Universiteit
Leiden
The Netherlands

Learning Cell-to-Cell Interactions for Vascular Network Formation

Scholten, Willem

Citation
Scholten, W. (2026). *Learning Cell-to-Cell Interactions for Vascular Network Formation*.

Version:	Not Applicable (or Unknown)
License:	License to inclusion and publication of a Bachelor or Master Thesis, 2023
Downloaded from:	https://hdl.handle.net/1887/4297329

Note: To cite this publication please use the final published version (if applicable).

W. Scholten

Learning Cell-to-Cell Interactions for Vascular Network Formation

Bachelor thesis

July 18, 2025

Thesis supervisors: M. van den Bosch
dr. D. van der Hoeven
prof. dr. H.J. Hupkes

Leiden University
Mathematical Institute

Contents

1 Introduction

Differential equations have long stood at the heart of mathematical biology, offering powerful frameworks to unravel the intricate dynamics underlying biological phenomena [6, 22]. From the pioneering formulations of Newton and Leibniz to modern computational models, mathematical tools have continually expanded our capacity to explore and predict biological behaviors. Yet, when faced with complex biological systems like vascular network formation, deriving governing equations from first principles can be extraordinarily challenging [19]. This complexity arises due to the inherent variability and nonlinearity of biological systems, the intricate spatial-temporal interactions among components, and the difficulty in precisely measuring system states. The biological focus of this thesis centers around vasculogenesis, the formation of new blood vessels from progenitor cells, a vital process whose underlying cell-to-cell interactions remain poorly understood [27].

We illuminate those interactions by leveraging modern machine learning methods to discern the cell-to-cell interactions responsible for vascular network development. First, we explore how the data-driven Sparse Identification of Nonlinear Dynamics (SINDy) approach infers governing equations from these simulations [3]. Second, we assess whether Shallow Recurrent Encoder-Decoder (SHRED) networks can predict cell movement based on the same simulations [34]. These methods are powerful tools capable of extracting meaningful information from high-dimensional and noisy biological data, even when traditional analytical methods prove inadequate.

To systematically examine vascular network formation, this work investigates two complementary modeling frameworks: the Cellular Potts Model (CPM) and Particle Based Model (PBM), as described by Boas et al. [2] and Palachanis et al. [24], respectively. The CPM provides a versatile stochastic cellular automaton framework well-suited to represent the complex interplay between mechanical cell interactions, stochastic fluctuations, and chemotactic guidance cues. In contrast, the PBM treat cells as interacting particles whose motion follows deterministic force laws augmented by a small stochastic noise term, providing a flexible framework for simulating cellular aggregation, migration, and network formation in continuous space. We seek to investigate the connection between these two simulation paradigms. Methods used in this comparative analysis can then be used on experimental data.

In deploying machine learning techniques, we first employ SINDy, a method designed to discover sparse representations of dynamical systems directly from observed data. Through SINDy, we aim to identify minimalistic models that capture the essential features governing cell interactions and movements during network formation. On the other hand, SHRED networks are employed to provide robust forecasting of cellular trajectories from observations, emphasizing predictive accuracy and computational efficiency.

Analysis and validation are conducted across multiple scales, dimensions, and parameter spaces. This evaluation seeks to ensure that the derived models and predictions are both mathematically sound and practically applicable. Integrating classical mathematical modeling approaches with advanced machine learning methods offers novel insights into vasculogenesis, enhancing our understanding while simultaneously laying robust computational groundwork to guide future research endeavors in vascular biology and mathematical modeling.

Organization. Section 2 motivates the biological problem and formalizes both the Cellular Potts Model (CPM) and the particle-based model (PBM). Section 3 presents SINDy, SHRED, and their hybrid variants SHRED-ROM and SINDy-SHRED, together with the velocity-estimation schemes and performance metrics. Using this theory, we first validate our methods on a sequence of simplified experiments, after which we compare the CPM and PBM dynamics leveraging SINDy in Section 4. We then turn to forecasting, applying SHRED to the same canonical test cases in Section 5, evaluating its predictive reach. Finally, Section 6 outlines limitations and charts directions for future work.

Main results. In Section 4 we demonstrate that reliable SINDy identification requires more than a single goodness-of-fit metric. Models that boast an excellent score on one metric can still harbour large coefficient errors, so we emphasize the need for multiple metrics. A second experiment highlights the need for sufficiently fine temporal resolution. Coarse sampling degrades finite-difference velocities and drives the sparse regression toward incorrect minima. When we advance to higher-dimensional particle systems, we establish that eight to ten distinct initial configurations are typically enough to break non-identifiability. Finally, the one- and two-dimensional toy models invariably relax to perfectly symmetric equilibria. Because real world vascular structures are not perfectly symmetrical, these toy models prove incapable of reproducing vascular networks. Introducing measurement noise further confirms that large data sets are indispensable.

The ellipse-based PBM experiments extend the insights found with the toy models. We confirm that elongation and long-range attraction alone do not yield branching. Only after injecting stochastic re-orientations and translations do vascular-like networks emerge. In the overdamped surrogate, where fewer force coefficients are unknown, SINDy recovers the parameters to machine accuracy once the step size is sufficiently small. For the full second-order model the drag coefficient enters as an additional unknown. Here we show that fine sampling is critical when de drag coefficient is large, otherwise the acceleration term is swamped and the interaction forces cannot be disentangled. Conversely, in the theoretical limit where the drag coefficient approaches infinity, another first-order approximation becomes exact, and our simplified library captures the dynamics with negligible error, in line with asymptotic expectations.

In Section 5 we show that for the one-dimensional toy system, a minimalist network and only 80 training trajectories is already sufficient for trajectory forecasting if the coordinates of the particles are ordered. The network is able to reliably place every particle at its correct equilibrium position. When the same architecture is applied to an unordered input vector, predictive accuracy degrades vastly, signaling that higher spatial complexity demands either a deeper latent space or a larger training set.

2 Vascular Network Formation

Vascular network formation is categorized in two stages, vasculogenesis and angiogenesis [25]. Vasculogenesis is the process by which new blood vessels form from progenitor cells, primarily during embryonic development [27]. Distinct from angiogenesis, which refers to the formation of new vessels from existing vasculature, vasculogenesis involves the differentiation and assembly of endothelial progenitor cells into primitive vascular structures [14]. This process establishes the initial vascular networks necessary for the subsequent development of the circulatory system. We will focus on this stage in vascular network formation. Because a rigorous definition of a vascular network is still elusive, this thesis evaluates all simulated structures by direct visual comparison with experimental images of vascular networks, as the one shown in Figure 1.

During embryogenesis, vasculogenesis initiates with the emergence of angioblasts from mesodermal precursors [30]. These angioblasts migrate and aggregate to form primitive blood islands, the earliest vascular structures, within the yolk sac and embryonic tissues. These blood islands differentiate into endothelial cells, which subsequently coalesce into tubular networks, creating the primary vascular plexus [28]. This network serves as a foundation for later angiogenesis and vessel remodeling. Figure 1 shows this process, where endothelial cells form robust vascular networks.

Vasculogenesis is not restricted solely to embryonic stages. Recent research indicates a similar process may also occur postnatally, in cases as tissue ischemia, wound healing, and tumor growth [1, 31, 35]. This process involves circulating endothelial progenitor cells mobilized from the bone marrow. These recent findings broaden the potential implications of understanding vasculogenesis in therapeutic contexts, particularly in regenerative medicine and cancer treatment.

Mathematical and computational models of vasculogenesis aim to elucidate the formation of initial vascular networks. Such models range from discrete stochastic simulations, capturing individual cell behaviors and interactions while employing partial differential equations to describe the diffusion and reaction of signaling molecules [24], to ordinary differential equations governing particle movement [2]. Integrating these models with experimental data helps illuminate vasculogenesis, advancing our understanding of fundamental developmental biology and informing therapeutic strategies targeting vascular disorders and tissue engineering applications.

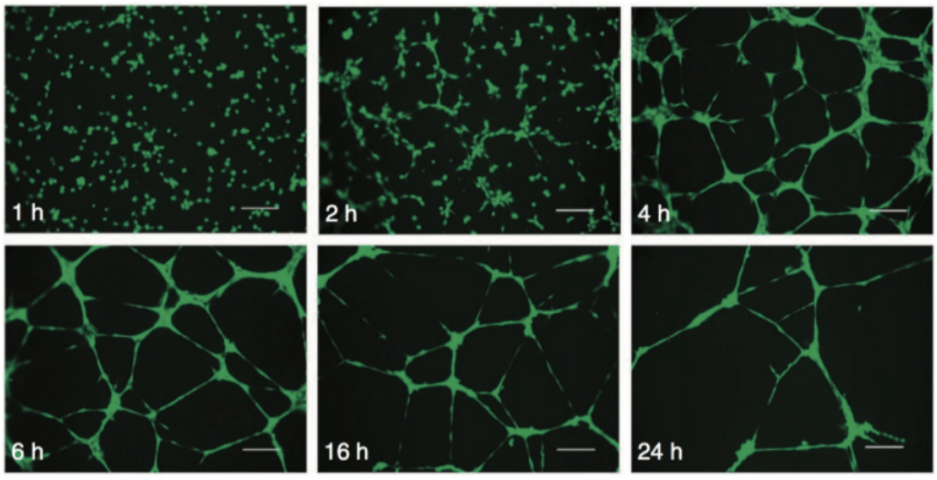


Figure 1: Time series of vasculogenesis, obtained from Nakano et al. [23], showing endothelial progenitor cells initially distributed at random, sprouting into nascent vessel segments, and finally maturing into a stable, interconnected vascular network.

2.1 Cellular Potts Model

The Cellular Potts Model (CPM) is a versatile, stochastic cellular automaton originally introduced to simulate cell sorting [12]. By defining the energy of the system with biologically motivated terms and exploring lower-energy configurations, the CPM can reproduce complex vascular network formation and vasculogenesis [2]. In this framework, each cell occupies a connected cluster of lattice sites distinguished by a unique index. Monte Carlo updates propose reassigning individual sites to neighboring cells. These updates are accepted with a probability that depends on the change in the system's energy, allowing the model to capture both mechanical interactions and stochastic fluctuations. Figure 2 presents a schematic CPM lattice, where each color denotes a different cell and white represents the extracellular medium. Algorithm 1 provides pseudocode for the Cellular Potts Monte Carlo procedure, which we explain step by step below.

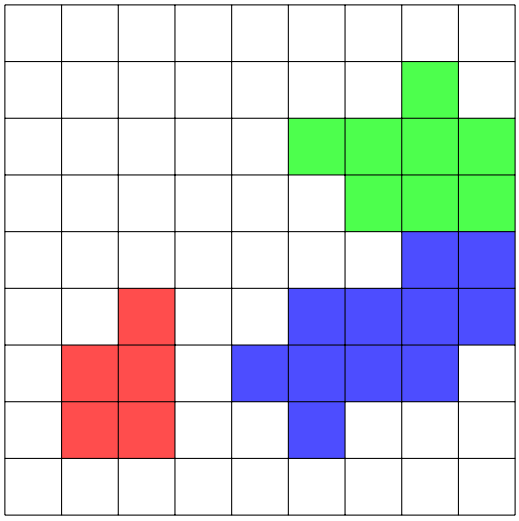


Figure 2: Schematic of a Cellular Potts Model lattice. Each grid site is colored according to its cell identity (distinct hues), with the surrounding medium shown in white. Cell shapes emerge from clusters of grid sites sharing the same label, illustrating how the CPM represents multicellular configurations on a discrete lattice. We identify three cells, comprised of 5, 8 and 11 lattice sites.

CPM Hamiltonian. To compute the acceptance probability for a proposed alteration, we first specify the system's energy via the CPM Hamiltonian. In its simplest form, as introduced by Graner et al. [12], the Hamiltonian comprises two contributions. The first is an area constraint, which penalizes deviations from a target cell area A_{target}, preventing cells from growing arbitrarily large or shrinking to zero. The second is adhesion energy. For every lattice site i we inspect each neighbour $j \in \Gamma(i)$. If i and j belong to different cells, the shared interface incurs a cost $J\left(\tau\left(\sigma_i\right), \tau\left(\sigma_j\right)\right)$, where $\tau(\sigma)$ maps a cell to the corresponding cell type. Together, these terms yield

$$H_{std} = \sum_{\sigma} \nu_A \left(A_\sigma - A_{\text{target}}\right)^2 + \sum_i \sum_{j \in \Gamma(i)} J\left(\tau(\sigma_i), \tau(\sigma_j)\right)\left(1 - \delta(\sigma_i, \sigma_j)\right), \tag{1}$$

where ν_A controls area stiffness, τ labels the cell type of cell σ_x and δ is the Kronecker delta. Note that the left sum is over biological cells, while the right sum is over all lattice sites. In our model of vasculogenesis, we consider only two cell types: the biological cell and the medium. The algorithm now successively selects a random lattice site i and a neighboring site j, and proposes to change the index σ_j of site j to the index σ_i of site i. We then calculate the difference between the Hamiltonian

Algorithm 1 Cellular Potts Monte Carlo loop

Require: lattice Λ with cells $\sigma \subseteq \Lambda$ and size $n = |\Lambda|$, Hamiltonian H, chemotactic field $c(x)$, temperature $T > 0$, number of Monte Carlo steps N_{MCS}

```
for m ← 1 to N_MCS do
    for ℓ ← 1 to n do                                   ▷ one sweep
        Pick random site i ∈ Λ uniformly
        Pick random neighbour j ∈ Γ(i) uniformly
        if σ_i ≠ σ_j then                               ▷ no change
            σ̃_j ← σ_i                                   ▷ proposed copy
            ΔH_std ← H_std(σ̃_j) − H_std(σ_j)
            ΔH_elon ← H_elon(σ̃_j) − H_elon(σ_j)
            ΔH_chem ← −χ[c(j) − c(i)]
            ΔH ← ΔH_std + ΔH_elon + ΔH_chem
            if ΔH < 0 or rand() < exp(−ΔH/T) then   ▷ accept proposed copy
                σ_j ← σ̃_j
                Update cell statistics
            end if
        end if
    end for
end for
```

Note. Each Monte Carlo step performs $n = |\Lambda|$ copy attempts. In each attempt a random lattice site i and neighbour j are selected. If $\sigma_i \neq \sigma_j$ we propose copying σ_i into j. The energy change $\Delta H = \Delta H_{\mathrm{std}} + \Delta H_{\mathrm{elon}} + \Delta H_{\mathrm{chem}}$ is evaluated and the move accepted with Metropolis probability $\min\{1, e^{-\Delta H/T}\}$. Accepted copies update cell statistics and drive the stochastic evolution of cell shapes and contacts.

