015-07-28	CRAN (R 3.4.1)
BBmisc		1.11	2017-03-10	cran (**???**)
beeswarm		0.2.3	2016-04-25	CRAN (R 3.4.1)
bigmemory	*	4.5.31	2017-11-20	CRAN (R 3.4.2)
bigmemory.sri	*	0.1.3	2014-08-18	CRAN (R 3.4.1)
bindr		0.1	2016-11-13	CRAN (R 3.4.1)
bindrcpp	*	0.2	2017-06-17	CRAN (R 3.4.1)
Biobase	*	2.38.0	2018-01-11	cran (**???**)

package	*	version	date	source
BiocGenerics	*	0.24.0	2018-01-11	cran (**???**)
BiocParallel		1.12.0	2018-01-11	cran (**???**)
biomaRt		2.34.1	2018-01-11	cran (**???**)
bit		1.1-12	2014-04-09	CRAN (R 3.4.1)
bit64		0.9-7	2017-05-08	CRAN (R 3.4.1)
bitops		1.0-6	2013-08-17	CRAN (R 3.4.1)
blob		1.1.0	2017-06-17	CRAN (R 3.4.1)
bookdown		0.6	2018-01-25	CRAN (R 3.4.3)
broom		0.4.3	2017-11-20	cran (**???**)
caret		6.0-77	2017-09-07	CRAN (R 3.4.1)
caTools		1.17.1	2014-09-10	cran (**???**)
cellranger		1.1.0	2016-07-27	CRAN (R 3.4.1)
checkmate		1.8.5	2017-10-24	CRAN (R 3.4.2)
class		7.3-14	2015-08-30	CRAN (R 3.4.0)
cli		1.0.0	2017-11-05	cran (**???**)
cluster		2.0.6	2017-03-16	CRAN (R 3.4.0)
codetools		0.2-15	2016-10-05	CRAN (R 3.3.1)
colorspace		1.3-2	2016-12-14	CRAN (R 3.4.1)
compiler		3.4.3	2017-12-01	local
coRanking		0.1.3	2016-09-16	cran (**???**)
cowplot	*	0.9.2	2017-12-17	cran (**???**)
crayon		1.3.4	2017-09-16	CRAN (R 3.4.1)
curl		3.1	2017-12-12	cran (**???**)
CVST		0.2-1	2013-12-10	CRAN (R 3.4.1)
data.table		1.10.4-3	2017-10-27	CRAN (R 3.4.2)
datasets	*	3.4.3	2017-12-01	local
DBI		0.7	2017-06-18	CRAN (R 3.4.1)
ddalpha		1.3.1	2017-09-27	CRAN (R 3.4.1)
DelayedArray		0.4.1	2018-01-11	cran (**???**)
DEoptimR		1.0-8	2016-11-19	cran (**???**)
devtools		1.13.4	2017-11-09	cran (**???**)
diffusionMap		1.1-0	2014-02-20	cran (**???**)
digest		0.6.15	2018-01-28	cran (**???**)
dimRed		0.1.0	2017-05-04	CRAN (R 3.4.1)
diptest		0.75-7	2016-12-05	cran (**???**)
doParallel		1.0.11	2017-09-28	CRAN (R 3.4.2)
dplyr	*	0.7.4	2017-09-28	CRAN (R 3.4.3)
DRR		0.0.2	2016-09-15	CRAN (R 3.4.1)
DT		0.2	2016-08-09	cran (**???**)
dtw		1.18-1	2015-09-01	CRAN (R 3.4.1)
dynalysis	*	0.0.0.9000	2018-01-30	local
dynamicTreeCut		1.63-1	2016-03-11	cran (**???**)
dyneval	*	0.1.0	2018-01-30	local
dyngen	*	0.1	2018-02-01	local

