Synthetic datasets

To generate synthetic datasets, we used four different synthetic data simulators:

- **dyngen:** Simulations of gene regulatory networks, available at github.com/dynverse/dyngen
- **dyntoy:** Random gradients of expression in the reduced space, available at github.com/dynverse/dyntoy
- **PROSSTT:** Expression is sampled from a linear model which depends on pseudotime[1]
- **Splatter:** Simulations of non-linear paths between different expression states[2]

For every simulator, we took great care to make the datasets as realistic as possible. To do this, we extracted several parameters from all real datasets. We calculated the number of differentially expressed features within a trajectory using a two-way Mann–Whitney U test between every pair of cell groups. These values were corrected for multiple testing using the Benjamini-Hochberg procedure (FDR < 0.05) and we required that a gene was expressed in at least 5% of cells, and had at least a fold-change of 2. We also calculated several other parameters, such as drop-out rates and library sizes using the Splatter package[2]. These parameters were then given to the simulators when applicable, as described for each simulator below. Not every real dataset was selected to serve as a reference for a synthetic dataset. Instead, we chose a set of ten distinct reference real datasets by clustering all the parameters of each real dataset, and used the reference real datasets at the cluster centers from a pam clustering (with $k = 10$, implemented in the R cluster package) to generate synthetic data.

dyngen

The dyngen (github.com/dynverse/dyngen) workflow to generate synthetic data is based on the well established workflow used in the evaluation of network inference methods[3,4] and consists of four main steps: network generation, simulation, gold standard extraction and simulation of the scRNA-seq experiment. At every step, we tried to mirror real regulatory networks, while keeping the model simple and easily extendable. We simulated a total of 110 datasets, with 11 different topologies.

Network generation

One of the main processes involved in cellular dynamic processes is 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 bifurcation seems to be mutual antagonism, where genes[5] strongly repress each other, forcing one of the two to become inactive[6]. Such mutual antagonism can be modelled and simulated[7,8]. Although such a two-gene model is simple and elegant, the reality is frequently more complex, with multiple genes (grouped into modules) repressing each other[9].

To simulate certain trajectory topologies, we therefore designed module networks in which the cells follow a particular trajectory topology given certain parameters. Two module networks generated linear trajectories (linear and linear long), one generated a bifurcation, one generated a convergence, one generated a multifurcation (trifurcating), two generated a tree (consecutive bifurcating and binary tree), one generated an acyclic graph (bifurcating and converging), one generated a complex fork (trifurcating), one generated a rooted tree (consecutive bifurcating) and two generated simple graph structures (bifurcating loop and bifurcating cycle). The structure of these module networks is available at https://github.com/dynverse/dyngen/tree/master/inst/ext_data/modulenetworks.

From these module networks we generated gene regulatory networks in two steps: the main regulatory network was first generated, and extra target genes from real regulatory networks were added. For each dataset, we used the same number of genes as were differentially expressed in the real datasets. 5% of the genes were assigned to be part of the main regulatory network, 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 a 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[10]), and extracted for every regulator a local network around it using personalized pagerank (with damping factor set to 0.1), as implemented in the `page_rank` function of the *igraph* package.

Simulation of gene regulatory systems using thermodynamic models

To simulate the gene regulatory network, we used a system of differential equations similar to those used in evaluations of gene regulatory network inference methods[4]. In this model, the changes in gene expression (x_i) and protein expression

(y_i) are modeled using ordinary differential equations[3] (ODEs):

$$\frac{dx_i}{dt} = \underbrace{m \times f(y_1, y_2, ...)}_{\text{production}} - \underbrace{\lambda \times x_i}_{\text{degradation}}$$
$$\frac{dy_i}{dt} = \underbrace{r \times x_i}_{\text{production}} - \underbrace{\Lambda \times y_i}_{\text{degradation}}$$

where m, λ, r and Λ 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 α_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:

$$f(y_1, y_2, ..., y_n) = \sum_{j \in \{0,1,...,n^2\}} \alpha_j \times P_j$$

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:

$$P_j \propto \nu = \left(\frac{y}{k}\right)^n$$

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:

$$P_j \propto \nu = \rho \times \prod_j \left(\frac{y_j}{k_j}\right)^{n_j}$$

where ρ represents the cooperativity of binding between the different transcription factors.