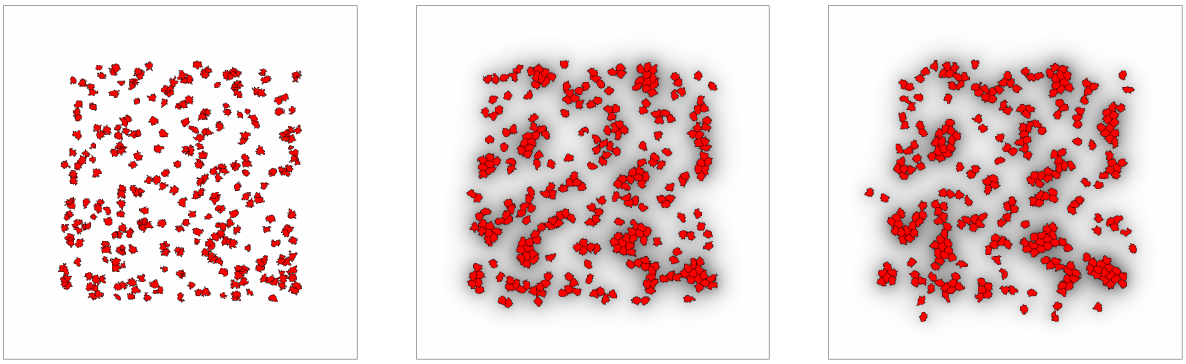
of the original system and that of the proposed system, say ΔH, and accept the proposed change with the following probability:

$$p = \begin{cases} 1 & if \quad \Delta H < 0, \\ \exp(\frac{-\Delta H}{T}) & if \quad \Delta H > 0, \end{cases} \tag{2}$$

where $T > 0$ is the temperature, a parameter inherited from the Boltzmann distribution [20]. It underpins methods across the natural sciences, for instance enthalpy in physics [16], and the Arrhenius equation in chemistry [15]. In those physical contexts T is linked directly to thermodynamic temperature, but in the Cellular Potts Model it functions purely as a dimensionless noise amplifier that regulates the acceptance of energetically unfavorable copy attempts.

In the CPM algorithm, one Monte Carlo step consists of as many proposed updates as there are lattice sites. Successive Monte Carlo Steps gradually drive the system toward lower-energy configurations while preserving the stochastic fluctuations necessary to escape local minima, as seen in (2). To illustrate the behavior of these systems, we simulate the model for the standard Hamiltonian, as described in (1), with 200 cells randomly distributed. We choose the adhesion energy between cells smaller than between a cell and the medium, so we expect cells to clump together. The result for time steps 0, 3000 and 6000 are shown in Figure 3. All CPM simulations have been run using open-source C++ code named Tissue Simulation Toolkit, as introduced by Daub et al. [5].

To drive vascular network formation, we augment the CPM Hamiltonian with two biologically motivated terms [2]. The first is elongation, as cells often adopt an elongated morphology. We capture this by penalizing deviations of each cell's major-axis length L_σ from a preferred length L_{target} :



(a) $t = 0$ (b) $t = 3000$ (c) $t = 6000$

Figure 3: Illustration of CPM simulation with standard Hamiltonian, 200 cells and parameters: $T = 10^{-3}$, $A_{\text{target}} = 50$, $\nu_A = 20$, $J(\sigma_i, \sigma_j) = 40$ if lattice site i belongs to a cell and j belongs to the medium or vise versa, and $J(\sigma_i, \sigma_j) = 20$ if both i and j belong to a cell.

$$H_{elon} = \sum_{\sigma} \nu_\ell \left(L_\sigma - L_{\text{target}} \right)^2,$$

where ν_ℓ sets the strength of the elongation constraint. The second is chemotaxis. To bias cell movement up gradients of a diffusible chemoattractant $c(x)$, we introduce a chemotactic energy contribution, given by:

$$H_{chem} = -\sum_{\sigma} \chi_\sigma \sum_{i \in \Lambda} c(i) \delta_{\sigma(i),\sigma},$$

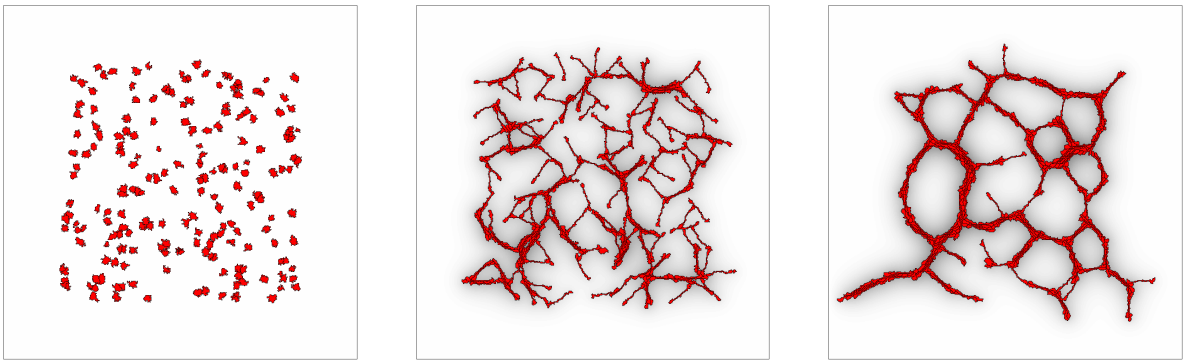
where Λ is the set of all lattice points. For each proposed copy, the only contribution to H_{chem} that changes is the one coming from the single neighbouring lattice site j whose spin is updated. Consequently, the chemotactic part of the energy difference reduces to

$$\Delta H_{chem} = -\chi \left(c\left(j\right) - c\left(i\right) \right),$$

where i and j are the randomly selected lattice sites for the proposed change and χ is the chemotactic sensitivity. Because the model contains only one cell type, every cell shares the same chemotactic sensitivity, therefore we can drop the cell index and write simply χ instead of χ_σ. Moving into higher-concentration regions lowers the Hamiltonian, driving directed migration. Finally, we see the complete hamiltonian $H = H_{std} + H_{elon} + H_{chem}$, and we can calculate the difference for each proposed change

$$\Delta H = \left[H_{std} + H_{elon} \right]_{\text{after}} - \left[H_{std} + H_{elon} \right]_{\text{before}} - \chi_\sigma \left[c\left(j\right) - c\left(i\right) \right].$$

By combining adhesion, area, elongation, and chemotaxis terms, the model recapitulates key features of network formation, yielding realistic vascular networks. We now replicate the experiment as depicted in Figure 3, but with the additional terms included in the Hamiltonian, using additional parameters $\nu_\ell = 60$ and $\chi_\sigma = 10^{-3}$. We obtain Figure 4.



(a) $t = 0$ (b) $t = 1000$ (c) $t = 9000$

Figure 4: Illustration of CPM simulation with extended Hamiltonian, 200 cells and parameters: $T = 10^{-3}$, $A_{\text{target}} = 50$, $\nu_A = 20$, $J(\sigma_i, \sigma_j) = 40$ if lattice site i belongs to a cell and j belongs to the medium or vise versa, and $J(\sigma_i, \sigma_j) = 20$ if both i and j belong to a cell, $\nu_\ell = 60$ and $\chi = 10^{-3}$. We see the current configuration allows for network formation.

Ellipse identification. In the CPM, each cell is represented as a connected collection of lattice sites on a grid. These sites form irregular, potentially highly non-convex shapes due to the CPM's energy minimization rules. To extract a simple geometric description of such cells, we approximate each cell by an ellipse whose parameters best capture the cell. The mathematical foundation for this approximation lies in computing raw and central moments of a discrete pixel set and then fitting the corresponding inertia ellipse via analysis of the covariance matrix [21].

Suppose a cell occupies n lattice sites, each with integer coordinates (x_k, y_k) for $k = 1, \ldots, n$. The first step is to compute the cell's centroid, denoted by $(\bar{x}, \bar{y})$ and computed using the following equation:

$$\bar{x} = \frac{1}{n}\sum_{k=1}^{n} x_k, \quad \bar{y} = \frac{1}{n}\sum_{k=1}^{n} y_k.$$

To quantify how the cell's pixels are distributed around $(\bar{x}, \bar{y})$, we compute the centralized second moments, given by

$$M_{xx} = \sum_{k=1}^{n} (x_k - \bar{x})^2, \quad M_{yy} = \sum_{k=1}^{n} (y_k - \bar{y})^2, \quad M_{xy} = \sum_{k=1}^{n} (x_k - \bar{x})(y_k - \bar{y}).$$

Dividing each by $(n-1)$ gives estimates of variance and covariance. In matrix form, the inertia or covariance matrix is given by the following:

$$\mathbf{C} = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix}, \quad \sigma_{xx} = \frac{M_{xx}}{n-1}, \quad \sigma_{yy} = \frac{M_{yy}}{n-1}, \quad \sigma_{xy} = \frac{M_{xy}}{n-1}.$$

Intuitively, σ_{xx} and σ_{yy} measure how spread out the cell is in the x-and y-directions, while σ_{xy} captures how much those directions are correlated. Any symmetric 2×2 matrix $\mathbf{C}$ can be diagonalized by an orthonormal rotation. Its eigenvalues μ_1, μ_2 are given by the following equation:

$$\mu_{1,2} = \frac{\sigma_{xx} + \sigma_{yy}}{2} \pm \frac{1}{2}\sqrt{(\sigma_{xx} - \sigma_{yy})^2 + 4\sigma_{xy}^2}.$$

Each eigenvalue represents the variance of the pixel cloud along one principal axis. Concretely, the semiaxis lengths a and b of the approximate ellipse are

$$a = \sqrt{\mu_1}, \quad b = \sqrt{\mu_2}.$$

The corresponding eigenvector for μ_1 gives the direction of maximum spread. We can now calculate the angle between the major axis of the ellipse and the horizontal axis, using the following equation:

$$\theta = \tan^{-1}(m), \quad m = \frac{\mu_1 - \sigma_{xx}}{\sigma_{xy}} = \frac{\sigma_{xy}}{\mu_1 - \sigma_{yy}}.$$

2.2 Particle Based Model

To validate the results found with the CPM, Palachanis et al. proposed a different method to reproduce the results using a different simulation paradigm [24]. In this model, cells are represented as ellipse shaped particles, with an inner repulsive and outer attractive ellipse, as depicted in Figure 5. The inner ellipse represents the physical cell, while the outer ellipse is the range of interaction with other cells.

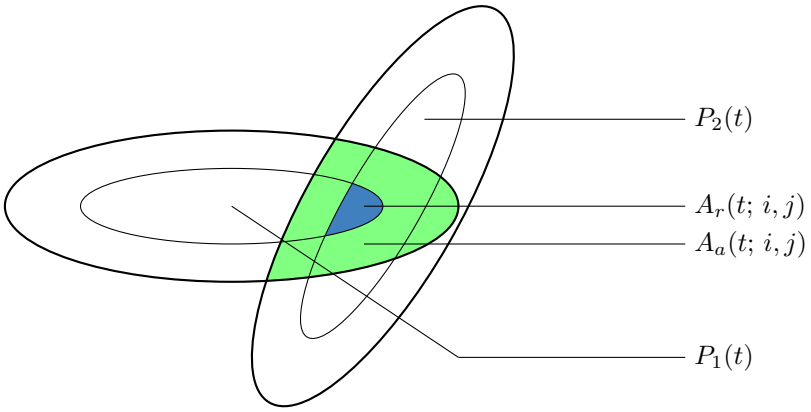


Figure 5: Illustration of ellipse shaped particles $P_1(t)$ and $P_2(t)$, with an inner repulsive and outer attractive ellipse. The attractive overlap $A_a(t;\, i, j)$ is colored green, the repulsive area $A_r(t;\, i, j)$ is colored blue.

The cells are located in the continuous real plane, where the evolution of the system is modeled with the following Langevin equation:

$$m_i \cdot \dot{\vec{v_i}}(t) = -\alpha \vec{v_i}(t) + \sum_{j \neq i} \frac{\vec{x_i}(t) - \vec{x_j}(t)}{r_{ij}(t)} F_{ij}(t) + \vec{\eta}(t), \tag{3}$$

where α is a damping constant, $r_{ij}(t) = ||\vec{x_i} - \vec{x_j}||$ is the distance between cells i and j at time t, $F_{ij}(t)$ is the force between particles i and j at time t and $\vec{\eta}(t)$ is a stochastic noise vector. To facilitate network formation, the force is set to:

$$F_{ij}(t) = \lambda_r A_r(t;\, i, j) - \lambda_a A_a(t;\, i, j), \tag{4}$$

where $A_r(t;\, i, j)$ and $A_a(t;\, i, j)$ are the areas of repulsive and attractive overlap, respectively, at time t between particles i and j, and the stochastic noise vector $\vec{\eta}(t)$ is chosen as the following:

$$\vec{\eta}(t) = N_v \vec{\xi_v}(t) \Delta t^{-1/2}, \tag{5}$$

where $\vec{\xi_v}(t)$ is a randomly determined noise vector at time t and N_v is the magnitude of noise. When running the simulation in discrete time intervals, at every time step the algorithm selects N cells at random and proposes a small reorientation for each. Whether the change is kept is decided by a rule that favors configurations with lower net interaction forces:

$$P_i = \min \left[1, \exp \left\{ \frac{1}{N_a} \left(\sum_{j \neq i} F_{ij}(t) - \sum_{j \neq i} F'_{ij}(t) \right) \right\} \right]. \tag{6}$$

Here $F_{ij}(t)$ and $F'_{ij}(t)$ are the forces between cells i and j before and after the proposed rotation, respectively, and N_a is a normalizing constant. A reorientation that lowers the aggregate force is always accepted, whereas one that raises it is accepted with an exponentially decreasing probability.

In order to simulate the system, we discretize the equations governing its motion. For simplicity, we will only consider the case without noise, but that is easily included in (7). We obtain

$$m_i \cdot \vec{a}_i(t + \Delta t) = -\alpha \, \vec{v}_i(t) + \sum_{j \neq i} \frac{\vec{x}_i(t) - \vec{x}_j(t)}{r_{ij}(t)} \left(\lambda_r A_r(t;\, i, j) - \lambda_a A_a(t;\, i, j) \right), \tag{7}$$

$$\vec{v}_i(t + \Delta t) = \vec{v}_i(t) + \vec{a}_i(t + \Delta t) \, \Delta t, \tag{8}$$

$$\vec{x}_i(t + \Delta t) = \vec{x}_i(t) + \vec{v}_i(t + \Delta t) \, \Delta t. \tag{9}$$

Unless otherwise stated, all cells are assigned unit mass. With noise and reorganisations included, Figure 6 confirms that this framework also reproduces vascular-like networks, despite the two models contrasting foundations. The CPM is a purely stochastic, lattice based model in which cells dynamically change shape. The particle based scheme unfolds deterministically on a continuous plane with each cell represented by a rigid ellipse. Crucially, both models share two key ingredients. In both models cells are elongated and there is a long range attractive interaction. Together, they suffice to drive the emergence of branched, interconnected vascular patterns.