package	*	version	date	source
dynmethods	*	0.1.0	2018-01-29	local
dynplot		0.1.0	2018-01-11	Github (Zouter/dynplot@a6c33a0)
dyntoy	*	0.1.0	2018-01-24	local
dynutils	*	0.1.0	2018-01-31	local
edgeR		3.20.6	2018-01-11	cran (**???**)
emoa		0.5-0	2012-09-25	cran (**???**)
evaluate		0.10.1	2017-06-24	CRAN (R 3.4.1)
fastgssa		0.6-0	2018-01-11	Github (dynverse/fastgssa@92cfc08)
fitdistrplus		1.0-9	2017-03-24	cran (**???**)
flexmix		2.3-14	2017-04-28	cran (**???**)
FNN		1.1	2013-07-31	cran (**???**)
forcats	*	0.2.0	2017-01-23	CRAN (R 3.4.1)
foreach		1.4.4	2017-12-12	cran (**???**)
foreign		0.8-69	2017-06-21	CRAN (R 3.4.0)
Formula		1.2-2	2017-07-10	CRAN (R 3.4.1)
fpc		2.1-10	2015-08-14	cran (**???**)
GA		3.0.2	2016-06-07	CRAN (R 3.4.1)
gdata		2.18.0	2017-06-06	CRAN (R 3.4.1)
gdtools	*	0.1.6	2017-09-01	CRAN (R 3.4.3)
GEDEVO		1.0	2018-01-11	Github (dynverse/GEDEVO@c5e2669)
GenomeInfoDb		1.14.0	2018-01-11	cran (**???**)
GenomeInfoDbData		1.0.0	2018-01-11	cran (**???**)
GenomicRanges		1.30.1	2018-01-11	cran (**???**)
ggbeeswarm		0.6.0	2017-08-07	CRAN (R 3.4.1)
ggforce		0.1.1	2016-11-28	cran (**???**)
ggplot2	*	2.2.1	2016-12-30	CRAN (R 3.4.1)
ggraph	*	1.0.0	2017-02-24	CRAN (R 3.4.3)
ggrepel		0.7.0	2017-09-29	cran (**???**)
ggridges		0.4.1	2017-09-15	cran (**???**)
git2r		0.21.0.9000	2018-02-01	Github (ropensci/git2r@156d4fd)
glue		1.2.0	2017-10-29	CRAN (R 3.4.2)
googledrive	*	0.1.1	2017-08-28	CRAN (R 3.4.3)
googlesheets	*	0.2.2	2017-05-07	CRAN (R 3.4.3)
gower		0.1.2	2017-02-23	CRAN (R 3.4.1)
gplots		3.0.1	2016-03-30	cran (**???**)
graphics	*	3.4.3	2017-12-01	local
grDevices	*	3.4.3	2017-12-01	local
grid		3.4.3	2017-12-01	local
gridBase		0.4-7	2014-02-24	CRAN (R 3.4.1)
gridExtra		2.3	2017-09-09	CRAN (R 3.4.1)
gtable		0.2.0	2016-02-26	CRAN (R 3.4.1)
gtools		3.5.0	2015-05-29	CRAN (R 3.4.1)
haven		1.1.0	2017-07-09	CRAN (R 3.4.1)
here		0.1-10	2018-01-16	Github (krlmlr/here@154d8c7)

package	*	version	date	source
highr		0.6	2016-05-09	CRAN (R 3.4.1)
Hmisc		4.1-1	2018-01-03	cran (**???**)
hms		0.4.0	2017-11-23	cran (**???**)
htmlTable		1.11.1	2017-12-27	cran (**???**)
htmltools		0.3.6	2017-04-28	CRAN (R 3.4.1)
htmlwidgets		0.9	2017-07-10	CRAN (R 3.4.1)
httpuv		1.3.5	2017-07-04	CRAN (R 3.4.1)
httr		1.3.1	2017-08-20	cran (**???**)
ica		1.0-1	2015-08-25	cran (**???**)
igraph		1.1.2	2017-07-21	CRAN (R 3.4.1)
ipred		0.9-6	2017-03-01	CRAN (R 3.4.1)
IRanges		2.12.0	2018-01-11	cran (**???**)
irlba		2.3.1	2017-10-18	CRAN (R 3.4.2)
iterators		1.0.9	2017-12-12	cran (**???**)
jsonlite		1.5	2017-06-01	cran (**???**)
kableExtra		0.7.0	2018-01-15	CRAN (R 3.4.3)
kernlab		0.9-25	2016-10-03	cran (**???**)
KernSmooth		2.23-15	2015-06-29	CRAN (R 3.4.0)
knitr		1.19	2018-01-29	CRAN (R 3.4.3)
labeling		0.3	2014-08-23	CRAN (R 3.4.1)
lars		1.2	2013-04-24	cran (**???**)
lattice		0.20-35	2017-03-25	CRAN (R 3.3.3)
latticeExtra		0.6-28	2016-02-09	CRAN (R 3.4.1)
lava		1.5.1	2017-09-27	CRAN (R 3.4.1)
lazyeval		0.2.1	2017-10-29	cran (**???**)
lhs		0.16	2018-01-04	cran (**???**)
limma		3.34.5	2018-01-11	cran (**???**)
limSolve		1.5.5.3	2017-08-14	cran (**???**)
locfit		1.5-9.1	2013-04-20	CRAN (R 3.4.1)
lpSolve		5.6.13	2015-09-19	cran (**???**)
lubridate		1.7.1	2017-11-03	cran (**???**)
magrittr		1.5	2014-11-22	CRAN (R 3.4.1)
MASS		7.3-48	2017-12-24	CRAN (R 3.4.3)
Matrix		1.2-11	2017-08-16	CRAN (R 3.4.1)
matrixStats		0.52.2	2017-04-14	CRAN (R 3.4.1)
mclust		5.4	2017-11-22	cran (**???**)
mco		1.0-15.1	2014-11-29	cran (**???**.)
memoise		1.1.0	2017-04-21	CRAN (R 3.4.1)
methods	*	3.4.3	2017-12-01	local
mime		0.5	2016-07-07	CRAN (R 3.4.1)
miniUI		0.1.1	2016-01-15	cran (**???**)
misc3d		0.8-4	2013-01-25	cran (**???**)
mixtools		1.1.0	2017-03-10	cran (**???**)
mlr		2.11	2017-03-15	cran (**???**)

