Waddle – Waddington Epigenetic Landscape

Project 1 Computing – Group Report

Felicia Burtscher, Lucas Ducrot, Madeleine Hall, Luis Torada

March 2018

MSc in Bioinformatics and Theoretical Systems Biology

Imperial College London

Abstract

The *Waddington* or *epigenetic landscape* concept has become an important framework to reason about developmental processes. This project develops a set of Julia tools to determine the structure of such landscapes for a given dynamical system, as well as single-cell data. For this purpose, we implement the probability flux method, additionally applying kernel density estimation. We also implement the action-based method on dynamical systems with the aid of stability analysis. For high dimensional systems, the topic of dimensionality reduction is addressed, with corresponding tools developed to aid visualisation. The code can be found on:
`https://github.com/burfel/waddington-project`.

Acknowledgements

This project was conducted under the supervision of Prof Michael PH Stumpf, whom we thank for his advice and guidance. Thanks also to other members of the Theoretical Systems Biology Group at Imperial College London, Rowan Brackston and Ivan Croydon Veleslavov, for their help and expertise in both the topic and Julia as well as Suhail A Islam for fixing technical issues and his incredible patience. Finally, thanks to Prof Michael Sternberg and others involved in conducting and overseeing the MSc in Bioinformatics and Theoretical Systems Biology at Imperial College London.

Abbreviations

ABM: action-based method

CPU: central processing unit

DR: dimensionality reduction

GPU: graphics processing unit

KDE: kernel density estimation

ODE: ordinary differential equation

PCA: principal component analysis

PFM: probability-flux method

SBML: Systems Biology Markup Language

SDE: stochastic differential equation

SVD: singular value decomposition

UML: General Unified Modeling Language

XML: Extensible Markup Language

Contents

1 Introduction and Context

1.1 Historical Background [MH]

First proposed in 1940 by Conrad H Waddington, the *epigenetic landscape* was primarily suggested as the diagrammatic representation of a developing system [23]. In the original publication, Waddington states "the path followed by the ball corresponds to the developmental history of a particular part of the egg". The ball can be thought of as a cell, tracing a path corresponding to the expression of genes as it progresses through development. In Figure 1, there are four different destinations the ball could find itself in, corresponding to four different cell types a developing cell could end up as. Of course, in a given organism, the number of types of distinguished cell an undifferentiated cell could specialise to could be immense.

Waddington's landscape was widely considered to be nothing more than a metaphor to aid understanding of developing systems. However, due to recent developments in sophisticated methods of modelling such systems, in addition to the availability of new technologies to obtain novel data, it is now possible to mathematically and computationally construct a landscape for a developmental process.

An epigenetic landscape can be visualised as a surface on which a cell may sit. Throughout development, the cell will move over the landscape as it's gene expression levels change. As with any landscape, it is easier to navigate by following

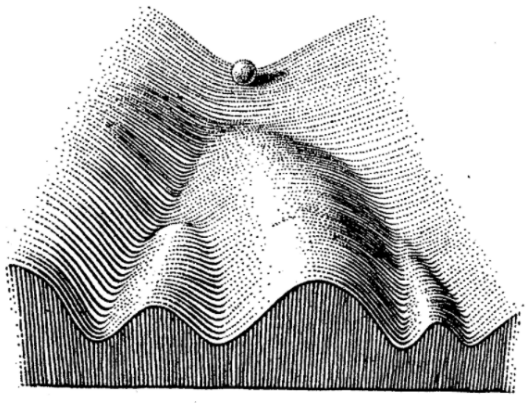


Figure 1: The original Figure from Waddington, *Part of an Epigenetic Landscape* [23].

the slopes and valleys, as opposed to traversing steep and unstable ridges. However, an element of stochasticity is displayed in development, and the path that is traced by a cell across a landscape is rarely smooth and undeviating before it reaches its destination [17].

The capacity to visualise an epigenetic landscape not only aids understanding of interactions in a gene regulatory network (GRN), but could also allow for predictions of cell fates given a starting position of current gene expression level.

1.2 Author Contributions

Authorship in this report is indicated in section headings by inclusion of author initials.

- Felicia Burtscher - [FB]
- Lucas Ducrot - [LD]
- Madeleine Hall - [MH]
- Luis Torada - [LT]

Headings without such notation should be assumed to have been developed and written in equal collaboration between all members.

2 Literature Review [FB]

At the start of our project was an extensive literature research on the work that has already been done regarding Waddington's landscape.

One of the first tools we came across was *NetLand*, a software to model and visualise Waddington's epigenetic landscape [8]. It was designed to quantitatively model, simulate and visualise gene regulatory networks (GRNs) and their corresponding quasi-potential landscapes [8]; one can import models of GRN files including SBML format and manually adjust the networks structure. For the visualisation it allows to manually select genes or apply dimensionality reduction. It runs on a Java virtual machine and comes with a nice user interface.

In order to familiarise with the research in this field and to later situate our work within the wider literature, we will highlight major state-of-the-art tools that have been developed by other groups.

A lot of these models use pseudotime estimation to help understand the transition of gene expression profiles. *Pseudotime* is used to measure the progress through a biological process along which cells are arranged based on their expression profiles [9]. By using estimated pseudotime of single-cell data, we can identify important regulators by comparing the expression profiles around the branching time points [9]. Due to high variabiliy in the gene expression between cells, recovery of pseudotime is challenging.

One interesting model that tries to do that and achieves currently the most accurate pseudotime prediction compared to other methods, is *HopLand* which we studied in more detail (see [MH]'s report). It uses a continuous Hopfield network, a type of recurrent neural network, to map cells to a Waddington's epigenetic landscape. From the single-cell data, it tries to reveal the regulatory interaction between genes that control the progression through successive cell states [9]. It performs well on identifying key genes and regulatory interactions driving the transition of cell states. The pseudotime is estimated by computing the geodesic distance between every two cells in the landscape [9]. A major advantage to this method is also, that

we do not need temporal information or prior knowledge of marker genes [9] and we can simulate differentiation processes with multiple lineages.

Several other models came before HopLand. One of them is *Wanderlust* [1] which uses a graph-based trajectory detection algorithm to align cells on a 1D trajectory on ther developmental path. On the one hand, this approach had a couple of constraints including that cells have to be representative of the whole developmental process and that trajectories are non-branching. On the other hand, graph-based representation of the data can handle challenges like constructing an accurate trajectory when the relationships between markers cannot be assumed to be linear and a standard metric would result in poor measurement of the actual distance in development [1].
Another interesting model we discovered was *Topslam* [28]. It estimates the pseudotime by mapping the individual cells to the surface of the landscape. Similar to HopLand it also uses a probabilistic dimensionality reduction technique. The locations of the individual cells reflect their degree of maturity during differentiation as the as they move along the topography of the probabilistic landscape [9]. To redefine distance, Topslam uses the topography of the landscape which, however, lacks biological meaning.
Another model called *diffusion map* [10] uses diffusion distances to simulate cell differentiation. This way, it can order cells along the differentiation path without losing the non-linear structure of the data.

Other models include *SCUBA* [18] which uses temporal information to perform bifurcation analysis of single-cell data to recover the cell lineages [18]. Also *Wishbone* [19] which is based upon Wanderlust aligns single cells into bifurcation branches, it identifies the bifurcation points and recovers the pseudotemporal ordering of cells. *Monocle* [22] which builds a minimum spanning tree connecting cells to estimate the pseudotime. We analysed these models, with a special interest in how they cope

with dimensionality reduction (DR) (see [FB]'s report).