Overdamped simplification. Since numerical differentiation introduces approximation error, we would like to make a simplification that reduces the second order dynamics into a single position-position update rule. Consider the model in (3)–(4) with $\vec{\eta}_i(t) = \vec{0}$ and fixed orientations. For brevity, define

$$S_i(t) := \sum_{j \neq i} \frac{\vec{x}_i(t) - \vec{x}_j(t)}{r_{ij}(t)} \left(\lambda_r A_r(t;\, i, j) - \lambda_a A_a(t;\, i, j) \right).$$

Dividing (3) by α and using the $S_i(t)$ notation gives

$$\frac{m_i}{\alpha} \, \dot{\vec{v}}_i(t) = -\vec{v}_i(t) + \frac{1}{\alpha} S_i(t).$$

In the overdamped regime, where the velocity relaxation time $\tau_i = m_i/\alpha$ is much shorter than the timescale over which the forcing $S_i(t)$ changes, the inertial term can be neglected [4]. This implies that the velocity relaxes almost instantaneously to the force balance:

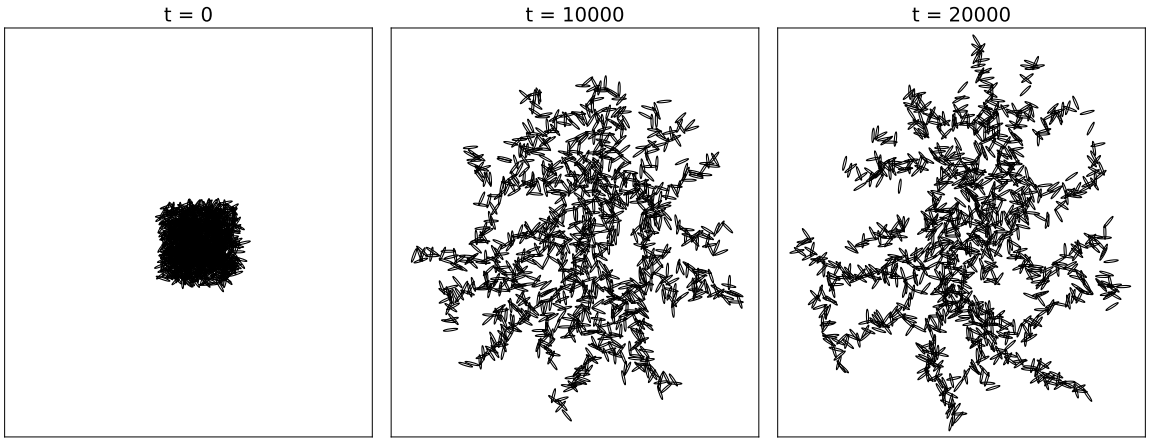


Figure 6: Illustration of PBM simulation with 1000 cells, stochastic noise and reorientations, using parameters $\lambda_r = 0.02$, $\lambda_a = 0.0006$, $N_a = 1$, $N_v = 0.4$, $\alpha = 0.4$ and $\Delta t = 1.0$. The ellipses have major-to-minor axis ratio 7, and outer-to-inner ratio 1.5. The resulting configuration exhibits network formation.

$$\vec{v}_i(t) \approx \frac{1}{\alpha} S_i(t).$$

Using a first-order forward Euler discretization of $\dot{\vec{x}}_i(t) = \vec{v}_i(t)$ then yields

$$\vec{x}_i(t + \Delta t) \approx \vec{x}_i(t) + \frac{\Delta t}{\alpha} S_i(t).$$

This approximation is called the quasi steady state approximation. We will investigate how well this approximation captures the true model for different values of α in Section 4.3.

3 Machine Learning Methods

Machine learning techniques offer a complementary route to first-principles modeling by learning governing dynamics directly from data. In this section we describe such methods. We start with SINDy, whose sparse regression framework extracts concise, interpretable differential equations when an adequate candidate function library is available. We then introduce the sensor based SHRED architecture, which replaces hand-crafted libraries with a shallow recurrent neural network capable of compressing high-dimensional time series into a low-dimensional latent space before reconstructing the full state. Finally, we outline a hybrid SINDy-SHRED strategy that combines SINDy's interpretability with SHRED's representational power, followed by performance metrics required to evaluate all three approaches and the numerical differentiation scheme.

3.1 SINDy

SINDy is a data-driven framework for discovering the underlying differential equations that govern complex systems [3]. Rather than presupposing a specific model, SINDy begins with time series measurements of the system's state while researchers construct a large library of candidate functions that might appear in the governing equations. By leveraging sparse regression techniques, SINDy automatically selects only as little terms as needed to describe the dynamics, thereby yielding an interpretable model. This makes it especially attractive for situations where first-principles derivations are difficult or impossible but abundant data are available.

Applications of SINDy span many fields, ranging from fluid mechanics [10] and epidemiology [18] to neuroscience [17]. In order to use SINDy, we assume the dynamical system of interest can be written as follows:

$$\dot{\vec{x}}(t) = f(\vec{x}(t)),$$

where $\vec{x}(t) \in \mathbb{R}^n$ denotes the state vector at time t and $f : \mathbb{R}^n \to \mathbb{R}^n$ is the vector-valued function governing its evolution. To find the function f, we construct a library of candidate functions $\Theta = (\theta_1, \theta_2, \ldots, \theta_K)$, with each $\theta_i : \mathbb{R}^n \to \mathbb{R}$ representing a potential term in the true dynamics. We then posit that f can be expressed as a linear combination of these candidates. Now consider a dynamical system for which we have m measurements of both the state and time derivative, collected at times $t_1, t_2, \ldots, t_m$. We assemble the data in the following form:

$$X = \begin{bmatrix} \vec{x}(t_1)^\top \\ \vec{x}(t_2)^\top \\ \vdots \\ \vec{x}(t_m)^\top \end{bmatrix} \in \mathbb{R}^{m \times n}, \quad \dot{X} = \begin{bmatrix} \dot{\vec{x}}(t_1)^\top \\ \dot{\vec{x}}(t_2)^\top \\ \vdots \\ \dot{\vec{x}}(t_m)^\top \end{bmatrix} \in \mathbb{R}^{m \times n},$$

where each row corresponds to one time measurement and each column to one state component. Evaluating the candidate library $\Theta = (\theta_1, \ldots, \theta_K)$ at these snapshots yields the following feature matrix:

$$\Theta(X) = \begin{bmatrix} \theta_1(\vec{x}(t_1)) & \ldots & \theta_K(\vec{x}(t_1)) \\ \theta_1(\vec{x}(t_2)) & \ldots & \theta_K(\vec{x}(t_2)) \\ \vdots & & \vdots \\ \theta_1(\vec{x}(t_m)) & \ldots & \theta_K(\vec{x}(t_m)) \end{bmatrix} \in \mathbb{R}^{m \times K}.$$

Each column represents one candidate term, and each row one observation. Assuming the dynamics are a sparse linear combination of these features, we attempt to approximate the following:

$$\dot{X} \approx \Theta(X)\Xi,$$

where the coefficient matrix $\Xi \in \mathbb{R}^{K\times n}$ collects the weights: the j-th column contains the coefficients that govern the j-th state variable. By isolating Ξ's few nonzero coefficients via sparse regression, we aim to uncover the true governing equations. To promote sparsity in the estimated coefficients, we pose the problem as a Least Absolute Shrinkage and Selection Operator (LASSO) problem, instead of simply Ordinary Least Squares. Recall that the coefficient matrix for an n-dimensional system is given by:

$$\Xi = \begin{bmatrix} \vec{\xi}_1 & \cdots & \vec{\xi}_n \end{bmatrix} \in \mathbb{R}^{K\times n},$$

where the i-th column constitutes the coefficient vector for the i-th state vector. For each column, LASSO finds

$$\hat{\xi}_i = \arg\min_{\xi\in\mathbb{R}^K} \frac{1}{2\cdot m}\left\|\Theta(X)\xi - \dot{\vec{x}}_i\right\|_2^2 + \kappa\|\xi\|_1, \tag{10}$$

with $\dot{\vec{x}}_i$ the i-th column of $\dot{X}$, m the number of time steps considered, $\kappa > 0$ the regularization parameter and the factor $\frac{1}{2\cdot m}$ added for numerical stability. The ℓ_1-penalty induces sparsity by imposing a penalty for the absolute size of coefficients. There are many algorithms that can solve the minimization problem in (10), we will use coordinate descent [9].

Example. We now present a simple example to illustrate all previous notation. Consider the system of ODE's given by $\dot{x} = -2y$ and $\dot{y} = x$, with initial condition $(1, 1)$ at $t = 0$ and candidate library $\Theta = (1, x, y, x^2, xy, y^2)$. An analytical notation reads $x = \cos(2t) - 2\sin(2t)$ and $y = \frac{1}{2}\sin(2t) + \cos(2t)$. Let us review the system at $t_1 = 0$ and $t_2 = \frac{1}{4}\pi$. This results in the following data:

$$X = \begin{bmatrix} 1 & 1 \\ -2 & 1/2 \end{bmatrix} \in \mathbb{R}^{2\times 2}, \quad \dot{X} = \begin{bmatrix} -2 & 1 \\ -1 & -2 \end{bmatrix} \in \mathbb{R}^{2\times 2},$$

and candidate library:

$$\Theta(X) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -2 & 1/2 & 4 & -1 & 1/4 \end{bmatrix}.$$

Finally, we can estimate the coefficient matrix Ξ that results in the least loss:

$$\dot{X} \approx \Theta(X)\Xi \quad \Longleftrightarrow \quad \begin{bmatrix} -2 & 1 \\ -1 & -2 \end{bmatrix} \approx \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -2 & 1/2 & 4 & -1 & 1/4 \end{bmatrix} \begin{bmatrix} \xi_{1,1} & \xi_{1,2} \\ \xi_{2,1} & \xi_{2,2} \\ \xi_{3,1} & \xi_{3,2} \\ \xi_{4,1} & \xi_{4,2} \\ \xi_{5,1} & \xi_{5,2} \\ \xi_{6,1} & \xi_{6,2} \end{bmatrix}.$$

The linear system defined by this approximation does not have a unique solution. By imposing an ℓ_1-norm penalty, however, we single out the solution with the smallest ℓ_1 norm, which in turn promotes sparsity in the coefficient vector. The solution with minimal ℓ_1 norm is $\xi_{3,1} = -2$, $\xi_{2,2} = 1$ and $\xi_{ki} = 0$ for all other k and i. The first column in the coefficient matrix, corresponding to the equation for x, contains a single non-zero entry in the row for candidate function θ_3, with weight -2. Likewise, the

second column, corresponding to the equation for y, is non-zero only in the row for candidate term θ_2, with weight 1. This gives us the equations: $\dot{x} = -2\theta_2 = -2y$ and $\dot{y} = \theta_3 = x$, which are correct.

In practice, however, the sparse regression at the heart of SINDy does not always result in the correct solution. ODE identifiability theory shows that without sufficiently diverse trajectories, multiple parameters can approximate the data. In such cases, the standard regularized regression can converge to an incorrect minimum, fitting the data better than the true solution because of numerical errors. Mitigating this degeneracy requires enriching the dataset with for example multiple initial conditions or more trajectories.

While SINDy has proven effective for extracting concise, interpretable models in many settings [3], it relies on relatively simple statistical tools which can struggle when the true dynamics involve strongly nonlinear interactions, high noise levels, or complex, high-dimensional data. Moreover, the selection of the candidate-function library is critical and represents an additional potential source of error. To address these limitations we also explore more powerful machine-learning approaches that can capture richer, nonlinear relationships and incorporate spatio-temporal features. The following section describes these techniques and explains how they complement SINDy by offering greater flexibility for more challenging data sets.

3.2 SHRED

SHallow Recurrent Encoder-Decoder (SHRED) is a compact neural-sensing framework that turns a small set of time-resolved, possibly noisy sensor signals into an accurate picture of a system's full state [34]. Introduced as a sensing architecture for physical systems, SHRED networks couple a Recurrent Neural Network (RNN) with a Shallow Decoder Network (SDN). The recurrent component compresses the time-history of sparse, possibly noisy, sensors into a low-dimensional latent vector, while the decoder maps this vector back to the high-dimensional state space. Because the overall network is shallow, it trains rapidly and mitigates over-fitting. Benchmarks on turbulent-flow and other chaotic systems show that SHRED improves reconstruction and forecasting errors compared to previous methods. These qualities make SHRED an attractive candidate for predicting cell trajectories in vascular network formation.

In our SHRED architecture we employ a Long Short-Term Memory (LSTM) network as the recurrent encoder [13]. First introduced by Hochreiter et al. to solve problems with standard recurrent neural networks, LSTMs have become the standard in sensing and forecasting tasks [26]. Recently it has achieved success within the SHRED framework of Williams et al. [34]. The network is organized as a stack of LSTM layers. Each layer ingests the full hidden sequence from its predecessor and forwards a newly calculated hidden state. Figure 7 depicts a two-layer configuration identical to the one we adopt in this study.

Each LSTM cell maintains two complementary memories: the cell state c_t, which stores long-term information, and the hidden state h_t, a short-term context passed to subsequent layers. Together with the current input z_t, they generate three gates and a candidate update. They do so according to the subsequent rules:

$$
\begin{aligned}
f_t &= \sigma\left(W_f z_t + U_f h_{t-1} + b_f\right), && \text{(forget)} \\
i_t &= \sigma\left(W_i z_t + U_i h_{t-1} + b_i\right), && \text{(input)} \\
o_t &= \sigma\left(W_o z_t + U_o h_{t-1} + b_o\right), && \text{(output)} \\
\tilde{c}_t &= \tanh\left(W_c z_t + U_c h_{t-1} + b_c\right), && \text{(candidate)}
\end{aligned}
$$

where W_*, U_*, and b_* are trainable weights and biases and σ is a sigmoid function. The long-term memory is updated by blending the retained past with the candidate content, performing the following calculation:

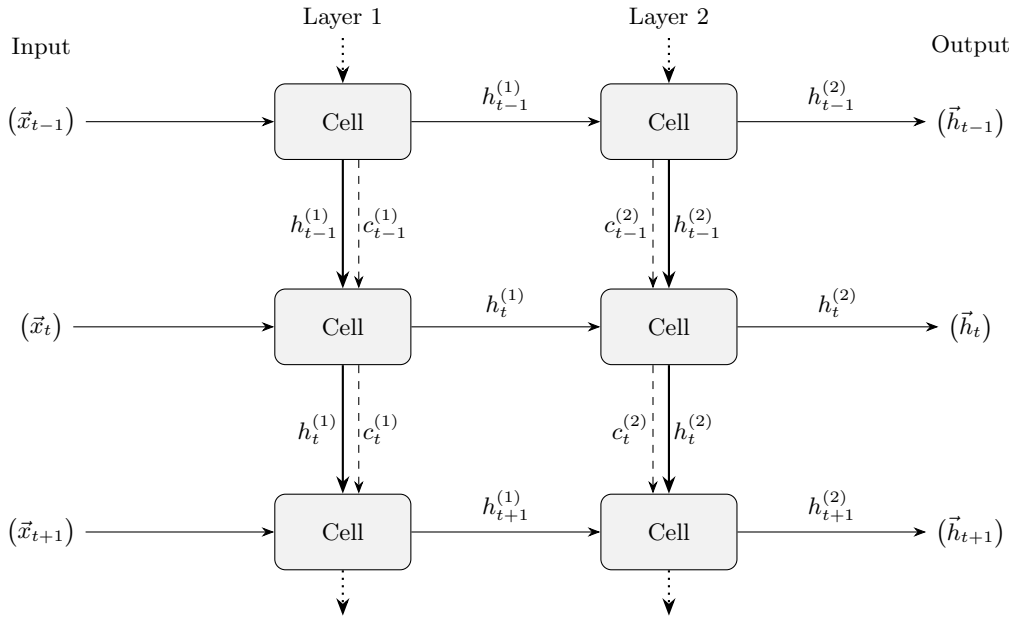


Figure 7: Two-layer LSTM unrolled in time. At each time step t an input vector $\vec{x}_t \in \mathbb{R}^k$ enters the first LSTM layer. Its hidden output $\vec{h}_t^{(1)} \in \mathbb{R}^l$ becomes the input to a second LSTM layer, whose hidden state $\vec{h}_t^{(2)} \in \mathbb{R}^m$ is taken as the network output $\vec{h}_t$. Solid vertical arrows show the hidden-state flow $h_{t-1}^{(l)} \to h_t^{(l)}$, dashed arrows the cell-state flow $c_{t-1}^{(l)} \to c_t^{(l)}$. Dotted extensions indicate that the sequence continues beyond the three illustrated time steps.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t,$$

where $\odot$ represents the element wise product between vectors. The information revealed to the rest of the network is calculated as follows:

$$h_t = o_t \odot \tanh\left(c_t\right).$$

Intuitively, the forget gate f_t decides what to erase from c_{t-1}, the input gate i_t selects what new information to write, and the output gate o_t governs how much of the updated memory is exposed. Because the update of c_t is additive rather than purely multiplicative, the network is able to capture long-range dependencies while keeping the parameter count modest [13]. A diagram of a single LSTM cell is shown in Figure 8.