P_i is only proportional to ν because ν is normalized such that $\sum_i P_i = 1$.

To each differential equation, we added an additional stochastic term:

$$\frac{dx_i}{dt} = m \times f(y_1, y_2, ...) - \lambda \times x_i + \eta \times \sqrt{x_i} \times \Delta W_t$$
$$\frac{dy_i}{dt} = r \times x_i - \Lambda \times y_i + \eta \times \sqrt{y_i} \times \Delta W_t$$

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

Similar to GeneNetWeaver[3], we sample the different parameters from random distributions, defined as follows. e defines whether a transcription factor activates (1) or represses (-1), as defined within the regulatory network network.

$$
\begin{aligned}
r &= \mathcal{U}(10, 200) \\
d &= \mathcal{U}(2, 8) \\
p &= \mathcal{U}(2, 8) \\
q &= \mathcal{U}(1, 5) \\
a_0 &= \begin{cases} 1 & \text{if } |e| = 0 \\ 1 & \text{if } \forall x \in e, x = -1 \\ 0 & \text{if } \forall x \in e, x = 1 \\ 0.5 & \text{otherwise} \end{cases} \\
a_i &= \begin{cases} 0 & \text{if } \exists x \in e_i, x = -1 \\ 1 & \text{otherwise} \end{cases} \\
s &= \mathcal{U}(1, 20) \\
k &= y_{max}/(2 * s), \\
&\quad \text{where } y_{max} = r/d \times p/q \\
c &= \mathcal{U}(1, 4)
\end{aligned}
$$

We converted each ODE to an SDE by adding a chemical Langevin equation, as described in[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. These cells were sampled uniformly across the different steps of the 32 simulations. Next, we used the Splatter package[2] 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 $Y'_{i,j} \sim \text{Poisson}(\lambda_{i,j})$. Finally, we simulated dropouts by setting true counts to zero by sampling from a Bernoulli distribution using a dropout probability $\pi^D_{i,j} = \frac{1}{1+e^{-k(\ln(\lambda_{i,j})-x_0)}}$. Both x_0 (the midpoint for the dropout logistic function) and k (the shape of the dropout logistic function) were estimated by Splatter.

This count matrix was then filtered and normalised using the pipeline described below.

Gold standard extraction

Because each cellular simulation follows the trajectory at its own speed, knowing the exact position of a cell within the trajectory topology is not straightforward. Furthermore, the speed at which simulated cells make a decision between two or more alternative paths is highly variable. We therefore first constructed a backbone expression profile for each branch within the trajectory. To do this, we first defined in which order the expression of the modules is expected to change, and then generated a backbone expression profile in which the expression of these modules increases and decreases smoothly between 0 and 1. We also 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[11,12], 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 topologies, the number of possible milestones a backbone could progress through was limited to 20.

dyntoy

For more simplistic data generation ("toy" datasets), we created the dyntoy workflow (github.com/dynverse/dyntoy) . We created 12 topology generators (described below), and with 10 datasets per generator, this lead to a total of 120 datasets.

We created a set of topology generators, were $B(n, p)$ denotes a binomial distribution, and $U(a, b)$ denotes a uniform distribution:

- Linear and cyclic, with number of milestones $\sim B(10, 0.25)$
- Bifurcating and converging, with four milestones
- Binary tree, with number of branching points $\sim U(3, 6)$
- Tree, with number of branching points $\sim U(3, 6)$ and maximal degree $\sim U(3, 6)$