As much on the visualisation side of Waddington's landscape has been achieved already (see [8] and last year's group), we wanted to more focus on the methodology side of Waddington's landscape. Having said that, we wanted to provide our user different tools to simulate and analyse the landscape, or based on single-cell data visualise certain genes or gene combinations after applying dimensionality reduction. We believe this to be a good compromise between more elaborated models that already exist as outlined in section 2 and the visualisation amongst NetLand and the result from last year's group. In the end, our package is written in Julia and does not require a Java virtual machine as NetLand does.

3 Mathematical Framework

3.1 Potential Landscapes [MH, FB]

Epigenetic landscapes are similar to *potential functions* (or harmonic functions), which are real functions $u(x, y)$ with continuous second partial derivatives and satisfying Laplace's equation

$$\Delta u(x, y) = \nabla^2 u(x, y) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0.$$

Models for stem cell development describe systems of chemical reactions. The time evolution of these systems can be approximated by the Chemical Langevin Equation [7]. Given S the stoichiometry matrix associated with the system, and a the vector of propensities (rates) for each of the reactions, we have

$$dX = Sa(X)dt + S\text{diag}(\sqrt{a(X)})dW_t.$$

Therefore, given a model (of Chemical Langevin Equation type as above), hypothetically one can generate the corresponding epigenetic landscape.

In general, a stochastic model takes the form

$$dX = f(X)dt + g(X)dW_t. \tag{1}$$

Low noise (or equivalently, high copy numbers) allows approximation of the Stochastic Differential Equation (SDE) into an Ordinary Differential Equation (ODE) in the limit. The minimum action path (MAP) is the most probable path between two points, and provides the relative probabilities of different trajectories [6]. The action is zero for deterministic trajectories, which follow the 'slope' of the landscape. Action is regulated by noise intensity.

For $dX = f(X)dt + g(X)dW_t$, which may be parameterised by θ, the deterministic forcing vector $f(X;\theta)$ can be decomposed into the gradient of a potential, ∇U, and a remaining component, $f_U(X;\theta)$, often referred to as the *curl*:

$$f(X;\theta) = -\nabla U(X;\theta) + f_U(X;\theta). \tag{2}$$

U is the *quasi-potential* and is analogous to the epigenetic landscape. f_U is the remainder term, and fills out remaining dynamics (in some occasions, the curl is parallel to the contours). Key requirements for a potential landscape U are [27]:

1. U is a Lyapunov function for the deterministic system.

2. U provides the action of the MAP between points.

A Lyapunov function is continuous, positive definite ($U(x,y) > 0 \, \forall x, y \neq 0$) and has continuous first order partial derivatives.

From a dynamical systems perspective, the potential represents the relative amount of dynamics happening in a certain state. Intuitively, as the system thermodynamically thrives towards an energetic minimum, it will move away from regions with higher potential and stay in regions with least dynamic potential, so called *basins of attraction*. From that viewpoint it is intuitive to think of the potential as an inversely proportional to the probability of a particular state, i.e. a cell ending

up in this state, which can be expressed as the negative logarithm of the probability distribution

$$U \propto -\ln(P_s(X)), \tag{3}$$

where P_s denotes the steady state probability distribution over the state space.

For an implementation of the *probability flux method* (PFM) we refer to chapter 3.1.2. Other methods of constructing the epigenetic landscape include the *action based method* (see chapter 3.1.1), and *orthogonal vector field decomposition* [27]. Implementations of artificial neural networks have also been used as an alternative approach to tackling the task [9].

3.1.1 Large Deviations Theory: Action Based Method [LT]

Large Deviation Theory describes the behavior of a dynamical system under stochastic perturbations, where perturbations are small and arise from long-tails distributions.

One key object from this theory is the action functional, given by

$$S(\varphi, T) = \int_0^T \sum_i (\dot{x}_i - f_i(x)))^2 / g_i^2(x) dt \tag{4}$$

It is a function of a trajectory between two points, and corresponds to the line integral of the difference between the deterministic and stochastic parts, normalized by the noise intensity, along that path.

The trajectory that is mapped to the minimum action functional is called the Minimum-Action Path (MAP), $V_{x_1 \to x_2}$

$$V(x_1, x_2) = \inf_{T>0} \inf_{\varphi \in \bar{C}_{x_1}^{x_2}(0,T)} S_T(\varphi) \tag{5}$$

Furthermore, it can be proved that $V_{x_1 \to x_2}$ is unique [27].

Intuitively, $V_{x_1 \to x_2}$ is the difficulty with which a stochastic system can move from the initial to the final point of the path (equivalently, it can be understood as the force required to do so). In the context of Lyapunov functions, $V_{x_1 \to x_2}$ is the height

of the "energy barrier" between both points.

If the noise intensity g is small, deviations from the deterministic trajectories are very infrequent. If a deviation occurs, it is likely to be one with the highest probability [11].

Heymann *et al.* simplified the analytical form of the action functional so that it involves only one minimization step. They also show this analytical result for the specific case of stochastic differential equations (SDEs) [11]:

$$V(x_1, x_2) = \inf_{T>0} \inf_{\varphi \in \bar{C}^{x_2}_{x_1}(0,T)} \int_0^T |\dot{\varphi}(t) - b(\varphi(t))|^2_{a(\varphi)} dt \tag{6}$$

The action based method forms a landscape based on the MAP between points; of special interest are the stable fixed points identified by stability analysis (see section 3.2).

3.1.2 Fokker-Planck Equation: Probability Flux Method and Kernel Density Estimation [LT, LD]

Given a dynamical system under stochastic perturbations, trajectories can be simulated using SDEs and the probability distribution of its state-space can be quantified. When the state-space is continuous, the time-evolution of its probability density function (PDF) can be described with the Fokker-Planck equation, which can be derived from the corresponding SDE:

$$\frac{\partial P(x,t)}{\partial t} = -\nabla \cdot (fP) + D \cdot \nabla^2 P \tag{7}$$

If the system has reached its equilibrium state, so that the states PDF has converged and does not change, we can set $\frac{dP}{dt} = 0$, obtaining

$$f = -D \cdot \nabla(-\ln P_S) + f_c \tag{8}$$

where D is the diffusion coefficient, and is often dependent on concentration. This resulting expression contains implicitly a decomposition of the deterministic non-

gradient vector field driving the rate of change of the system into a gradient field ($\nabla(-\ln P_s)$) and a remaining term [27]. We will refer to the integral of the gradient field as the 'quasi-potential', U, and to the remaining term as the 'curl force', f_c, thus obtain the equivalent expression:

$$f = -\nabla U + f_c \tag{9}$$

such that $U \propto -\ln P_s$.

In summary, the gradient field obtained from the deterministic part of the model, $f(X)$, is proportional to the PDF of the equilibrium state of the system, and it can be intuitively interpreted as its 'energy' function (that we call quasi-potential because it accounts only for a part of the system dynamics).

The Kernel Density Estimation (KDE) method follows the same idea but differs on the computation of P_S. In the primary PFM, the density distribution is calculated by counting the number of points falling in a certain area of the space. KDE computes an approximate density function $\hat{f}_h$ with data (x_i, N points) and a bandwidth (h) as parameters and position in space (x) as input.

$$\hat{f}_h(x) = \frac{1}{Nh} \sum_{i=1}^{N} K(\frac{x - x_i}{h}) \tag{10}$$

Here K is a kernel function. There are many options of kernels to use. The Gaussian Kernel (also known as the squared exponential one) is the most widely used:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

From this, the potential U can still be computed with (3). Other kernel functions including the exponential kernel, the linear kernel, the cosine kernel, the tophat kernel and the epanechikov kernel can be used; however, they will not be discussed any further here.