After compressing the past observations into a latent vector, SHRED passes this compact representation through a shallow decoder that lifts it back to the full state space, delivering a forecast of the next configuration. In practice the decoder consists of several connected layers, interleaved with ReLU activations. This minimal depth keeps the parameter count low and speeds training. Because LSTMs rely on stored hidden-state information, they require a sequence of past data points to generate an accurate prediction, referred to as lag. The general convention is to set the previous hidden states to 0 for the first time step.

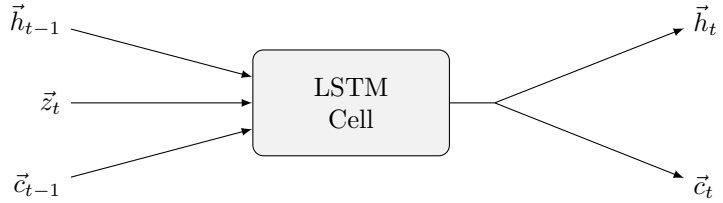


Figure 8: Diagram of a single LSTM cell showing all inputs and outputs. The hidden state $\vec{h}_t$ is forwarded to subsequent layers, while both $\vec{h}_t$ and $\vec{c}_t$ are used in the next time step. The vector $\vec{z}_t$ is comprised of the input vector or hidden state of the previous layer.

3.2.1 SHRED-ROM

State spaces can be so large that mapping onto it is infeasible. For this reason, Tomasetto et al. have proposed a combination of the SHRED network with classical reduced order modeling (ROM) techniques [32], named SHRED-ROM. This method was originally proposed in the context of PDE's, where it is for example used in fluid dynamics. A proper orthogonal decomposition (POD), connected to principal component analysis in statistics, first compresses a collection of high-fidelity snapshots into a low-rank basis that retains the dominant modes of the system. The SHRED network is then trained to map the input vector directly to the coefficients of those POD modes. By mapping onto the coefficients of the modes rather than the total state space, the size of the network is reduced by several orders of magnitude.

POD seeks to represent a spatio-temporal field as a sum of mutually orthogonal modes, each of which is purely spatial and is modulated by a corresponding time coefficient. For a field $y : \mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R}^k$ that maps an n-dimensional spatial point to an k-component state, POD approximates

$$y(\vec{w}, t) \approx \sum_{j=1}^{r} u_j(\vec{w}) a_j(t),$$

where $u_j(\vec{x})$ are POD modes and $a_j(t)$ are their corresponding, scalar time coefficients. For a given field y, suppose we collect spatial locations $x_1, \dots x_n$ at times $t_1, \dots t_m$. We can collect this data in an $n \times m$ matrix, by writing

$$Y = \begin{bmatrix} y(w_1, t_1) & y(w_1, t_2) & \cdots & y(w_1, t_m) \\ y(w_2, t_1) & y(w_2, t_2) & \cdots & y(w_2, t_m) \\ \vdots & \vdots & \ddots & \vdots \\ y(w_n, t_1) & y(w_n, t_2) & \cdots & y(w_n, t_m) \end{bmatrix}.$$

Applying Singular Value Decomposition (SVD), as described by Delft University of Technology [7], yields

$$Y = U \Sigma V^\top,$$

where $U \in \mathbb{R}^{n \times n}$ contains the spatial POD modes, $\Sigma \in \mathbb{R}^{n \times m}$ is a matrix with only non-zero entries on the main diagonal, sorted in descending order and called singular values, and $V \in \mathbb{R}^{m \times m}$ holds the associated time-coefficient vectors. This decomposition contains all collected information about the field y, but we would like to condense it without exceeding a specified error bound. Because the singular values σ_i are ordered by magnitude, the first column of U, paired with σ_1 and the first row

of $V^\top$, captures the largest share of the field's variance. Each successive column accounts for the next most energetic pattern. Retaining only the leading r columns of U and rows of $V^\top$ therefore yields the optimal rank r approximation of the data, as guaranteed by the Eckart-Young-Mirsky theorem [8].

After truncation of the matrices, we retain $U_r \in \mathbb{R}^{n\times r}, \Sigma_r \in \mathbb{R}^{r\times r}$ and $V_r \in \mathbb{R}^{m\times r}$. The columns of U_r form the reduced spatial basis, while the rows of $\Sigma_r V_r^\top$ supply the associated time coefficients $a_j(t)$. In SHRED-ROM these coefficients become the learning target. The recurrent encoder ingests the past k snapshots and the shallow decoder outputs the next coefficient vector. A full-field prediction is recovered a-posteriori by reversing the POD.

Because the POD only retains the most important modes, the network size is reduced by several orders of magnitude relative to a decoder that reconstructs all n state variables directly. The choice of r balances accuracy and efficiency. Note that the model training happens in two stages: we first extract an optimal POD basis from spatio-temporal snapshots and then train the SHRED network to predict the corresponding modal coefficients.

3.2.2 SINDy-SHRED

While SHRED-ROM compresses the full state via a POD basis, SINDy-SHRED instead learns a low-dimensional latent coordinate system in which the governing dynamics are themselves sparse and interpretable. Originally proposed by Gao et al. [11], SINDy-SHRED combines SHRED with SINDy to jointly solve sensing, reconstruction, and model discovery from sparse measurements.

Concretely, a gated recurrent unit ingests a short history of sensor readings and produces at each time t a latent state vector $z_t \in \mathbb{R}^r$. A shallow decoder then reconstructs the full field $\hat{x}_t$ from z_t. To regularize the latent dynamics, SINDy-SHRED enforces

$$z_{t+1} \approx z_t + \Theta(z_t)\Xi\Delta t,$$

where $\Theta(z)$ is a library of candidate functions and Ξ the sparse coefficient matrix to be discovered. Training minimizes the composite loss

$$\mathcal{L} = \underbrace{\| x_t - \hat{x}_t \|_2^2}_{\substack{\text{(i) decoder}\\ \text{reconstruction}}} + \underbrace{\| z_{t+1} - \big(z_t + \Theta(z_t)\Xi\,\Delta t\big) \|_2^2}_{\substack{\text{(ii) dynamics of}\\ \text{the latent space}}} + \underbrace{\kappa\|\Xi\|_0}_{\substack{\text{(iii) sparsity}\\ \text{constraint}}} ,$$

where x_t is the true state vector, $\hat{x}_t$ is the decoder output, z_t the latent representation of the input and κ the regularization parameter. We now use the ℓ_0 norm instead of the ℓ_1 norm for regularization, meaning we simply count the number of non-zero entries. The result is an end-to-end model that (i) reconstructs high-dimensional fields from a handful of sensors, (ii) yields a parsimonious ODE $\dot{z} = f(z)$ governing the latent space, and (iii) affords interpretability. By combining SINDy and SHRED, this method enables both forecasting and model recovery in the latent space.

3.3 Metrics for Model Performance

To quantify how well the inferred model captures the underlying dynamics, we consider a set of commonly used performance metrics. Each of these metrics evaluate either how accurately the model reproduces the velocity and positional data or how well it recovers the parameters governing the underlying differential equation.

1. **Mean Squared Error (MSE)**: We employ this metric to quantify two distinct errors. Firstly, we utilize it to assess the accuracy of the predicted velocities with respect to the true velocities. Secondly, with it we evaluate how closely the estimated coefficients approximate the true coefficients. For the difference in velocities, we define it as:

$$\mathrm{MSE}_x = \frac{1}{m \cdot n} \sum_{j=1}^{m} \sum_{i=1}^{n} \left\| \dot{x}_i(t_j) - \dot{x}'_i(t_j) \right\|_2^2,$$

where n and m are the numbers of particles and time points, respectively, $\dot{x}_i(t_j)$ is the true velocity of particle i at time t_j and $\dot{x}'_i(t_j)$ is the inferred velocity for that particle and time. A lower MSE_x indicates that the inferred velocities better match the observed data.

For the difference in coefficients, let $\vec{\xi'}$ be a vector with the inferred parameters as coefficients and $\vec{\xi}$ the true parameter vector. We compute this mean squared error as follows:

$$\mathrm{MSE}_\xi = \frac{1}{K} \sum_{k=1}^{K} \left(\xi'_k - \xi_k \right)^2.$$

Here, a smaller value indicates that the inferred parameters closely match the true parameters, signifying that the differential equation governing the system has been accurately identified.

2. **Coefficient of Determination (R^2)**: To evaluate the predictive performance compared to a simple baseline, we use the R^2 metric, defined by:

$$R^2 = 1 - \frac{\sum_{j=1}^{m} \sum_{i=1}^{n} \left\| \dot{x}_i(t_j) - \dot{x}'_i(t_j) \right\|_2^2}{\sum_{j=1}^{m} \sum_{i=1}^{n} \left\| \dot{x}_i(t_j) - \overline{\dot{x}_i(t_j)} \right\|_2^2},$$

where $\dot{x}_i(t_j)$ and $\dot{x}'_i(t_j)$ are the observed and model-predicted velocities of particle i at time t_j, respectively and $\overline{\dot{x}_i(t_j)}$ is the global mean true velocity used as the baseline in the denominator. An R^2 value close to 1 indicates that the model explains most of the variability in the data, whereas an R^2 near 0 indicates that the model performs similarly to simply predicting the mean velocity.

3. **Mean Displacement Error (MDE)**: Small differences in estimated velocities can lead to big differences in estimated positions. It is therefore important that we evaluate how close the simulated positions are to the ground-truth positions at a certain time t. To this end, let $x_j(t)$ be the true position of particle j at the time t, and let $x'_j(t)$ be the position obtained when the system is simulated with the inferred parameters. The Mean Displacement Error is then given by

$$\mathrm{MDE}(t) = \frac{1}{n} \sum_{i=1}^{n} \left\| x_i(t) - x'_i(t) \right\|_2,$$

where n is the number of particles. A smaller value of MDE implies that the simulated positions using the inferred coefficients closely match the true positions of the particles.

From the definitions, we see that these metrics quantify different aspects of performance. Specifically, the MSE_x and R^2 measure how closely the inferred velocities match the observed data, the MSE_ξ evaluates the accuracy of the recovered differential equation coefficients, and the $\mathrm{MDE}(t)$ assesses how well the inferred model captures the positions in a simulated scenario at a certain time t. Consequently, these metrics may not always align, and one may need to prioritize which metric to optimize based on the specific objective at hand.

3.4 Estimating Velocities

Given a set of positional data, we need to estimate the velocities of particles for all aforementioned methods. Given the positions of the particles at time points $t_1, t_2, \ldots, t_m$, we estimate the velocities at time points $t_2, t_3, \ldots, t_{m-1}$ by using second order accurate central differences, as described by

$$\dot{x}(t_i) \approx \frac{x\left(t_{i+1}\right) - x\left(t_{i-1}\right)}{2h}. \tag{11}$$

For the boundary points we use forward and backward differences, as given by

$$\dot{x}(t_1) \approx \frac{x(t_2) - x(t_1)}{h}, \tag{12a}$$

$$\dot{x}(t_m) \approx \frac{x(t_m) - x(t_{m-1})}{h}. \tag{12b}$$

In finite difference schemes such as these, the step size plays a decisive role in balancing truncation and round-off errors. When the step size is excessively large, the neglected higher-order terms produce significant truncation errors that degrade accuracy. Conversely, overly small step sizes introduce substantial floating-point round-off errors, compounded by repeated calculations [33]. It is therefore important to balance between these sources of error, to ensure reliable derivative estimates.

4 Retrieving the System

We now want to investigate whether it is possible to approximate cell movement in the CPM model using the PBM as described by (7) in Section 2.2. The models are very different in nature, one purely stochastic lattice based and the other governed by an ODE. To this end, we first apply SINDy to simpler examples to validate the methodology in Section 4.1 and Section 4.2. For this we use the same systems as in the thesis by Pham [26]. Successfully recovering the known models in these simpler settings verifies that our numerical differentiation, library construction, and sparse regression routines are correctly implemented before we tackle the more complex CPM data. We first infer the PBM from the ellipse based simulation data in Section 4.3. After that, we apply the identical inference pipeline to the CPM generated data in Section 4.4.

4.1 1D Particle System

We first consider a simple one-dimensional particle system, where each particle is represented by a point on the real line. The particles attract and repulse each other with a force proportional to the distance between them. Formally, let the ODE governing the motion of the particles be given by:

$$\dot{x}_i(t) = \sum_{j \neq i} \frac{x_i(t) - x_j(t)}{r_{ij}(t)} F_{ij}(t), \quad F_{ij}(t) := -k\left(r_{ij}(t) - \rho\right), \tag{13}$$

where $x_i(t)$ is the position of particle i, $r_{ij}(t) = |x_i(t) - x_j(t)|$ is the distance between particles i and j and $F_{ij}(t)$ is the force between particles i and j at time t. Furthermore, k and ρ are positive real constants. Since the force describes the particles' velocities rather than their accelerations, the resulting equation of motion is first-order in time. Consequently, this force functions as a damping term rather than a classical spring force.

We generate a dataset of particle positions and use SINDy to identify the governing differential equation. Figure 9 depicts the evolution of the particle positions in a simulation. Consistent with theoretical expectations, the particles reach an equilibrium configuration at which their displacements become negligible.

In SINDy we postulate that each velocity $\dot{x}_i$ can be written as a sparse linear combination of chosen basis functions. For the one-dimensional particle system we consider the following sum:

$$\dot{x}_i(t) = \sum_{k=0}^{K} \xi_k \underbrace{\sum_{j \neq i} \frac{x_i(t) - x_j(t)}{r_{ij}(t)} r_{ij}^k(t)}_{k\text{-th candidate function}}, \tag{14}$$

where K is the maximum exponent of the distance between particles considered, and ξ_k the coefficients we try to estimate. Decomposing the true dynamics given in (13) into this basis reveals that only the first two coefficients are non-zero, i.e.,

$$\xi_0 = k\rho, \quad \xi_1 = -k, \quad \xi_k = 0 \text{ for } k \geq 2.$$

Thus the original interaction law is fully captured by the $k = 0$ and $k = 1$ terms, and a successful sparse regression should recover exactly this pattern of coefficients.

As a first experiment, we demonstrate the deficiency of the R^2 metric when selecting the regularization parameter κ in LASSO regression. In Figure 10, we plot both the R^2 and MSE_ξ metrics for $K = 5$ and $K = 8$ for different κ. When K equals 5, the performance for the two metrics align for all considered

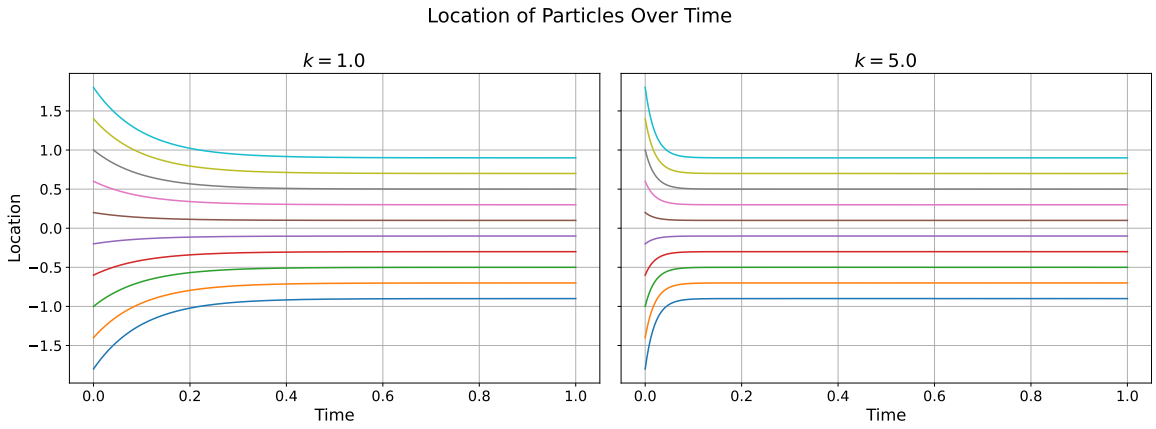


Figure 9: Simulated locations of the particles with (13), using $k = 2$ and $\rho = 1$ with initial positions equally distributed on the interval $[-1.8, 1.8]$. Particles converge to an equidistant steady state distribution, with faster convergence for greater k.