package	*	version	date	source
mlrMBO		1.1.1	2018-01-02	cran (**???**)
mnormt		1.5-5	2016-10-15	CRAN (R 3.4.1)
ModelMetrics		1.1.0	2016-08-26	CRAN (R 3.4.1)
modelr		0.1.1	2017-07-24	CRAN (R 3.4.1)
modeltools		0.2-21	2013-09-02	CRAN (R 3.4.1)
munsell		0.4.3	2016-02-13	CRAN (R 3.4.1)
mvtnorm		1.0-6	2017-03-02	cran (**???**)
nlme		3.1-131	2017-02-06	CRAN (R 3.4.0)
NMF		0.20.6	2015-05-26	CRAN (R 3.4.1)
nnet		7.3-12	2016-02-02	CRAN (R 3.4.0)
numDeriv		2016.8-1	2016-08-27	cran (**???**-)
openssl		0.9.9	2017-11-10	cran (**???**)
parallel	*	3.4.3	2017-12-01	local
parallelMap		1.3	2015-06-10	cran (**???**)
ParamHelpers		1.11	2018-01-16	Github (berndbischl/ParamHelpers@59c649e)
pbapply		1.3-4	2018-01-10	cran (**???**)
pdist		1.2	2013-02-03	CRAN (R 3.4.1)
pheatmap		1.0.8	2015-12-11	CRAN (R 3.4.1)
pillar		1.0.1	2017-11-27	cran (**???**)
pkgconfig		2.0.1	2017-03-21	CRAN (R 3.4.1)
pkgmaker		0.22	2014-05-14	CRAN (R 3.4.1)
plot3D		1.1.1	2017-08-28	cran (**???**)
plotly		4.7.1	2017-07-29	cran (**???**)
plyr		1.8.4	2016-06-08	CRAN (R 3.4.1)
prabclus		2.2-6	2015-01-14	cran (**???**)
prettyunits		1.0.2	2015-07-13	cran (**???**)
princurve		1.1-12	2013-04-25	cran (**???**)
PRISM		1.0	2018-01-11	Github (rcannood/PRISM@255f82b)
prodlim		1.6.1	2017-03-06	CRAN (R 3.4.1)
progress		1.1.2	2016-12-14	cran (**???**)
proxy		0.4-19	2017-10-29	CRAN (R 3.4.2)
psych		1.7.8	2017-09-09	CRAN (R 3.4.1)
purrr	*	0.2.4	2017-10-18	cran (**???**)
quadprog		1.5-5	2013-04-17	cran (**???**)
R.methodsS3		1.7.1	2016-02-16	cran (**???**)
R.oo		1.21.0	2016-11-01	cran (**???**)
R.utils		2.6.0	2017-11-05	cran (**???**)
R6		2.2.2	2017-06-17	CRAN (R 3.4.1)
random		0.2.6	2017-02-05	cran (**???**)
randomForest		4.6-12	2015-10-07	CRAN (R 3.4.1)
ranger		0.9.0	2018-01-09	cran (**???**)
RColorBrewer		1.1-2	2014-12-07	CRAN (R 3.4.1)
Rcpp		0.12.14	2017-11-23	cran (**???**)
RcppRoll		0.2.2	2015-04-05	CRAN (R 3.4.1)