3.1.3 Comparison of the Action-Based Method and the Probability-Flux Method [LT]

An equivalent formulation of conditions that we require for an informative Waddington landscape are that the landscape function 1) is actually a potential function (and therefore a Lyapunov function) and, if possible, 2) that it exclusively accounts for the global metastability of the system (that is, that it is orthogonal to the remaining 'curl force'). As a consequence of the later, spontaneity of a path is equivalent to the work of the quasi-potential gradient along it.

Zhou *et al.* provide a mathematical comparison between the probability-flux method and the action-based method (explained in the next section) by whether they meet or not these two desired conditions [27]. In summary,

1. The PFM does not guarantee, theoretically, that the result is a Lyapunov function. As explained in the mathematical background section, the PFM vector field decomposition was performed by introducing the steady-state equilibrium constraint into the Fokker-Planck equation setting $\frac{dP}{dt} = 0$, thus obtaining equation (9). If we then introduce the constraint, in an equivalent form, by setting $f = -\nabla U + f_c$ and $P = \exp(-U/D)$ (Arrhenius equation for the probability of transitions between two points in equilibrium), we obtain:

$$-\nabla U \cdot f_c = D\nabla \cdot f_c \tag{11}$$

and

$$dU/dt = -(\partial U/\partial x) - (\partial U/\partial y) + (D\nabla \cdot f_c) \tag{12}$$

But $\nabla \cdot f_c$ is not necessarily equal to 0 and then ∇U and f_c are not necessarily orthogonal. Intuitively, the consequence is that f_c can contribute to the ease of transitions between attractors, and therefore the height of the landscape loses information about the relative metastability of the such attractors. Also,

equation 12 shows that it is not even guaranteed that the stochastic dynamical system always evolves in time by decreasing the quasi-potential, so that $\frac{dU}{dt}$¡0. This would only be the case as noise intensity (represented by D) tends to zero.

2. The action-based method does guarantee that the result is a Lyapunov function, but only approximately in practice.

   The action-based method does guarantee, theoretically, that the obtained quasi-potential is orthogonal to the curl component, therefore satisfying the two listed conditions.

Although the action-based method is superior in theory to the PFM, its computation gives rise to practical issues that make their comparison less trivial. These issues, mainly related to the noise intensity, are discussed in the 'challenges and approaches' section. In practice, the noise intensity of the dynamical system could be a good decision criteria. This is because a small noise intensity would make the deviation theory approach consistent, while preventing the convergence to the stationary distributions during the simulations for the PFM. In the opposite situation, the PFM would be faster, and the action-based method inappropriate.

3.2 Dynamical Systems: Stability Analysis [LT]

Given a deterministic dynamical system, its equilibrium points can be found and characterized, equally the qualitative behavior of the non-equilibrium system.

In the context of Dynamical Systems Theory, equilibrium solutions of the system are called *fixed points*, and they can be classified as stable (*attractors*), unstable (*repellers* or saddle points). This classification is based on the property whether the system in an equilibrium state returns to it or moves away from it after an arbitrary small perturbation. Formally, we define a fixed point X as:

$$\frac{dX}{dt} = f(X) = 0. \tag{13}$$

We can minimally perturb the system and linearise it with respect to the fixed point X, obtaining:

$$\dot{\epsilon} = f(X + \epsilon) \approx f(X) + f'(X)\epsilon \tag{14}$$

We then say that a fixed point is stable if the linearised system evolves towards zero, that is, if the perturbed system evolves in time towards its initial equilibrium state. This can be examined through the sign of the eigenvalues of the Jacobian $f'(X)$ at that specific fixed point. All the eigenvalues being positive would result in an attractor; all being negative would result in a repeller; and an intermediate situation would result in a saddle point.

All the points that evolve towards a stable point are said to be 'attracted' by it, and the whole set of them constitute the *basin of attraction* of their stable fixed point. In the context of an epigenetic landscape, we have a deterministic potential function that drives (partially) the dynamics of the system. We can interpret each stable equilibrium as a cellular identity reached through a differentiation process (metabolic dynamics), and each basin of attraction as a cellular fate decision.

4 Workflow

Figure 2 summarises the workflow of the tools developed during this project. These scheme is further explained and discussed in the *Methods and Approaches* section.

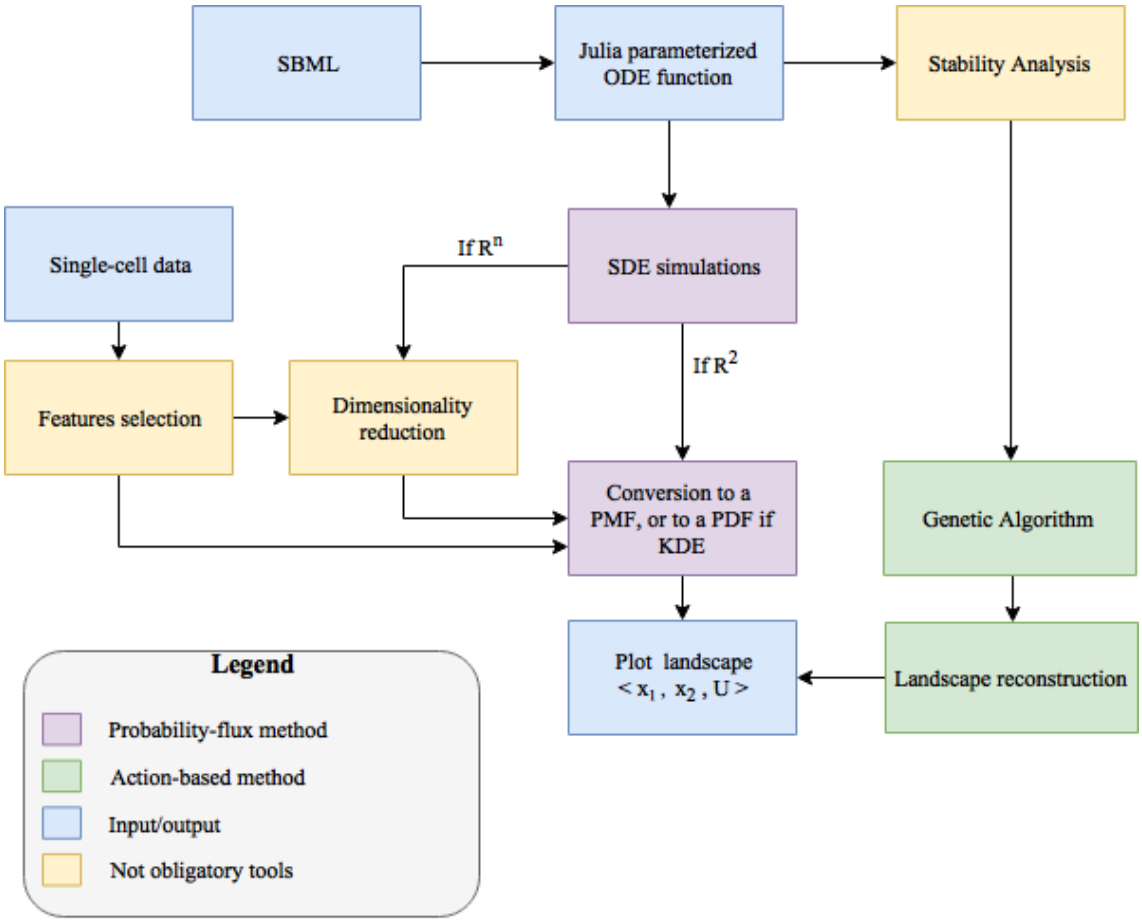


Figure 2: Overall workflow of the tools developed throughout our project. The input is either an SBML file, a parameterized ODE function, or a single-cell transcriptomics data file. Different levels in the diagram represent similar functional steps: the SBML file is equivalent to the Julia parameterized ODE function because the latter is extracted from the file; the single-cell data is equivalent to the SDE simulations data; both features-selection and dimensionality reduction deal with the visualization step; and the lower section corresponds to the landscape reconstruction itself. The legend indicates steps that are not obligatory, and are optional to the user. We also indicate the steps that fall under the different methods of landscape construction. (We note that $n > 2$).