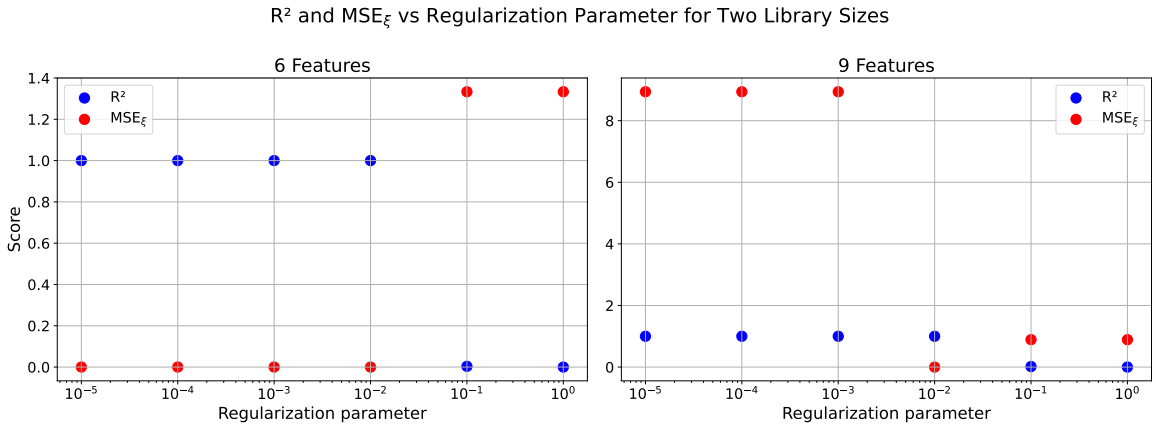


Figure 10: A comparison of the R^2 and MSE_ξ metric for different values of the regularization parameter κ and different sizes of the function library. We set $m = 10$, $n = 1000$, $k = 2$, $\rho = 1$, and simulate the system on the interval $[0, 1]$ with initial positions randomly distributed.

κ. However, for $K = 8$ and regularization values below 10^{-2}, we observe a discrepancy between the R^2 metric and the MSE_ξ. In a real-world setting, where the true parameters are unknown, one would be tempted by the excellent R^2 to accept the spurious model. The MSE_ξ, however, makes it clear that the underlying dynamics have been misidentified. This deviation highlights that the R^2 metric alone is not a reliable indicator of correct model recovery. It is therefore important to consider a diverse set of trajectories, as further explored for a two dimensional system in Section 4.2.

Another crucial parameter is the number of time steps in a fixed time interval, which in turn determines the time between consecutive steps. Owing to the model's deterministic nature, we expect that once the time series length surpasses a sufficient threshold, adding more time steps will no longer change the inferred parameters. Referring to the velocity-estimation approach described in Section 3.4, we explore the importance of step size. Using the same underlying model but altering the step size, we plot the MSE_ξ and R^2 metrics against the number of time steps within $[0, 1]$ in Figure 11. The result underscores the importance of choosing a sufficiently small step size. We hypothesize that the poor performance for relatively large time steps is primarily due to inaccuracies in the derivative estimation,

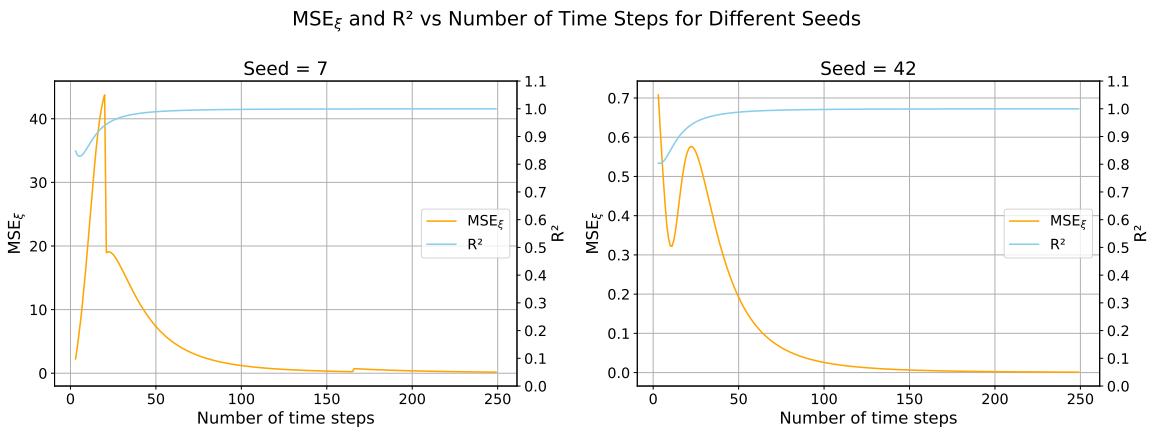


Figure 11: A comparison of the R^2 and MSE_ξ metric for different numbers of time steps on the interval $[0, 1]$ with $m = 10$, $k = 2$, $\rho = 1$, $\kappa = 0.01$ and $K = 8$ for different seeds. We see that the accuracy increases inversely proportional to the time step size.

which become more pronounced as the step size increases.

4.2 2D particle System

Let us now consider a more complex particle system, where each particle is represented by a point on the real plane. We choose the ODE governing the motion of the particles as

$$\dot{\vec{x}}(t) = \sum_{j \neq i} \frac{\vec{x}_i(t) - \vec{x}_j(t)}{r_{ij}(t)} F_{ij}(t), \quad F_{ij}(t) := k_1 \left(r_{ij}(t) - \rho\right)^3 - k_2 \left(r_{ij}(t) - \rho\right), \tag{15}$$

where as before $\vec{x}_i(t)$ is the position of particle i, $r_{ij}(t) = ||\vec{x}_i(t) - \vec{x}_j(t)||$ is the distance between particles i and j and $F_{ij}(t)$ is the force between particles i and j at time t. Moreover, k_1, k_2 and ρ are positive real constants. As in the one dimensional case, we need to assume a form for the equation governing the motion of the particles to apply LASSO. We assume the same form as described in (14), but now the positions and velocities are two dimensional vectors. This results in

$$\dot{\vec{x}}_i(t) = \sum_{k=0}^{K} \xi_k \sum_{j \neq i} \frac{\vec{x}_i(t) - \vec{x}_j(t)}{r_{ij}(t)} r_{ij}^k(t). \tag{16}$$

As a first experiment we simulate the particle system with different initial conditions. Using identical parameters in both simulations, we obtain Figure 12, where trajectories are discretized into 100 time points, with each particle's position marked by a dot. In both panels, particles start with high velocities that rapidly decline. Using SINDy, we estimate the governing differential equation for both sets of trajectories using $\kappa = 0.01$. The MSE_ξ for the left figure is 0.03, while the MSE_ξ for the right figure is 3.21. This indicates that the inferred model from the left figure is much closer to the true model than the inferred model from the right figure. Therefore, the accuracy of the retrieved system seems to depend on the initial configuration of the cells. For this reason, we hypothesize that considering multiple distinct initial configurations is important when inferring the system.

Before exploring that further, we extend the simulation to 100 particles over a longer time horizon with parameters $k_1 = 0.2$, $k_2 = 2$, $\rho = 3$ and $t \in [0, 50]$ in Figure 13 confirms that this interaction law appears unable to generate vascular-like networks. Instead, the particles relax into a highly symmetric

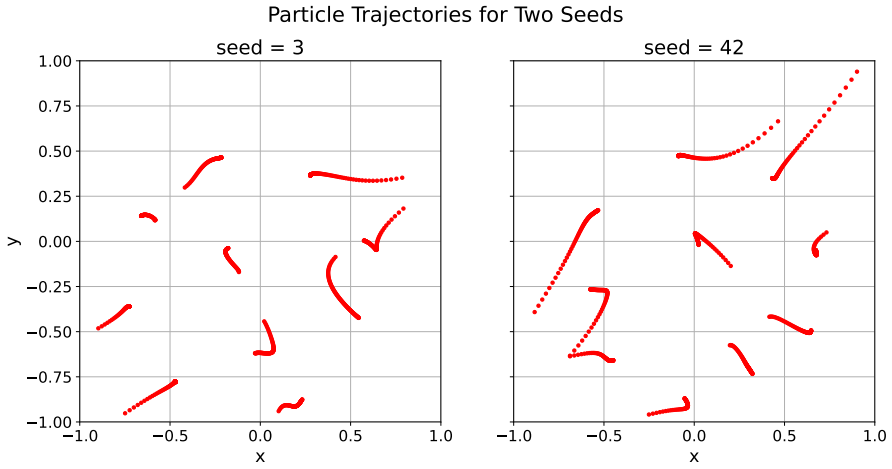


Figure 12: Particle trajectories were simulated with (15) from distinct initial configurations with particles randomly distributed on $[-1,1] \times [-1,1]$, using $k_1 = 0.8$, $k_2 = 2$, $\rho = 1$ and $t \in [0,1]$. Discretizing the time frame into 100 time steps, the parameter estimation from the trajectories shown in the left figure achieved a much better fit compared to the right trajectories, with an MSE_ξ of 0.03 versus 3.21. A regression coefficient of $\kappa = 0.01$ was used in model recovery.

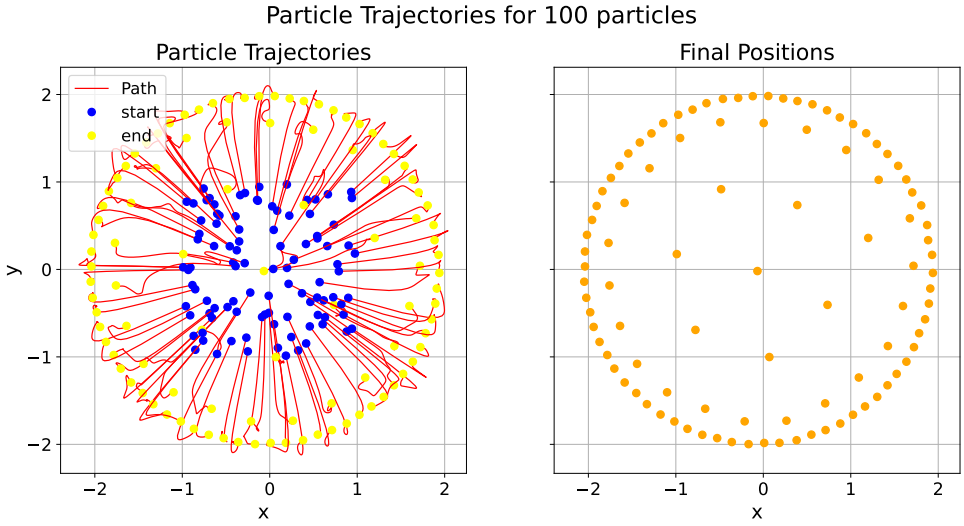


Figure 13: Simulation of 100 particles with parameters $k_1 = 0.2, k_2 = 2$, $\rho = 3$, $t \in [0,50]$ and initial positions randomly distributed in $[-1,1] \times [-1,1]$, showing the particles reach a highly symmetric equilibrium. Figure 1 shows not all vascular networks are symmetrical, illuminating that this model is insufficient for network formation.

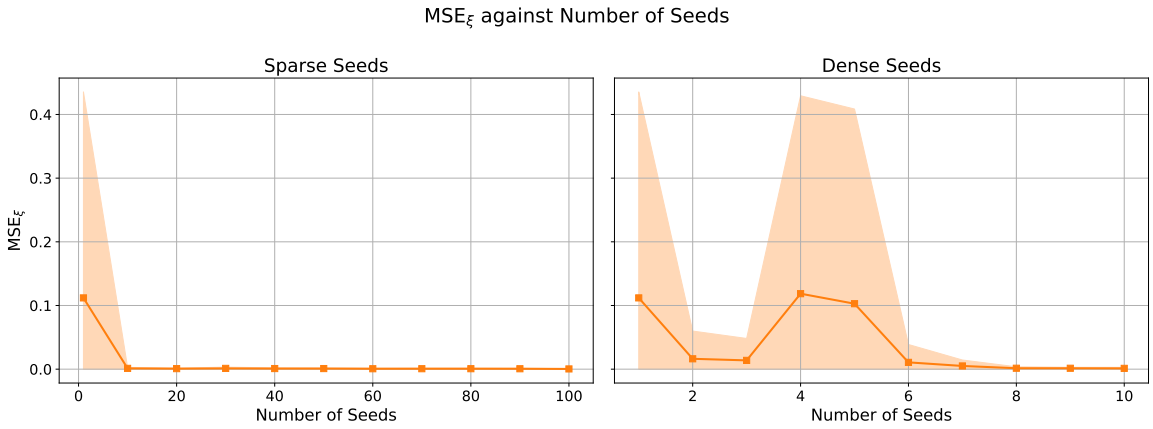


Figure 14: MSE$_\xi$ of the identified dynamics as a function of the number of different initial configurations included in training data. 10 distinct sets of 100 seeds were considered to generate the mean and standard deviation. The left panel explores wider seeding, from 0 to 100 seeds with interval 10, while the right panel zooms in, depicting the MSE$_\xi$ for 1 through 10 number of initial configurations. The left figure indicates considering more than 10 different initial configurations provides little additional value, while the right figure shows the value of considering multiple initial conditions.

equilibrium, much like the one dimensional case, without any branching or clustering. This reinforces that additional mechanisms are required to break symmetry and drive genuine network formation.

To probe how strongly model recovery depends on the number of starting configurations considered, we repeat the following experiment across a broad set of initial states. We generate 100 initial configurations and replicate this procedure ten times, giving ten independent data sets. For each configuration, we simulate particle trajectories on the time interval $[0, 1]$ and record the data on 100 uniformly distributed time steps. We then apply SINDy to an increasing number of initial configurations. Figure 14 plots the average MSE$_\xi$ over the ten data sets, with one standard-deviation bands.

The left panel of Figure 14 samples the number of seeds in coarse steps of ten. The MSE$_\xi$ falls sharply and tends to zero once ten distinct initial configurations are included, indicating that additional seeds beyond this threshold add little value. To investigate the critical region more finely, the right panel shows every seed count from 1 to 10. Here the error decays less steadily, but plateaus close to zero at about eight seeds, underscoring that a modest diversity of starting states is needed for accurate model identification. These results indicate that around 10 different initial configurations need to be considered if sufficiently small time steps are infeasible.

To investigate the effect of the number of time steps on model recovery, we perform the following experiment. For 1000 different initial configurations, we calculate the MSE$_\xi$ and R^2 metric. We then calculate the mean and standard deviation of these metrics. The results are shown in Table 1. For larger time steps, we see that the MSE$_\xi$ indeed has significantly large mean and standard deviation. This indicates a large dependence on a favorable initial configuration for the regression algorithm to recover the underlying differential equation. The R^2 metric, however, is still quite high. This indicates that the model is still able to explain a large part of the variance in the data. This example vividly illustrates the identifiability issue discussed in Section 3.1. When inference is performed on a limited set of trajectories, multiple distinct parameter vectors can reproduce the observed particle velocities with approximately equal accuracy.