package	*	version	date	source
RCurl		1.95-4.10	2018-01-04	cran (**???**.)
readr	*	1.1.1	2017-05-16	CRAN (R 3.4.1)
readxl		1.0.0	2017-04-18	CRAN (R 3.4.1)
recipes		0.1.0	2017-07-27	CRAN (R 3.4.1)
registry		0.3	2015-07-08	CRAN (R 3.4.1)
reshape2		1.4.3	2017-12-11	cran (**???**)
rhdf5		2.22.0	2018-01-11	cran (**???**)
rjson		0.2.15	2014-11-03	CRAN (R 3.4.1)
RJSONIO		1.3-0	2014-07-28	cran (**???**)
rlang		0.1.6	2017-12-21	cran (**???**)
rmarkdown	*	1.8	2017-11-17	cran (**???**)
rngtools		1.2.4	2014-03-06	CRAN (R 3.4.1)
robustbase		0.92-8	2017-11-01	cran (**???**)
ROCR		1.0-7	2015-03-26	cran (**???**)
rpart		4.1-11	2017-04-21	CRAN (R 3.4.0)
rprojroot		1.3-2	2018-01-03	cran (**???**)
RSQLite		2.0	2017-06-19	CRAN (R 3.4.1)
rstudioapi		0.7	2017-09-07	CRAN (R 3.4.1)
Rtsne		0.13	2017-04-14	CRAN (R 3.4.1)
rvest		0.3.2	2016-06-17	CRAN (R 3.4.1)
S4Vectors		0.16.0	2018-01-11	cran (**???**)
scales		0.5.0	2017-08-24	CRAN (R 3.4.1)
scater		1.6.1	2018-01-11	cran (**???**)
scatterplot3d		0.3-40	2017-04-22	cran (**???**)
SCORPIUS		1.0	2018-01-11	Github (rcannood/SCORPIUS@6e4d1a5)
scran		1.6.6	2018-01-11	Bioconductor
SDMTools		1.1-221	2014-08-05	cran (**???**)
segmented		0.5-3.0	2017-11-30	cran (**???**)
Seurat		2.2.0	2018-01-10	cran (**???**)
sfsmisc		1.1-1	2017-06-08	CRAN (R 3.4.1)
shiny	*	1.0.5	2017-08-23	CRAN (R 3.4.1)
shinydashboard		0.6.1	2017-06-14	CRAN (R 3.4.1)
SingleCellExperiment		1.0.0	2018-01-11	cran (**???**)
smoof		1.5.1	2017-08-14	cran (**???**)
sn		1.5-1	2017-11-23	cran (**???**)
splines		3.4.3	2017-12-01	local
statmod		1.4.30	2017-06-18	cran (**???**)
stats	*	3.4.3	2017-12-01	local
stats4		3.4.3	2017-12-01	local
strcode		0.2.0	2018-01-29	Github (lorenzwalthert/strcode@f7b07a2)
stringi		1.1.6	2017-11-17	cran (**???**)
stringr	*	1.2.0	2017-02-18	CRAN (R 3.4.1)
SummarizedExperiment		1.8.1	2018-01-11	cran (**???**)
survival		2.41-3	2017-04-04	CRAN (R 3.4.0)

package	*	version	date	source
svglite		1.2.1	2017-09-11	CRAN (R 3.4.3)
tclust		1.3-1	2017-08-24	cran (**???**)
testthat		2.0.0	2017-12-13	cran (**???**)
tibble	*	1.4.1	2017-12-25	cran (**???**)
tidygraph	*	1.0.0	2017-07-07	CRAN (R 3.4.3)
tidyr	*	0.7.2	2017-10-16	cran (**???**)
tidyselect		0.2.3	2017-11-06	cran (**???**)
tidyverse	*	1.2.1	2017-11-14	cran (**???**)
timeDate		3042.101	2017-11-16	cran (**???**)
tools		3.4.3	2017-12-01	local
transport		0.9-4	2017-10-03	cran (**???**)
trimcluster		0.1-2	2012-10-29	cran (**???**)
tsne		0.1-3	2016-07-15	cran (**???**)
TSP		1.1-5	2017-02-22	cran (**???**)
tweenr		0.1.5	2016-10-10	cran (**???**)
tximport		1.6.0	2018-01-11	cran (**???**)
udunits2		0.13	2016-11-17	cran (**???**)
units		0.5-1	2018-01-08	cran (**???**)
utf8		1.1.3	2018-01-03	cran (**???**)
utils	*	3.4.3	2017-12-01	local
VGAM		1.0-4	2017-07-25	cran (**???**)
vipor		0.4.5	2017-03-22	CRAN (R 3.4.1)
viridis		0.4.1	2018-01-08	cran (**???**)
viridisLite		0.2.0	2017-03-24	CRAN (R 3.4.1)
withr		2.1.1	2017-12-19	cran (**???**)
xfun		0.1	2018-01-22	CRAN (R 3.4.3)
XML		3.98-1.9	2017-06-19	CRAN (R 3.4.1)
xml2	*	1.1.1	2017-01-24	CRAN (R 3.4.1)
xtable		1.8-2	2016-02-05	CRAN (R 3.4.1)
XVector		0.18.0	2018-01-11	cran (**???**)
yaml		2.1.16	2017-12-12	cran (**???**)
zlibbioc		1.24.0	2018-01-11	cran (**???**)
zoo		1.8-1	2018-01-08	cran (**???**)

The current Git commit details are:

```
#> Local:   master /home/wouters/thesis/projects/dynverse/dynalysis/
#> Remote:    master @ origin (git@github.com:Zouter/dynalysis.git)
#> Head:    [cac924f] 2018-02-01: update normalisation + synthetic data paper
```

[a]Outdated