5 Methods and Approaches

5.1 Model Inputs [LT]

5.1.1 Manual input

If the SBML tool is not used, the user is required to write two parameterized ODE Julia functions, $f(X)$ and $g(X)$, that correspond to the deterministic dynamics and noise intensity of the system, respectively. We chose the Julia parameterized functions for the model definition for several reasons:

- They are much easier to define manually.
- They point to defined parameters explicitly, therefore increasing their functionality for providing the possibility of being used by packages that require this information (such as parameter estimation or sensitivity analysis).
- The @ode_def macro (which generates them) automatically performs some calculations that improve the performance of the differential equation solvers. We also take advantage of these calculations, such as the Jacobian, in the stability analysis and PFM assessment tools. (Note that when huge ODE systems are extracted from SBML files, it is suggested in the corresponding section to turn off all these calculations, except the Jacobian, for speed purposes).

Unfortunately, there is a bug in the steady-state problem solver that does not accept parameterized functions with parameter objects. This has obliged us to temporarily solve the problem by requiring the user to enter an additional parameterized function, $f(x)$, but without stated parameters, if the user wants to perform the stability analysis. The function is then designed to get the information about their values from the truly parameterized one.

5.1.2 SBML input

We provide a command-line function, 'SBML_to_ODE.jl', to extract ODE systems (both $f(x)$ and the noise intensity, $g(x)$) from SBML files.

The Systems Biology Markup Language (SBML) is an XML-based format that was first conceived in the year 2000 in a forum (Software Platforms for Systems Biology) supported by the ERATO Japanese Kitano Systems Biology Project.

It was developed to address the challenges of the emergent Systems Biology field related to computational flexibility. More concretely, its need for comprehensive descriptions and integration of complex models, together with the fast development of a huge variety of analysis softwares, made cross-compatibility a problem. SBML is therefore a reference format that unifies Systems Biology model descriptions, allowing for their easy exchange between different softwares [13]. The SBML has a core definition, termed 'level', and a more dynamic added structure called 'versions'. The core used during this project is the SBML level 3 version 2 [12]. This well-defined structure is exploited by softwares to make it functional.

The most fundamental softwares that operate on it are the parsers, which extract its objects and convert them to objects in the desired programming language. Currently, the main library for parsing and manipulating these objects is libSBML, which is written in ISO C and C++ and is wrapped by several programming languages such as Python, Java or Matlab [2]. However, the emergent nature of Julia is also reflected in its lack for one of such wrappers. Since the Julia community is actively working to fill these gaps, and the development of such wrapper has been already suggested, we decided to use libSBML with Python, through the Julia interface provided by the *PyCall.jl* package.

After parsing a SBML model, the next step for the purposes of the project is the extraction of an ODE system from it. Again, there are complete pipelines for the analysis and simulation of SBML models in many programming languages e.g. the ABC Sys-bio package in Python (for ABC inference and SDE GPU simulations) developed by the Theoretical Systems Biology group in 2010 [15], libRoadRunner in ISO C and wrapped by Python (for SDE simulations) [5], or SOSlib, also in ISO C and wrapped by Python [16]. However, this is still lacking in Julia. Consequently, given the impracticality of constantly redefining models manually, and given the

predicted future independence of the Julia SBML parser with the treatment of its retrieved objects, we decided to develop a simple prototype for the extraction of ODEs from SBML files.

We provide a command-line function called 'SBML_to_ODE.jl' The function relies on the Julia core and on the following packages:

- PyCall.jl : provides and interface between Python and Julia.
- libSBML : library for manipulating SBML models, written in ISO C and C++, but with a wrapper for Python.
- ParameterizedFunctions.jl (included in DifferentialEquations.jl) : provides the @ode_def macro required for the parameterized function definitions.

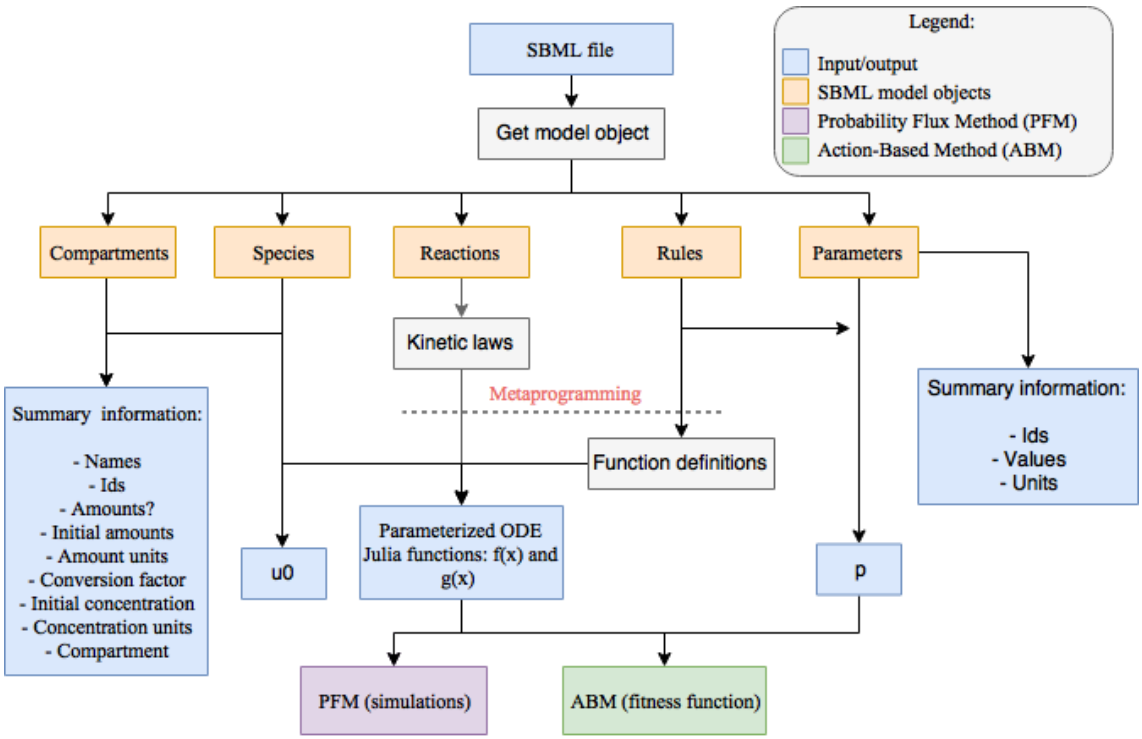


Figure 3: General scheme of the tool for extracting ODE systems from SBML files. First, the model-class object is parsed. Second, the objects contained in it are parsed (orange boxes) and information is derived from them by our tool. Finally, a parameterized ODE function is generated using the Julia metaprogramming functionality and a summary of the information used is returned both for variables (blue summary box in the left) and for parameters (blue summary box in the right. Also, the *u0* and *p* vectors, containing the parameter values and the variables initial conditions respectively, are return so that the generate parameterized function is completely functional (e.g. for simulations).