To evaluate how measurement noise affects our identification pipeline, we add independent, zero-mean Gaussian noise $\mathcal{N}(0, 1/100)$ to every simulated cell position before computing finite-difference

Table 1: Performance Metrics for Different Time Steps

Time Steps	Metric	Mean	Standard deviation
100	R^2	0.9994493742352435	0.0004387105315007332
	MSE_ξ	0.34123887121183555	0.8293812867113606
1000	R^2	0.9998300869558776	0.000697325479596244
	MSE_ξ	0.06727480301467423	0.20352897097189157

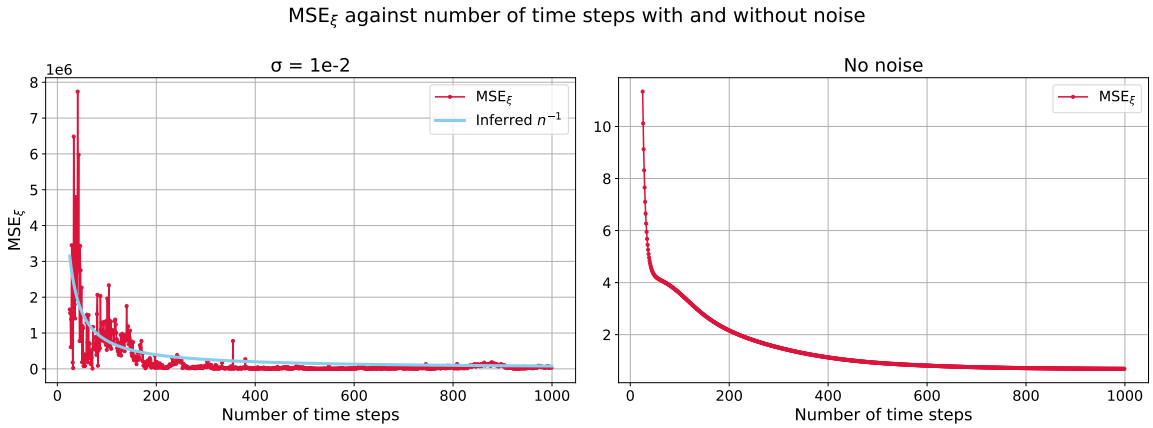


Figure 15: Effect of Gaussian measurement noise on coefficient recovery. Independent zero-mean noise with variance 10^{-2} is added to every simulated cell position before numerical differentiation. The plot shows the parameter-error MSE_ξ versus the number of snapshots n. The dashed line is the Ordinary Least Square best-fit n^{-1} trend. Despite the noise being injected before derivative computation, the observed error still decays close to the best fit n^{-1} rate.

derivatives and assembling the feature matrix. This emulates real microscopy data, where tracking errors arise at the measurement stage due to imprecise measurements. We then train the model on trajectories of increasing length and record the resulting MSE_ξ, as shown in Figure 15. Instead of using sparse regression, we use linear regression, setting $\kappa = 0$.

If the same perturbation were injected directly into the feature matrix, classical arguments predict an MSE_ξ would fall like n^{-1} when the number of data points n grows [29]. In our setting, however, the noise is introduced upstream and is subsequently amplified by division and multiplication, so a weaker variance-reduction rate is plausible. Surprisingly, an ordinary-least-squares fit of the empirical curve suggests an n^{-1} decay, implying that longer time series still provide a substantial buffer against measurement error, even when the noise is added at the earliest stage of the pipeline.

4.3 Ellipse Based Simulations

We now move on to a model that allows for network formation. We generate data using the governing equations as described in Section 2.2. First, we demonstrate the importance of including noise in the model. To this end, consider two models with $\alpha = 1$, $\lambda_r = 0.02$ and $\lambda_a = 0.0006$. For one simulation we set $N_v = 0$ and for the other $N_v = 0.4$. We begin with 1000 particles, identical initial configurations and set $\Delta t = 1$, while simulating from 0 through 20000. The results are shown in Figure 16. We see noise is indeed essential for networks to form.

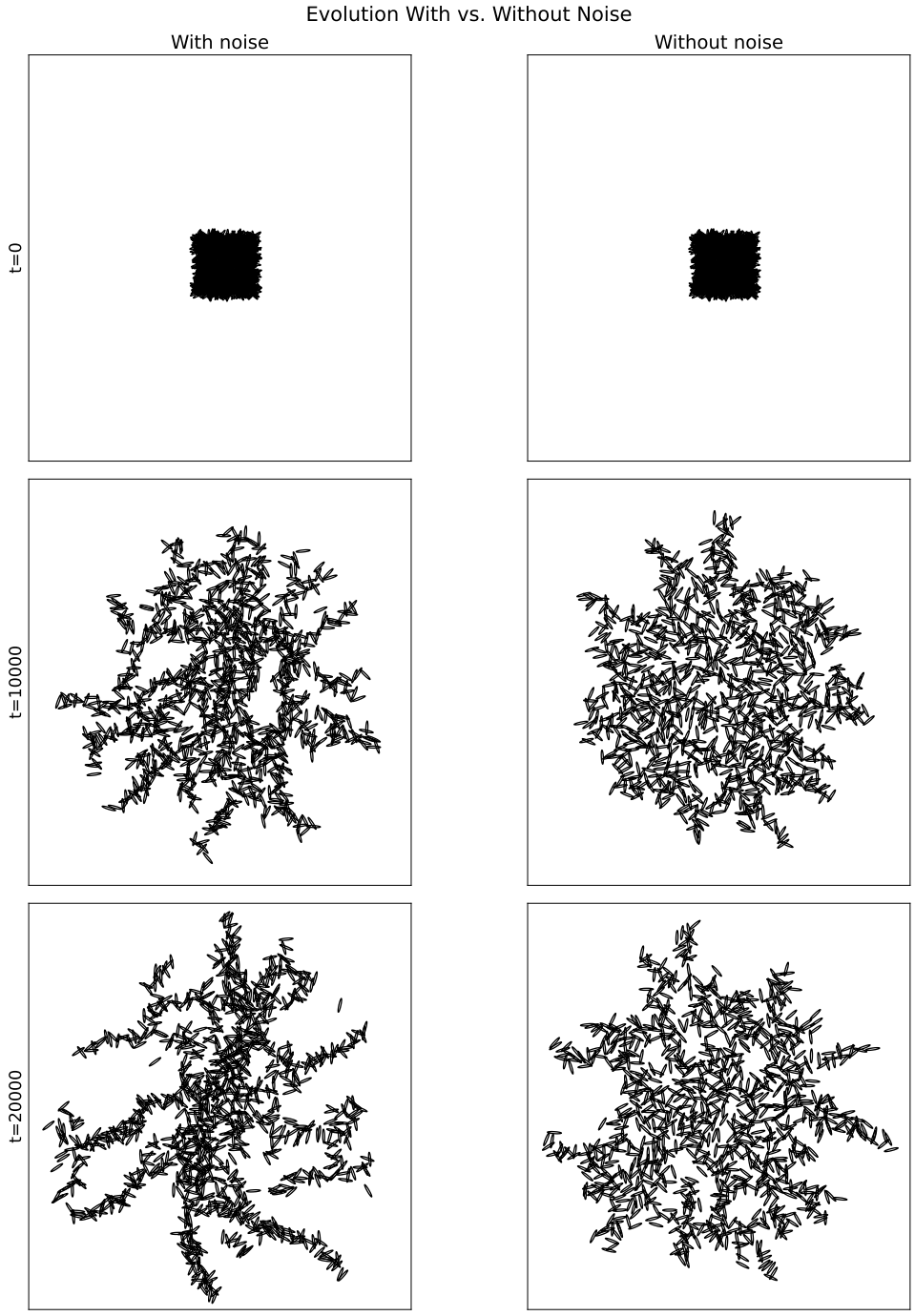


Figure 16: Trajectories of elliptical particles under the PBM, comparing a stochastic (left) and a deterministic (right) run. Both simulations use $\lambda_r = 0.02, \lambda_a = 0.0006, \alpha = 0.4, \Delta t = 1.0$, a major-minor axis ratio of 7, and an outer-inner ellipse ratio of 1.5. In the noisy case ($N_v = 0.4$, $N_a = 1$), particles undergo random reorientations and displacements. In the noise-free case ($N_v = 0$), orientations remain fixed.

Simplified model. We continue by learning a different simplified model from the overdamped regime. We remove noise and fix orientations for simplicity in all experiments, for an investigation in the quality of model retrieval including noise we refer to the thesis by Pham [26]. We can set $\alpha = \Delta t^{-1}$, so that we require one fewer parameter to learn. In that case, (7)-(9) collapse into

$$\frac{\vec{x}(t+\Delta t) - \vec{x}(t)}{\Delta t^2} = \lambda_r \left(\sum_{j \neq i} \frac{\vec{x_i}(t) - \vec{x_j}(t)}{r_{ij}(t)} A_r(t;\, i, j) \right) - \lambda_a \left(\sum_{j \neq i} \frac{\vec{x_i}(t) - \vec{x_j}(t)}{r_{ij}(t)} A_a(t;\, i, j) \right). \tag{17}$$

Note that coupling the physical parameter α to the numerical time step looses all physical meaning of this experiment. It is, however, a way to validate the methodology on a simpler example. Furthermore, the overdamped tests reported by Pham [26] effectively make this identification by taking $\Delta t = 1$ and $\alpha = 1$. We attribute their successful tests to this specific scaling. We find that for this α and any other choice of Δt, the overdamped approximation fails to reproduce the true dynamics. We also find model recovery for $\alpha = 1$ and $\Delta t = 1$ only succeeds if all time-derivatives are estimated with forward differences, not just at the initial step. Unfortunately, forward-difference approximations introduce a systematic bias and are less accurate than central differences.

We suspect this restriction stems from the underlying forward-Euler integration used in the simulator. Because the discrete trajectories themselves are generated with a first-order, one-sided scheme, any attempt to compute derivatives via a two-sided one introduces inconsistencies that degrade the regression. This discrepancy reflects limitations of the simulation's numerical accuracy rather than a deficiency in the system identification methodology. We therefore decrease the step size, and investigate whether these results hold.

We simulate the system using identical parameters, but with decreasing Δt and inversely proportionally increasing α, such that $\alpha = 1/\Delta t$ holds. This means our system follows (17). We start with a candidate library comprising only the correct terms, i.e.

$$F_{ij} = \lambda_r A_r(t;\, i, j) + \lambda_a A_a(t;\, i, j).$$

We then extend the library to include unnecessary terms, so that we assume the following form for the force term:

$$F_{ij} = \lambda_1 1 + \lambda_r A_r(t;\, i, j) + \lambda_a A_a(t;\, i, j) + \lambda_r' A_r^2(t;\ i, j) + \lambda_a' A_a^2(t;\ i, j).$$

We plot the R^2 and MSE_ξ for $\Delta t \in \{1, 1/10, 1/100\}$ for both these libraries. The results are shown in Figure 17. We see that the first reduction in time step drastically increases the quality of the retrieved system, while the second reduction only does so marginally. We also see the figures for the large and small candidate library are similar, indicating that increasing library size does not effect the quality of the inferred system.

Original model. Because the simplified update in (17) rigidly ties the damping constant to the time step by enforcing $\alpha = \Delta t^{-1}$, it cannot faithfully capture systems derived from noisy experimental trajectories. Accordingly, we revert to the full discretized dynamics of (7), in which the damping coefficient α remains to be identified. This more realistic approach, however, introduces two sources of additional complexity. First, estimating the acceleration demands a second numerical differentiation of the position data, which amplifies noise and magnifies any finite difference error. Second, α itself becomes an extra regression parameter, broadening the candidate space and raising the risk of underfitting.

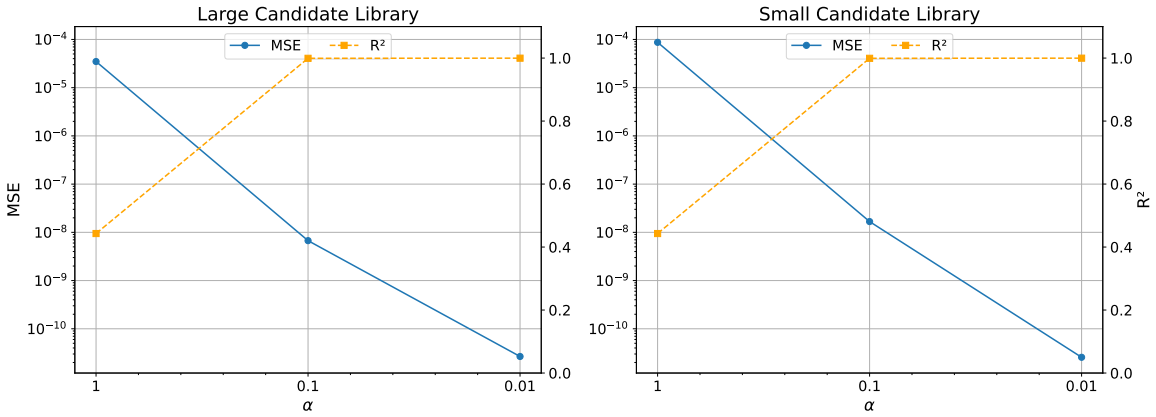


Figure 17: MSE_ξ and R^2 against different Δt for different size candidate library using $\lambda_r = 0.02$ and $\lambda_a = 0.0006$. Decreasing the time step from $\Delta t = 1$ to 0.1 yields a dramatic jump in R^2 and a steep drop in MSE_ξ, underscoring how critical moderately fine sampling is for accurate derivative estimation. By contrast, the further refinement to $\Delta t = 0.01$ produces only marginal improvements in R^2, indicating decreasing returns below a threshold of roughly $\Delta t = 0.1$.

We repeat the experiment, now trying to retrieve the parameter α, λ_a and λ_r in (7), as described by:

$$\vec{a}(t + \Delta t) = -\alpha \vec{v_i}(t) + \left(\sum_{j \neq i} \frac{\vec{x_i}(t) - \vec{x_j}(t)}{r_{ij}(t)} \left(\lambda_r A_r(t; i, j) - \lambda_a A_a(t; i, j) \right) \right). \tag{18}$$

We consider two time step sizes, $\Delta t = 0.1$ and $\Delta t = 0.01$ and drag coefficients $\alpha \in \{0.01, 0.1, 1, 10\}$. As shown in Figure 18, recovery quality is highest when α is small. Both the coefficient of determination R^2 and the parameter-error MSE_ξ indicate near-perfect fits for $\alpha = 0.01$ and $\alpha = 0.1$. As α increases to 1 and 10, performance degrades however. Stronger damping suppresses the acceleration term, making it harder to infer the underlying interaction forces from the trajectory data. It is important to note that we expect some increase is MSE_ξ as α increases, because the average size of the estimated coefficients increases.

Only considering the R^2 metric, we also observe that reducing Δt mitigates this effect. At $\Delta t = 0.01$, a drag coefficient of $\alpha = 10$ still yields a relatively high-quality reconstruction, whereas at $\Delta t = 0.1$ the same α leads to a poor fit. If we consider the MSE_ξ, however, we see that the identified system deviates more for $\Delta t = 0.01$. This indicates that finer sampling under strong damping can actually amplify biases in the recovered coefficients, despite the better trajectory reconstruction implied by a higher R^2. In other words, a small time step improves the shape of the trajectories but does not guarantee that the underlying interaction parameters are accurately estimated when α is large. In overdamped regimes with strong friction, sufficiently small Δt thus is a necessary, but not sufficient condition for reliable system identification.