For more complex topologies we first calculated a random number of "modifications" $\sim U(3, 6)$ and a *deg*$_{max}$ $\sim B(10, 0.25) + 1$. For each type of topology, we defined what kind of modifications are possible: divergences, loops, convergences and divergence-convergence. We then iteratively constructed the topology by uniformly sampling from the set of possible modifications, and adding this modification to the existing topology. For a divergence, we connected an existing milestone to a number of a new milestones. Conversely, for a convergence we connected a number of new nodes to an existing node. For a loop, we connected two existing milestones with a number of milestones in between. Finally for a divergence-convergence we connected an existing milestone to several new milestones which again converged on a new milestone. The number of nodes was sampled from $\sim B(\textit{deg}_{max} - 3, 0.25) + 2$

- Looping, allowed loop modifications
- Diverging-converging, allowed divergence and converging modifications
- Diverging with loops, allowed divergence and loop modifications
- Multiple looping, allowed looping modifications
- Connected, allowed looping, divergence and convergence modifications
- Disconnected, number of components sampled from $\sim B(5, 0.25) + 2$, for each component we randomly chose a topology from the ones listed above

After generating the topology, we sampled the length of each edge $\sim U(0.5, 1)$. We added regions of delayed commitment to a divergence in a random half of the cases. We then placed the number of cells (same number as from the reference real dataset), on this topology uniformly, based on the length of the edges in the milestone network.

For each gene (same number as from the reference real dataset), we calculated the Kamada-Kawai layout in 2 dimensions, with edge weight equal to the length of the edge. For this gene, we then extracted for each cell a density value using a bivariate normal distribution with $\mu \sim U(x_{min}, x_{min})$ and $\sigma \sim U(x_{min}/10, x_{min}/8)$. We used this density as input for a zero-inflated negative binomial distribution with $\mu\, U(100, 1000) \times$ *density*, $k\, U(\mu/10, \mu/4)$ and pi from the parameters of the reference real dataset, to get the final count values.

This count matrix was then filtered and normalised using the pipeline described below.

PROSSTT

PROSSTT is a recent data simulator[1], which simulates expression using linear mixtures of expression programs and random walks through the trajectory. We used 5 topology generators from dyntoy (linear, bifurcating, multifurcating, binary tree and tree), and simulated for each topology generator 10 datasets using different reference real datasets. However, due to frequent crashes of the tool, only 19 datasets created output and were thus used in the evaluation.

Using the `simulate_lineage` function, we simulated the lineage expression, with parameters $U(0.01, 0.1)$, *branch-tol*$_{intra}$ $\sim U(0, 0.9)$ and *branch-tol*$_{inter}$ $\sim U(0, 0.9)$. These parameter distributions were chosen very broad so as to make sure both easy and difficult datasets are simulated. After simulating base gene expression with `simulate_base_gene_exp`, we used the `sample_density` function to finally simulate expression values of a number of cells (the same as from the reference real dataset), with $\alpha \sim$ *Lognormal* ($\mu = 0.3$ and $\sigma = 1.5$) and $\beta \sim$ *Lognormal* ($\mu = 2$ and $\sigma = 1.5$). Each of these parameters were centered around the default values of PROSSTT, but with enough variability to ensure a varied set of datasets.

This count matrix was then filtered and normalised using the pipeline described below.

Splatter

Splatter[2] simulates expression values by constructing non-linear paths between different states, each having a distinct expression profile. We used 5 topology generators from dyntoy (linear, bifurcating, multifurcating, binary tree and tree), and simulated for each topology generator 10 datasets using different reference real datasets, leading to a total of 50 datasets.

We used the `splatSimulatePaths` function from Splatter to simulate datasets, with number of cells and genes equal to those in the reference real dataset, and with parameters *nonlinearProb*, *sigmaFac* and *skew* all sampled from $U(0, 1)$.

References

1. Papadopoulos, N., Parra, R. G. & Soeding, J. PROSSTT: Probabilistic simulation of single-cell RNA-seq data for complex differentiation processes. *bioRxiv* 256941 (2018). doi:10.1101/256941

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

3. 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).

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

5. 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).

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

7. 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).

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

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

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

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

12. 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).