The structure of the function consists of three main parts:

1. Parse the module using Python, through the libSBML library and the PyCall interface, producing the corresponding Julia objects.

2. Select and structure the information that is relevant for the definition of ODE functions, and combine as a string of Julia code.

3. Using metaprogramming, convert the contents of the string to actual Julia code, therefore defining the parameterized functions.

Given time limitations and our aim in this project to provide a prototype that covers most of the challenges for future scaling, this tool is limited to a specific degree of complexity of the SBML model definition.

We only tested the function for a small number of SBML models (two of which are provided as examples below), and therefore we expect not common exceptions in the model definitions to affect the robustness of the tool. However, we attempt to compensate this by introducing in the function a control that checks the satisfaction of the assumptions and throws and appropriate error otherwise. These assumptions and the rationale behind them are:

- There are no events: these objects define discontinuous jumps in the values of species and/or parameters as the simulation evolves, and involves tracking some variables of the system dynamically during the simulation. We did not explore how to include this advanced feature in our tool, and left it for its future scaling.

- There are no delay functions: these are related to events, and consist of functions that encode an interval of time between their execution and their outputs. Again, it is an advanced future only included in complex models.

- There are no algebraic functions: these are functions that contain implicitly the value of some parameter or species. It is a simple feature, but we found that it is not frequently found, so we did not spend time with it. However,

we make the 'algebraic_rules' variable global so that the user could simply access them, define them manually and solve them using the algebraic symbolic algebra package wrapped from Python (SymPy.jl) and run again the function (removing the corresponding 'if loop' in the control), if the corresponding error message is thrown in first place.

- Only species amounts defined, more than one compartment involved and their volumes not defined, do not concur: This is not a technical limitation, but a consistency control. The ODE simulation requires continuous states to be defined and, consequently, if there is no way of transforming amounts into concentrations, for example, the simulations may give rise to inconsistencies. Also, note that even if concentrations can be obtained, if the different compartment volumes are not constant, the corresponding rate equations for their volumes need to be defined to avoid inconsistencies.

As explained in the 'input' section, the exploit the additional automatic calculations that this macro performs in the simulation and stability analysis steps. However, we observed that when the number of variables is higher than 5, the speed can start to become a problem. An easy solution is to disable some of the calculations (except the Jacobian, needed for the stability analysis) by editing the macro options. This can be done by entering the following code in the terminal:

```
macro ode_def(name,ex,params...)
    opts = Dict{Symbol,Bool}(
      :build_tgrad => false,
      :build_jac => true,
      :build_expjac => false,
      :build_invjac => false,
      :build_invW => false,
      :build_invW_t => false,
      :build_hes => false,
      :build_invhes => false,
```

```
            :build_dpfuncs => false)
        ode_def_opts(name,opts,ex,params...)
    end
```

Given an input of an SBML file as a string (e.g."BIOMD0000000016.xml"), the output consists of two parameterized functions (called 'model_f' (deterministic dynamics part) and 'model_g' (noise intensity), respectively), a parameters summary (called 'parameters_summary') and a variables summary (called 'variables_summary'). The information of the parameters and variables summary is summarized in Figure 3.

Also, a global variable called 'ODEs' that contains the ODE system as an array of string equations is accessible by the user. This might be useful for manually editing the system of equations, since the ODEs array is the previous structure to the parameterized function definition. For this cases, we provide the 'ODEs_to_func.jl' function, which is the final part of the main 'SBML_to_ODE.jl' function and allows for the conversion of ODEs into the Julia function.

We tested the tool with models that satisfied the listed assumptions, and obtaining the same result as the curated ones provided in the BioModels database, from which we retrieved the SBML files. The first example corresponds to the repressilator model [4], the same used as a proof of concept for the landscape formation. The second example corresponds to a model describing oscillations in the MAPK cascade under certain conditions [14]. These examples can be found in Figures 4 and 5 respectively.

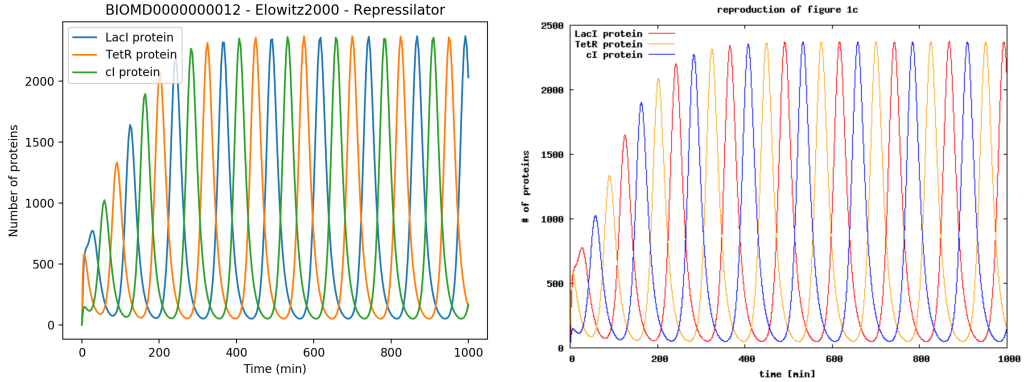


Figure 4: Left) Plot of the simulated parameterized ODE Julia function extracted from the SBML BIOMD0000000012.xml file. Right) Curated plot (thus coinciding with the one in the original publication of the model), provided by the BioModels database, of the simulated ODE extracted from the SBML BIOMD0000000012.xml file.

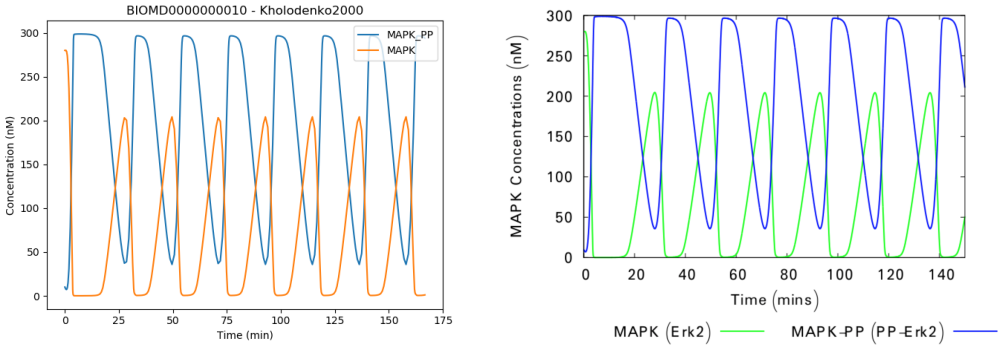


Figure 5: Left) Plot of the simulated parameterized ODE Julia function extracted from the SBML BIOMD0000000010.xml file. Right) Curated plot (thus coinciding with the one in the original publication of the model), provided by the BioModels database, of the simulated ODE extracted from the SBML BIOMD0000000010.xml file.

5.2 Single Cell Data Input [MH, FB]

We can also use single-cell transcriptomics data to generate epigenetic landscapes. The single cell data employed in this project was obtained by Neil Smyth, Southampton University, for the publication by Stumpf et al. on the study of stem cell differentiation [21]. It comprises of 96 genes, the expressions of which were measured across 547 cells over a period of time up to 168 hours.