Overdamped system. As a next experiment, we examine whether the system can be approximated by an overdamped regime. In this situation, we assume the acceleration to happen instantaneously, so that the equation is first order in time. In our case, the governing equation is then given by:

$$\vec{v}(t + \Delta t) = \sum_{j \neq i} \frac{\vec{x_i}(t) - \vec{x_j}(t)}{r_{ij}(t)} \left(\lambda_r A_r(t; i, j) - \lambda_a A_a(t; i, j) \right). \tag{19}$$

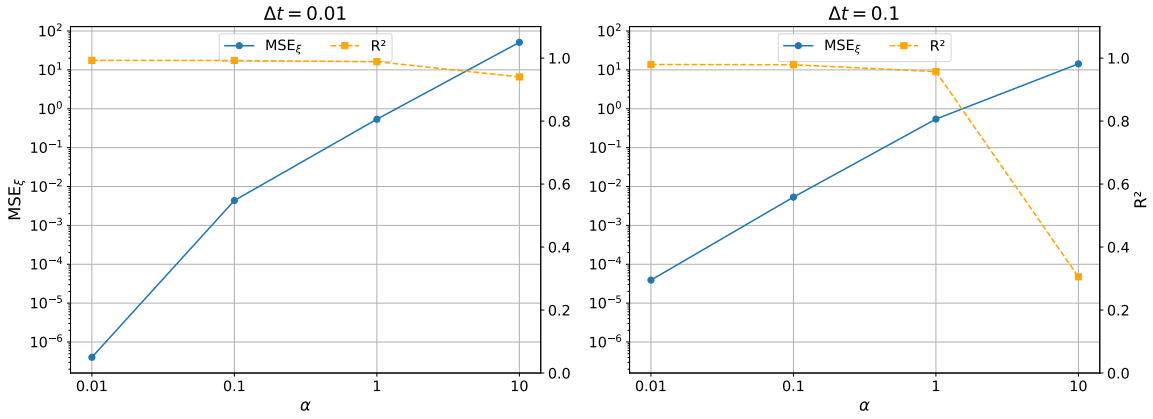


Figure 18: Reconstruction of performance versus the drag coefficient α for two temporal resolutions using $\lambda_r = 0.02$ and $\lambda_a = 0.0006$ using (18). In the left panel the R^2 metric remains near 1.0 even as α rises to 10, yet MSE_ξ increases sharply, revealing that although the inferred trajectories look excellent, the recovered coefficients deviate substantially under strong damping. In the right panel, both R^2 and MSE_ξ worsen once α exceeds 0.1, indicating that coarse sampling fails to capture the dynamics when friction is high.

We examine this system for different drag coefficients in Figure 19. When the coefficients get large, the system explodes due to the large $-\alpha$ term. We therefore use a small time step $\Delta t = 0.01$ to simulate the system. We expect the overdamped approximation to better capture the true system for larger damping coefficients, and see this is indeed the case for both metrics. We see an α of 100 is already sufficient to nearly perfectly infer the model.

4.4 Cellular Potts Simulations

We now apply the same methodology to the CPM data. We begin by simulating a CPM with 100 cells. The resulting configuration is shown in Figure 20. For every cell at each time point, we compute the best-fit ellipse as described in Section 2.1. The approximations are illustrated in Figure 21. Because the CPM output does not reveal a unique ratio between the repulsive inner ellipse and the attractive outer ellipse, we test several candidate ratios. We determine the pairwise overlap area for all cells at all time points and assemble the feature library using the governing law

$$\vec{a}(t+\Delta t) = -\alpha \vec{v}(t) + \left(\sum_{j \neq i} \frac{\vec{x_i}(t) - \vec{x_j}(t)}{r_{ij}(t)} \left(\lambda_r A_r(t;\, i,j) - \lambda_a A_a(t;\, i,j) \right) \right),$$

meaning we only seek the three terms included in the PBM. The cells are initialised as random blobs, requiring roughly 25 time steps to relax to an approximately elliptical shape; therefore, model inference is restricted to $t \geq 25$. The resulting coefficients of determination are

$$R^2 = 5.2 \times 10^{-4} \quad \text{(ratio 2)}, \qquad R^2 = 4.5 \times 10^{-4} \quad \text{(ratio 1.5)}, \qquad R^2 = 3.3 \times 10^{-4} \quad \text{(ratio 10)}.$$

Each value is statistically indistinguishable from predicting the median response, implying that the CPM dynamics are not well captured by the PBM framework, though several caveats remain, as described in Section 6.

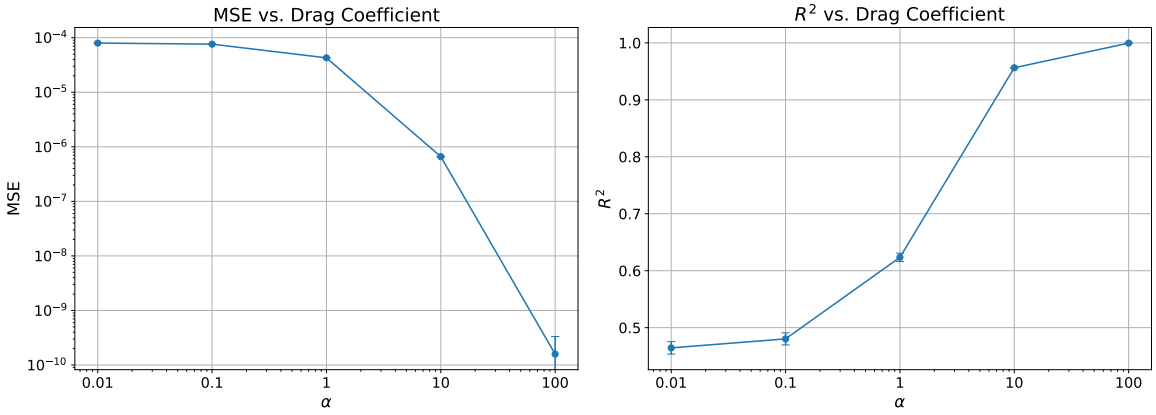
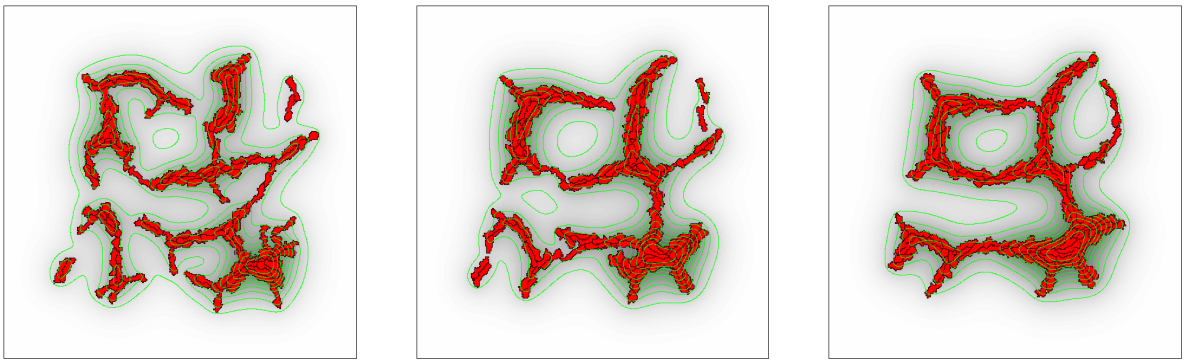


Figure 19: MSE_ξ (left) and R^2 (right) as functions of the drag coefficient α using (19). Markers give the across 3 seeds mean and vertical bars span one standard deviation. As α increases, the fit quality improves sharply. The coefficient of determination R^2 rises from below 0.5 to approximately 1, while MSE_ξ declines rapidly. The panels indicate an α of 100 is already sufficient to nearly perfectly infer the model.



(a) $t = 1000$ (b) $t = 2000$ (c) $t = 3000$

Figure 20: Illustration of CPM simulation with extended Hamiltonian, 100 cells and parameters: $T = 10^{-3}$, $A_{\text{target}} = 50$, $\nu_A = 20$, $J(\sigma_i, \sigma_j) = 40$ if lattice site i belongs to a cell and j belongs to the medium or vise versa, and $J(\sigma_i, \sigma_j) = 20$ if both i and j belong to a cell, $\nu_\ell = 60$ and $\chi = 10^{-3}$.

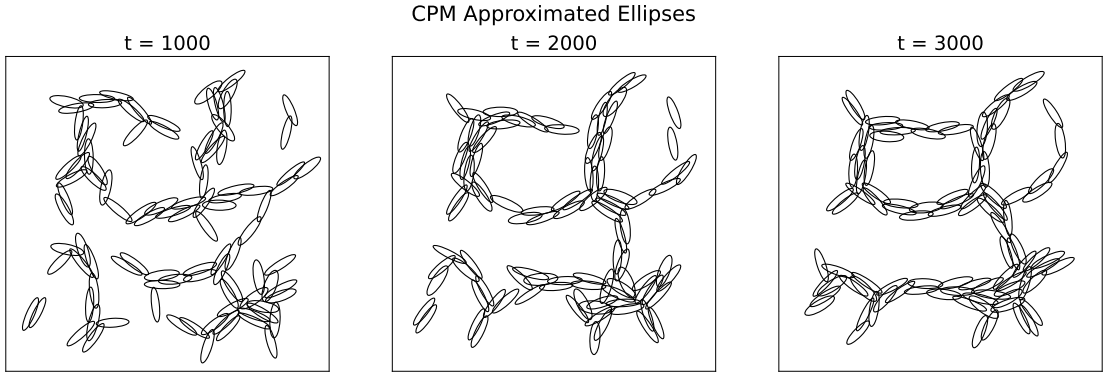


Figure 21: Illustration of the best fit elliptical approximation of the cells in Figure 20.

5 Predicting Movement

Owing to the limited success of SINDy on our data thus far, we pivot to a forecasting strategy grounded in deep learning. We employ SHallow REcurrent Decoder networks (SHRED), originally introduced by Williams et al. [34], which ingest the system state over a fixed time window and return a prediction for the next time step. This reframes our objective: instead of recovering explicit governing equations, we now focus on directly forecasting future cell trajectories from the available spatio-temporal measurements.

5.1 Particle Systems

We start by applying these networks to the simpler examples discussed in Section 4.1. The data at any time step t consists of the x-coordinates of all 10 particles, so the input vector has size 10. We create a network with 2 hidden and 2 decoder layers. The hidden layers of the encoder consist of 64 cells each, while the hidden layers of the decoder have 350 and 400 cells. The output of the last hidden layer of the decoder is then mapped onto an output vector of again size 10. This represents the coordinates of the particles at the next time step. We set the lag of the network to 10, meaning we use data from the previous 10 time steps to predict the next.

We simulate the 1 dimensional system for 1000 time steps in $[0, 2]$, with $k = 2$ and $\rho = 1$ for 100 different initial configurations. The particles are first placed uniformly at random in the interval $[-1.8, 1.8]$. Independent Gaussian noise $\mathcal{N}(0, 0.2)$ is then added to each position. We split the dataset into 80 train, 10 validation and 10 test instances. For the test instances we use the first ten measurements as starting point, after which we let the learned network successively predict the next position for all particles. The mean MDE across all datasets is plotted against the time in the left panel in Figure 22, along with one standard deviation bands. The right panel shows one specific example of ground truth trajectories against predicted trajectories. We see the model perform extremely well.

The procedure for generating initial positions systematically sorts the particles in ascending order of their coordinate values. The first element of the input vector is always approximately 1.8, the second approximately 1.4 , and so on. In more complex experimental settings, however, one cannot rely on such an ordering. To assess the impact of input-vector order on our results, we therefore repeat the experiment with the coordinates randomly permuted prior to analysis. Figure 23 shows the experiment. We see that the model fails to reproduce the initial dynamics. It does correctly indicate that the system attains an equilibrium, even though it does not accurately predict the equilibrium values.

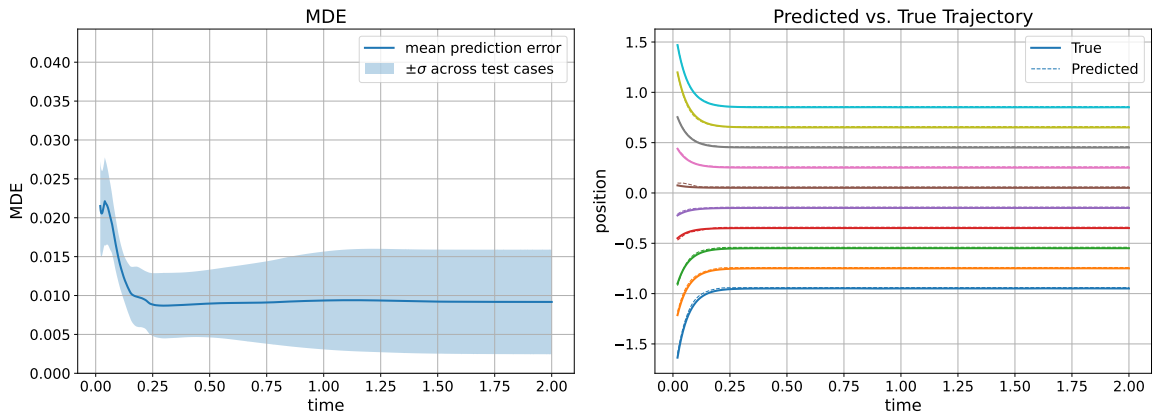


Figure 22: Time-resolved MDE for 10 unseen test trajectories (left) and representative ground-truth and SHRED predicted particle paths for one of those tests (right). The network was trained on 80 trajectories, each initialized at equidistant points spanning $[-1.8, 1.8]$, perturbed by Gaussian noise $\mathcal{N}(0, 0.2)$, with 10 additional trajectories reserved for validation. During testing the model autoregressively predicted each position from the preceding ten snapshots. The MDE rises slightly during the initial phase, then decays and plateaus near zero after $t \approx 1$, indicating that the network has learned the system has an equilibrium and no longer accumulates error once the system settles.

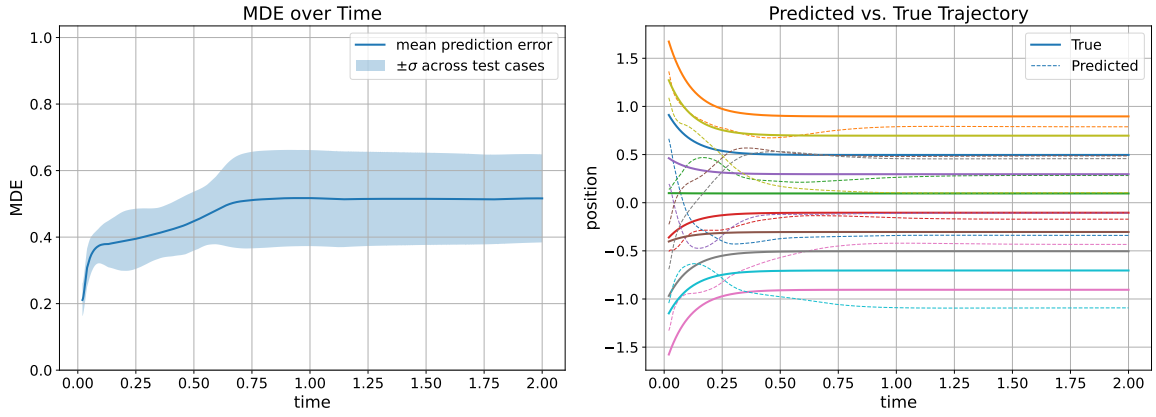


Figure 23: Time-resolved MDE for 10 unseen test trajectories (left) and representative ground-truth and SHRED predicted particle paths for one of those tests (right). The network was trained on 80 trajectories, each initialized at equidistant points spanning $[-1.8, 1.8]$, perturbed by Gaussian noise $\mathcal{N}(0, 0.2)$ and randomly shuffled, with 10 additional trajectories reserved for validation. During testing the model autoregressively predicted each position from the preceding ten snapshots. Although the model fails to reproduce the initial dynamics, it nonetheless correctly indicates that the system attains an equilibrium, even though it does not accurately predict the equilibrium values.