The data can be read to the workspace in a multitude of ways; segregating cells

based on state (embryonic stem cell (ESC), epiblast-like (EPI), neuroprogenitor cell (NPC)), or segregating reads by time point (0, 24, 48, 72, 96, 120, or 168 hours).

5.2.1 Exploratory Analysis of Data Set [FB]

We provide the user with an initial exploratory data analysis. This can help the user to "get a feel" of the data set, discard outliers and serve as a starting point for a subsequent more detailed analysis.

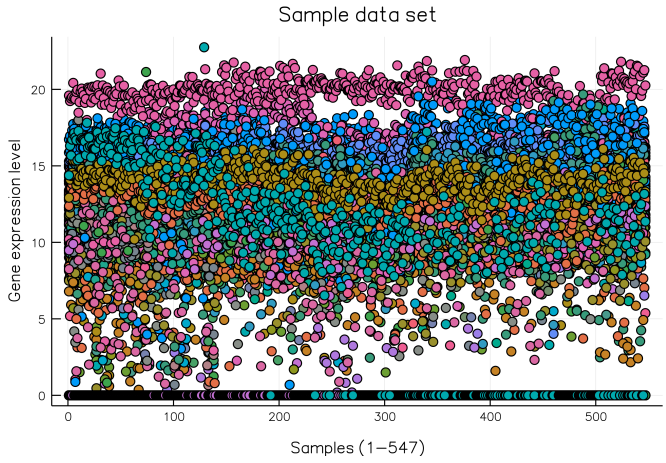


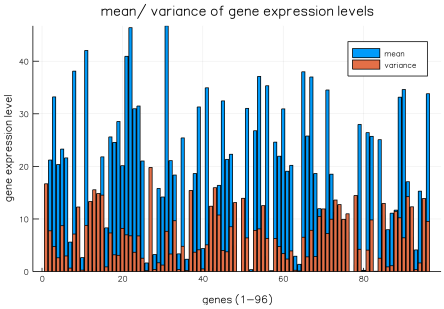
Figure 6: The raw data single cell sample data set: Gene expression levels of different genes (coloured) are plotted against the samples

The raw data plotted in Figure 6 serves as motivation for data analysis that continues in the subsequent sections.

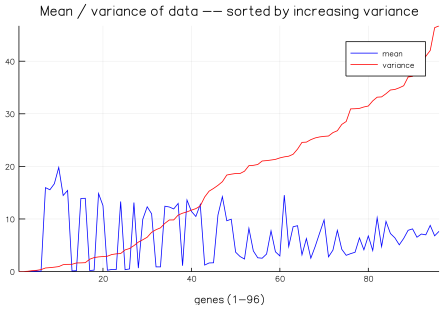
The user can obtain statistics on the whole sample set, such as mean and variance of the different genes, that can be visualised or on specific genes; also more detailed statistics such as percentiles by selecting individual genes can be obtained and subsequently visualised.

Proof-of-concept examples to show how the user can explore the data set are provided in Figures 7a and 8.

From this subset of information, we can already extract some information on our data set: The genes *Gdf3* and *Fgf4* are the ones with the highest mean; *MBP* and



(a) Mean and variance of the the expression of the genes: the mean is highlighted in blue, the variance is highlighted in red



(b) Mean and variance sorted by increasing variance: the mean is highlighted in blue, the variance is highlighted in red

Figure 7: Looking at the whole data set: Mean and variance of the expression of different cells

96×3 DataFrames.DataFrame

Row	gene_name	mean	variance
1	Gdf3	7.65382	46.7227
2	Fgf4	6.794	46.3794
3	Cldn6	8.77944	42.0427
4	Fbxo15	6.98268	40.9346
5	Cdh2	7.13443	38.1144
6	Otx2	6.52783	37.9887
7	Myc	8.11445	37.1106
8	Pou5f1	7.84664	36.9987
⋮			
88	Actb	16.6904	0.835816
89	Ctnnb1	15.5536	0.744714
90	Kdm1a	15.9548	0.680687
91	Mixl1	0.0606634	0.371968
92	Sox1	0.075656	0.234715
93	Ncam1	0.0213954	0.172736
94	T	0.0103924	0.0590774
95	MBP	0.0	0.0
96	Smarca4	0.0	0.0

(a) Mean and variance of the individual cells sorted by increasing variance

Bmi1

Summary Stats:

Mean:	7.747060
Minimum:	0.000000
1st Quartile:	3.791919
Median:	9.663653
3rd Quartile:	11.044525
Maximum:	13.161776
Length:	547
Type:	Float64
Number Missing:	0
% Missing:	0.000000

(b) More detailed information on one individual gene: *Bmi1*

Figure 8: Information on the whole data set as well as on individual genes

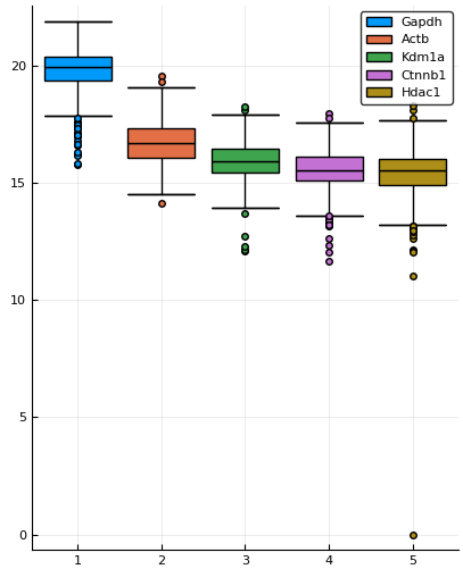


Figure 9: Compare segments of the data: A visualisation of characteristics of specific genes, here the top 5 genes with the highes mean

Smarca have variance 0, checking the mean, we conclude that they are not expressed at all in any of the samples and could be discarded.

For other data analysis such as expression of different genes over time points we refer to [21]; a dimensionality reduction technique will be applied in section 5.4.

5.2.2 Feature Selection [MH]

Primarily, relationships between genes were sought to be established through calculating the pairwise correlations between every possible pair of genes in the dataset. Subsequently more sophisticated methods of extracting relationships between genes were implemented through the *InformationMeasures.jl* package [3]. The calculation of mutual information (MI) between pairs in genes in the data is particularly informative. The mutual information of two random variables describes how much knowing one of these variables reduces uncertainty about the other. Therefore, this measure is ideal for selecting pairs of genes that yield informative landscapes. For further details on the measures from information theory used in this project we refer you to the individual report from [MH].

The code can be found on `https://github.com/burfel/waddington-project/blob/master/src/InformationMeasuresTest.jl`. A summary of the results of per-

forming these information measures on the single-cell data is shown in Table 1.

Subset of data	**Highest Correlated Genes**	**Highest Joint Entropy**	**Highest Mutual Information**
entire dataset	Fgf4, Gdf3	Actb, Gapdh	Fgf4, Pou5f1
24 hour cells	Pou5f1, Trp53	Kdm4c, Zfp24	Ctnnb1, Hdac1
48 hour cells	Dnmt3b, Gapdh	Actb, Dnmt1	Dnmt1, Hdac1
72 hour cells	Dnmt3b, Wdr5	Actb, Lin28a	Hdac1, Lin28a
96 hour cells	Eomes, Gata6	Kdm1a, Tcf7l1	Cdh2, Vim
120 hour cells	Gata4, Gata6	Actb, Zfp281	Cdh2, Cldn6
168 hour cells	Gsc, Utf1	Actb, Hdac1	Actb, Gapdh
Embryonic stem cells	Dnmt3b, Prmt7	Ctcf, Utf1	Trp53, Utf1

Table 1: Summary of information measures of the single cell data.

Calculation of the pairwise correlations of the 96 genes in the data showed that the two most highly correlated genes overall are *Fgf4* and *Gdf3*. However, of the pair of genes with the highest mutual information was found to be *Fgf4* and *Pou5f1*.

These analyses of the data are carried out to provide suggestions to the user of pairs of genes that could generate informative landscapes. However, it is ultimately the user who decides which genes are biologically relevant in the context they are wishing to generate a landscape.

Feature selection is also an active research field in machine learning where variations of dimensionality reduction methods, especially manifold learning methods, are used. However, this will not be the subject here as we apply DR methods only for visualisation purposes.

5.3 Stability Analysis [LT]

Figure 10 shows the work-flow of the 'stab_analysis_num.jl' command-line function, which performs the stability analysis of a dynamical system in $\mathbb{R}^n$.

In brief, the input is a pair of parameterized ODE functions (as explained above, only one will be necessary until a bug in the steady-state solver is be fixed), the

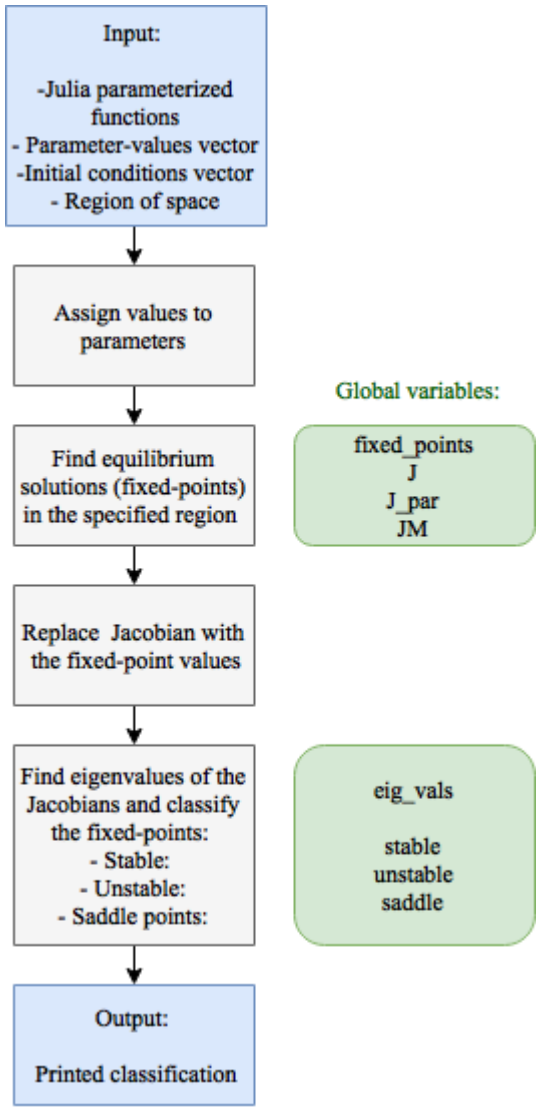


Figure 10: Work-flow of the optional stability analysis step situated in the overall workflow shown in Figure 2.

parameters and initial condition vectors ('p' and 'u0', respectively), and a region of space. The region of space is specified as two extremes of a line ('min' and 'max'), which will then be used to produce a squared coordinates matrix.

The function then benefits from the automatic creation of some objects by the @ode_def macro. These used objects are the list of parameter symbols, the list of differential equations of the system, and the Jacobian.

As outlined in Figure 11, values are automatically assigned to parameters using the function shown in Figure 11, and a Julia steady-state problem is defined and solved. The latter is possible thanks to the SteadyStateDiffEq.jl package (contained in the DifferentialEquations.jl package). The solver is a numerical root-finding algo-

Global variable name	Content
fixed_points	array containing the list of all fixed-points in the region
J	Jacobian (model.symjac object)
J_par	Jacobian with substituted parameter values
JM	array of J_pars, one for each fixed-point
eig_vals	array of eigenvalues, one for each element of the JM array (one for each Jacobian with its corresponding substituted fixed-point)
stable	array with all rounded stable fixed-points
unstable	array with all rounded unstable fixed-points
saddle	array with all rounded saddle fixed-points

Table 2: Names and brief description of the global variables generated by the stability analysis tool. The global variable names and the step of the tool where they are generated is shown in Figure 10.

rithm (currently the unique available) called 'SSRootfind', and because it requires an initial guess of the equilibrium points, our function automatically provides guesses covering the whole region and then integrates the solutions to avoid redundancy.

```
function string_as_varname(s, v)
        s = Symbol(s)
        @eval (($s) = ($v))
end

for i in 1:length(model.params)
    string_as_varname(model.params[i], p[i])
end
```

Figure 11: Function and later execution to assign values to the parameters contained in the parameterized ODE function objects. Parameters are there contained as symbols in the '.params' field-name of the parameterized function called 'model'. They therefore need to be put in correspondence with the 'p' vector containing their values if they want to be used externally to the parameterized function.

The steady-state problem solutions correspond to the fixed-points in the specified region. These are then substituted in the Jacobian and the eigenvalues of the resulting matrix are calculated using the 'eigvals' Julia core function. Then, fixed-points are classified as stable, unstable and saddle points and printed. Green boxes in Figure 10 show the global variables (described in Table 2) that the user can access after the execution of the function.

Figure 12 presents an example of the performance of the tool with the 2D re-

pressilator model defined in section 5.

```
julia> model = @ode_def toymodel begin
           dx1 = a*x1^n/(x1^n+S^n)+b1*S^n/(x2^n+S^n)-k1*x1
           dx2 = a*x2^n/(x2^n+S^n)+b2*S^n/(x1^n+S^n)-k2*x2
       end a b1 b2 k1 k2 n S
(::toymodel) (generic function with 9 methods)

julia> u0=zeros(2); tspan=(0.0,10.0); p=[1,1,1,1,1,4,0.5];

julia> np_model = @ode_def toymodel begin
           dx1 = a*x1^n/(x1^n+S^n)+b1*S^n/(x2^n+S^n)-k1*x1
           dx2 = a*x2^n/(x2^n+S^n)+b2*S^n/(x1^n+S^n)-k2*x2
       end
(::toymodel) (generic function with 9 methods)

julia> stab_analysis(model, np_model, u0, tspan, p, 0.0, 3.0)

In range 0.0 to 3.0:

stable:
[1.0, 1.0]
[0.0, 2.0]
[2.0, 0.0]

unstable:

saddle:
[0.5, 1.5]
[1.5, 0.5]
```

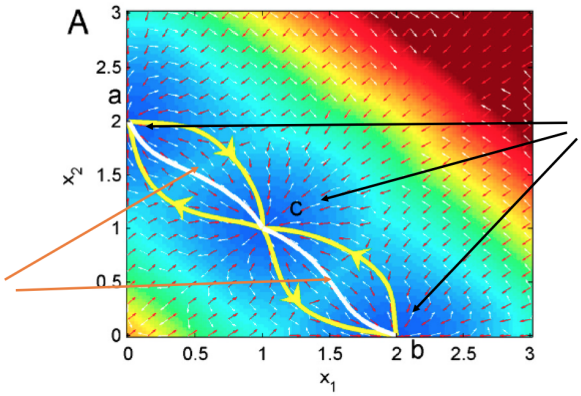


Figure 12: Example of the performance of our stability-analysis tool with the 2D repressilator model defined in section 5. The right column shows the definition of the parameterized functions, the calling of the function in the [0.0, 3.0] interval, and the output. The plot in the right is a contour plot with the $-\nabla U$ and f_c vector fields on the top, taken from [25] and with black and brown arrows added to point to the stable and saddle points identified by the function.