6 Discussion & Outlook

This thesis set out to learn, using modern data-driven system identification tools, the hidden cell-to-cell rules that drive de-novo vascular networks by confronting two very different simulation frameworks. One a stochastic, lattice-based Cellular Potts Model (CPM) and the other a deterministic, ellipse-based Particle Model (PBM).

On controlled 1D and 2D toy systems we confirmed that SINDy can, in principle, recover the exact interaction law, but only when three practical conditions are met. First, several complementary performance metrics must be monitored. Second, the temporal sampling must be fine enough to stabilise finite-difference velocity estimates. Third, eight to ten markedly different initial configurations are typically required to break parameter degeneracy. Violation of any of these conditions drives the regression toward spurious local minima, an observation that already hints at the challenges posed by richer biological data.

When the same pipeline was applied to ellipse-based PBM simulations, long-range attraction and elongation alone proved insufficient to trigger branching. Vascular-like patterns emerged only after injecting stochastic re-orientations and translations. In the overdamped limit, SINDy retrieved the remaining parameters progressively better for increasing damping coefficient. For the full second-order model, however, the drag coefficient itself becomes an extra unknown and coarse sampling again obscures the true forces.

Because explicit model discovery did not succeed for the CPM, we pivoted to SHRED. A remarkably small LSTM-based network, trained on only 80 trajectories, could forecast the 1D particle system all the way to equilibrium with negligible error. Yet when we removed ordering from the input vector, the very same architecture delivered insufficient accuracy and was unable to predict the equilibrium. That signals that more training data, a larger architecture, or both are needed for more complex forecasting tasks.

Future work should first broaden the data foundation on which the Particle-Based Model (PBM) is learned. Because the present study derives interaction forces from a single Cellular Potts Model (CPM) trajectory, the regressor is forced to explain one particular realisation of a highly stochastic process, risking over-fitting and parameter degeneracy. A logical next step is to generate an ensemble of CPM simulations that spans the experimentally relevant space of cell numbers, chemotactic cues and noise levels, and to revisit system identification in a hierarchical or Bayesian framework that pools information across all trajectories. Such an approach would expose the regression to diverse spatial configurations, and thereby sharpen parameter estimates.

A second priority is to replace two ad-hoc choices with data-driven calibration. The inner-to-outer ellipse ratio used to approximate CPM cell shapes was set heuristically, yet this geometric parameter controls the balance between short-range steric repulsion and long-range alignment in the PBM. Systematically extracting ellipse fits from many CPM snapshots, and optimizing the ratio would potentially drastically improve the learned model.

Finally, the forecasting component of the thesis currently relies on a minimal two-layer LSTM. Exploring deeper recurrent networks, coupled with more extensive training data, could improve long-horizon accuracy and open the door to hybrid architectures. In this context it is interesting to investigate how SINDy-SHRED could be of use in forecasting cell movement. Together, a larger CPM training ensemble, rigorously calibrated cell geometry and more expressive neural forecasters promise a more robust, interpretable and biologically faithful framework for simulating vascular network formation.

References

[1] Takayuki Asahara et al. "Bone marrow origin of endothelial progenitor cells responsible for postnatal vasculogenesis in physiological and pathological neovascularization". In: *Circulation research* 85.3 (1999), pp. 221–228.

[2] Sonja E.M. Boas et al. "Cellular potts model: applications to vasculogenesis and angiogenesis". In: *Probabilistic Cellular Automata: Theory, Applications and Future Perspectives* (2018), pp. 279–310.

[3] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113.15 (2016), pp. 3932–3937.

[4] Hyun-Myung Chun, Xavier Durang, and Jae Dong Noh. "Emergence of nonwhite noise in Langevin dynamics with magnetic Lorentz force". In: *Physical Review E* 97.3 (2018), p. 032117.

[5] Josephine T. Daub and Roeland M.H. Merks. "Cell-based computational modeling of vascular morphogenesis using Tissue Simulation Toolkit". In: *Vascular morphogenesis: methods and protocols*. Springer, 2014, pp. 67–127.

[6] Silvia Daun et al. "Equation-based models of dynamic biological systems". In: *Journal of critical care* 23.4 (2008), pp. 585–594.

[7] Delft University of Technology. *Singular Value Decomposition (SVD)*. `https://interactivetextbooks.tudelft.nl/linear-algebra/Chapter8/SingularValueDecomp.html`. Accessed: 2025-06-06.

[8] Carl Eckart and Gale Young. "The Approximation of One Matrix by Another of Lower Rank". In: *Psychometrika* 1.3 (1936), pp. 211–218. DOI: 10.1007/BF02288967.

[9] Jerome H. Friedman, Trevor Hastie, and Rob Tibshirani. "Regularization paths for generalized linear models via coordinate descent". In: *Journal of statistical software* 33 (2010), pp. 1–22.

[10] Kai Fukami et al. "Sparse identification of nonlinear dynamics with low-dimensionalized flow representations". In: *Journal of Fluid Mechanics* 926 (2021), A10.

[11] Mars L. Gao, Jan P. Williams, and J Nathan Kutz. "Sparse identification of nonlinear dynamics and Koopman operators with Shallow Recurrent Decoder Networks". In: *arXiv preprint arXiv:2501.13329* (2025).

[12] François Graner and James A Glazier. "Simulation of biological cell sorting using a two-dimensional extended Potts model". In: *Physical review letters* 69.13 (1992), p. 2013.

[13] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[14] Suk-Won Jin and Cam Patterson. "The opening act: vasculogenesis and the origins of circulation". In: *Arteriosclerosis, thrombosis, and vascular biology* 29.5 (2009), pp. 623–629.

[15] Keith J Laidler. "The development of the Arrhenius equation". In: *Journal of chemical Education* 61.6 (1984), p. 494.

[16] John E. Leffler. "The interpretation of enthalpy and entropy data". In: *The Journal of Organic Chemistry* 31.2 (1966), pp. 533–537.

[17] Brendan Lenfesty, Saugat Bhattacharyya, and KongFatt Wong-Lin. "Uncovering dynamical equations of stochastic decision models using data-driven SINDy algorithm". In: *Neural Computation* 37.3 (2025), pp. 569–587.

[18] Jinwen Liang et al. "Discovering dynamic models of COVID-19 transmission". In: *Transboundary and Emerging Diseases* 69.4 (2022), e64–e70.

[19] Nikos V. Mantzaris, Steve Webb, and Hans G. Othmer. "Mathematical modeling of tumor-induced angiogenesis". In: *Journal of mathematical biology* 49.2 (2004), pp. 111–187.

[20] Sean A.C. McDowell. "A simple derivation of the Boltzmann distribution". In: *Journal of chemical education* 76.10 (1999), p. 1393.

[21] Kieran F. Mulchrone and Kingshuk Roy Choudhury. "Fitting an ellipse to an arbitrary shape: implications for strain analysis". In: *Journal of structural Geology* 26.1 (2004), pp. 143–153.

[22] James D. Murray. *Mathematical biology: I. An introduction*. Vol. 17. Springer Science & Business Media, 2007.

[23] Tadashi Nakano and Tatsuya Suda. "Modeling and simulations of bio-nanomachines for spatiotemporal pattern formation". In: *Proceedings of the 5th ACM International Conference on Nanoscale Computing and Communication*. NANOCOM '18. Reykjavik, Iceland: Association for Computing Machinery, 2018. ISBN: 9781450357111. DOI: 10.1145/3233188.3233225.

[24] Dimitrios Palachanis, András Szabó, and Roeland M.H. Merks. "Particle-based simulation of ellipse-shaped particle aggregation as a model for vascular network formation". In: *Computational Particle Mechanics* 2.4 (2015), pp. 401–414. DOI: 10.1007/s40571-015-0064-5.

[25] Sybill Patan. "Vasculogenesis and angiogenesis as mechanisms of vascular network formation, growth and remodeling". In: *Journal of neuro-oncology* 50 (2000), pp. 1–15.

[26] Tuan D. Pham. "Time–frequency time–space LSTM for robust classification of physiological signals". In: *Scientific reports* 11.1 (2021), p. 6936.

[27] Werner Risau. "Mechanisms of angiogenesis". In: *Nature* 386.6626 (1997), pp. 671–674.

[28] Werner Risau and Ingo Flamme. "Vasculogenesis". In: *Annual review of cell and developmental biology* 11.1 (1995), pp. 73–91.

[29] George A.F. Seber and Alan J. Lee. *Linear regression analysis*. John Wiley & Sons, 2003.

[30] Fouad Shalaby et al. "Failure of blood-island formation and vasculogenesis in Flk-1-deficient mice". In: *Nature* 376.6535 (1995), pp. 62–66.

[31] Dean G. Tang and Claudio J. Conti. "Endothelial cell development, vasculogenesis, angiogenesis, and tumor neovascularization: an update". In: *Seminars in thrombosis and hemostasis*. Vol. 30. 01. Thieme Medical Publishers. 2004, pp. 109–117.

[32] Matteo Tomasetto et al. "Reduced order modeling with shallow recurrent decoder networks". In: *arXiv preprint arXiv:2502.10930* (2025).

[33] Kees Vuik et al. *Numerical methods for ordinary differential equations*. TU Delft Open Publishing, 2023.

[34] Jan P. Williams, Olivia Zahn, and J Nathan Kutz. "Sensing with shallow recurrent decoder networks". In: *arXiv preprint arXiv:2301.12011* (2023).

[35] Mervin C. Yoder. "Defining human endothelial progenitor cells". In: *Journal of Thrombosis and Haemostasis* 7 (2009), pp. 49–52.

A Code Adaptations

All source code developed or modified for this work that is not licensed is available at `https://github.com/WillemScholtenLeiden/Cell-to-Cell-Interactions`. The changes made to licensed code are listed below.

A.1 CPM Adaptations

All CPM simulations were performed with the *Tissue Simulation Toolkit* of Daub et al. [5]. To extract best-fit ellipses for every cell at each snapshot we inserted the C++ fragment in Listing 1 inside the `try` block of the executable's main loop, immediately after the first `if` statement that checks whether a snapshot should be written. The routine gathers the lattice coordinates of every cell, computes the centroid and second-order central moments, diagonalises the covariance matrix to obtain the semi-axes a, b and orientation ϕ, and writes (c_x, c_y, a, b, ϕ) to `output/ellipse_###.dat`.

Listing 1: Ellipse-fit export routine added to the CPM executable

```
std::map<int,std::vector<std::pair<int,int>>> cellCoords =
        dish->CPM->GetCellCoordinates();
if (cellCoords.empty()) return;

std::ostringstream name;
name << "output/ellipse_" << std::setw(5) << std::setfill('0') << i << ".dat";
std::ofstream out(name.str());
out << "# cx cy a b phi\n";

for (const auto& [id,pix] : cellCoords) {
    if (pix.size() < 5) continue;   // too few points for a robust ellipse

    /* --- centroid --- */
    double sx=0.0, sy=0.0;
    for (auto [x,y] : pix) { sx += x; sy += y; }
    const double cx = sx / pix.size();
    const double cy = sy / pix.size();

    /* --- covariance matrix --- */
    double mu20=0.0, mu02=0.0, mu11=0.0;
    for (auto [x,y] : pix) {
        const double dx = x - cx, dy = y - cy;
        mu20 += dx*dx;  mu02 += dy*dy;  mu11 += dx*dy;
    }
    const double cxx = mu20 / pix.size();
    const double cyy = mu02 / pix.size();
    const double cxy = mu11 / pix.size();

    /* --- eigen analysis --- */
    const double tr  = cxx + cyy;
    const double diff = cxx - cyy;
    const double sq  = std::sqrt(diff*diff + 4.0*cxy*cxy);
    const double lmax = 0.5*(tr + sq);
    const double lmin = 0.5*(tr - sq);

    const double a = 2.0*std::sqrt(lmax);
    const double b = 2.0*std::sqrt(std::max(0.0,lmin));
    const double phi = 0.5*std::atan2(2.0*cxy, diff) * 180.0/M_PI;

```

```
    out << cx << ' ' << cy << ' ' << a << ' ' << b << ' ' << phi << '\n';
}
```

A.2 PBM Adaptations

The C++ PBM simulator of Palachanis et al. [24] was modified to output, at every snapshot, the pair-wise overlap areas required for force identification. Two symmetric matrices are written:

- `areaRep` – inner-ellipse overlaps (repulsion),
- `areaAtt` – outer-ellipse overlaps (attraction).

Overlap calculation. Listing 2 shows the helper function `computeAreaMatrices` that computes both matrices in one pass. It is placed directly above `move_cells()` in `simulation.cpp`.

Listing 2: Computation of repulsive and attractive overlap areas

```
void computeAreaMatrices(const vector<Cell*>& cells,
                         double attRange,
                         vector<vector<double>>& areaRep,
                         vector<vector<double>>& areaAtt)
{
    const int N = static_cast<int>(cells.size());
    areaRep.assign(N, vector<double>(N, 0.0));
    areaAtt.assign(N, vector<double>(N, 0.0));

    for (int i = 0; i < N; ++i) {
        for (int j = i+1; j < N; ++j) {

            if (Distance(cells[i],cells[j]) >
                2.0*attRange*cells[i]->getLength()) continue;

            const double rep = ellipse_ellipse_overlap(
                /* inner ellipses */);
            const double att = ellipse_ellipse_overlap(
                /* outer ellipses */);

            areaRep[i][j] = areaRep[j][i] = rep;
            areaAtt[i][j] = areaAtt[j][i] = att;
        }
    }
}
```

Snapshot export. The call site inside `move_cells()` (Listing 3) executes every t_{snapshot} steps and writes the matrices to `output_###_areaRep.dat` and `output_###_areaAtt.dat` alongside the standard position/angle dump.

Listing 3: Writing overlap matrices inside `move_cells()`

```
if (LONGSIM && t % t_snapshot == 0) {

    vector<vector<double>> areaRep, areaAtt;
    computeAreaMatrices(particles, attRange, areaRep, areaAtt);

    ostringstream repName, attName;
    repName << "output_" << ctr2 << "_areaRep.dat";
```

```
    attName << "output_" << ctr2 << "_areaAtt.dat";

    FILE* fRep = fopen(repName.str().c_str(),"w");
    FILE* fAtt = fopen(attName.str().c_str(),"w");

    for (int r = 0; r < maxSize; ++r) {
        for (int c = 0; c < maxSize; ++c) {
            fprintf(fRep,"%f ",areaRep[r][c]);
            fprintf(fAtt,"%f ",areaAtt[r][c]);
        }
        fprintf(fRep,"\n");
        fprintf(fAtt,"\n");
    }
    fclose(fRep); fclose(fAtt);
}
```

Initial snapshot. Finally, before the main time loop we record the initial configuration (`output_data_0.dat`) so that every run has a time-zero reference; see Listing 4. Coordinates, semi-axes, and orientations are written in the same order used by subsequent snapshots.

Listing 4: Initial state dump before time integration

```
FILE* out0 = fopen(make_output_filename(ctr2).c_str(),"w");
for (int i = 0; i < maxSize; ++i) {
    fprintf(out0,"%f %f %f %f %f\n",
            particles[i]->getCurrX(),
            particles[i]->getCurrY(),
            A1, B1,
            particles[i]->getTheta()*180.0/M_PI);
}
fclose(out0);
```