5.4 Dimensionality Reduction [FB]

5.4.1 Motivation

If the number of dimensions, ie genes in our case, is large (96 in case of the sample single cell data set), the number of possible states in this space grows exponentially. To put it differently, with increasing dimensionality it becomes harder to sample from the space; even for a small model of 3 genes this can be a major challenge. Simulations take very long to run; similarly, to train machine learning (ML) algorithms on such high-dimensional data sets is very time-intensive and cause over-fitting. This problem is often referred to as *Curse of dimensionality*.

Therefore, it is necessary to reduce the data to fewer dimensions to make algorithms more efficient. Another reason to apply dimensionality reduction (DR) is for the purpose of visualisation. It is hard for humans to comprehend data in many dimensions. In our specific example this translates to the obvious fact that only two genes at a time (with the amplitude of the quasi-potential in a third dimension) can be visualised in our 3D world, or respectively projected onto a 2D plane.

Dimensionality reduction (DR) reduces the number of features, in our case genes by creating new linear, or non-linear, combinations of the original features.

In the following the widely used Principal Component Analysis (PCA) method will be explained in greater detail as it is a simple yet powerful approach to DR as well as Singular value decomposition (SVD). Non-linear, specifically manifold learning algorithms, that have or can be applied to our given data set in order to tackle the challenges will be motivated.

As DR reduction methods, beyond the use for this project, have been examined by the group but are not subject of this group report, we refer to [FB]'s group report for DR.

A good overview of the main DR methods, including manifold learning methods, could not be found; we created one with explanation, and advantages and disadvantages for which we refer to *https://github.com/burfel/waddington-project/tree/master/papers/dimRed*.

Figure 13 gives a classification and overview of the plethora of available DR methods that are mainly available in Julia and which ones are working. For a bigger version we refer to [FB]'s individual report.

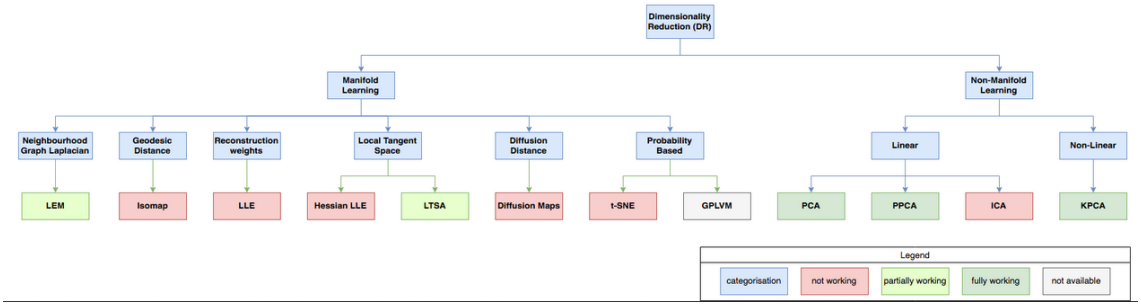


Figure 13: An overview of the main DR methods classified by manifold and non-manifold learning techniques, linear and non-linear methods and core elements of the algorithms. Highlighted in dark green are the methods which are working, highlighted in light green are the ones that partially working, in red are the ones that (as of today) are not working, shaded in grey is not implemented in Julia yet.

5.4.2 Principal Component Analysis (PCA)

Mathematically speaking, all PCA is is a basis transformation with an optional subsequent projection. It is linear in the sense that it only takes linear combinations of the old basis (features, here: genes) to form the new basis.

In other words, PCA finds the principal components (PCs) that explain the data "best", ie containing most information which is usually quantified by the total variance captured by the basis elements. The PCs are, therefore, the directions of maximum variance in the high-dimensional data. Projecting the data onto this smaller dimensional subspace (spanned by the PCs) will reduce the dimensionality of the original feature space and still retain most of the information (variance).
We will see that the new basis can be obtained by performing an eigendecomposition on the initial data set, and in fact equals the eigenvectors.

PCA can be summed up in three simple steps:

1. Eigendecomposition: Computing eigenvectors and eigenvalues

2. Selecting principal components

3. Projection onto the new feature space

Eigendecomposition: Computing eigenvectors and eigenvalues

Eigenvectors and eigenvalues of a covariance (or correlation) matrix are at the very

centre of PCA. The eigenvectors (PCs) determine the direction of the new feature space; the eigenvalues determine their magnitude. We can obtain the eigenvectors and eigenvalues from the covariance matrix or correlation matrix or perform singular value decomposition (see approach SVD / `generalPCA.jl`)

For the reader who is unfamiliar to eigendecomposition, we insert a brief mathematical interlude on this central result in linear algebra. To learn about how to find eigenvalues and its corresponding eigenvectors, we refer to [20].

Let $A \in \mathbb{R}^{d,d}$ be a matrix. A non-zero vector u is an eigenvector of A with a corresponding eigenvalaue λ if

$$Au = \lambda u. \tag{15}$$

It can be shown with basic linear algebra results that every (non-singular) diagonalisable matrix A can be factorised into a canonical form, whereby A is represented in terms of its eigenvalues and eigenvectors. (Note, the matrix A is diagonalisable if and only if the algebraic multiplicity equals the geometric multiplicity of each eigenvalues, ie A has d distinct eigenvalues.)

This result follows from the *Spectral Decomposition Theorem* [20].

Theorem 5.1. *If $A \in \mathbb{R}^{d,d}$ is a symmetric matrix of rank k, then there exists an orthonormal basis of $\mathbb{R}^d$, namely $u_1, u_2, ..., u_d$, such that each u_i is an eigenvector of A. Furthermore, A can be written as $A = \sum_{i=1}^{d} \lambda_i u_i u_i^T$, where each λ_i is the eigenvalue corresponding to the eigenvector u_i. Equivalently, this can be written as $A = UDU^T$, where the columns of U are the vectors $u_1, u_2, ..., u_d$, and D is a diagonal matrix with $D_{i,i} = \lambda_i$ and for $i \neq j$ we have $D_{i,j} = 0$. Finally, the number of $\lambda_i \neq 0$ determines the rank of the matrix; the eigenvectors which correspond to the non-zero eigenvalues span the range of A, and the eigenvectors which correspond to zero eigenvalues span the null space of A.*

Assuming that the reader now knows the meaning and power of eigendecomposition, we now go back to PCA.

a. PCA with covariance matrix: Here, before starting, one usually standardises

the data. The classic approach of PCA is to perform the eigendecomposition of the covariance matrix Σ, which is a $d \times d$ matrix whose elements represent the covariance between two features calculated by

$$\sigma_{jk} = \frac{1}{n-1} \sum_{i=1}^{N} (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x_k})$$

which equals the matrix equation

$$\Sigma = \frac{1}{n-1}((X - \bar{x})^{\mathrm{T}}(X - \bar{x}))$$

where $\bar{x}$ is the mean vector $\bar{x} = \sum_{k=1}^{n} x_i$.

The mean vector is a d-dimensional vector where each value in this vector represents the sample mean of a feature column in the data set.

b. PCA with correlation matrix: Instead of the covariance matrix, one can use the correlation matrix. Since the correlation matrix can be understood as the normalised covariance matrix, the eigendecomposition on the covariance matrix yields the same results (eigenvectors and eigenvalues) as one on the correlation matrix in case of standardised input data. In fact, we could show that the following three approaches yield the same results:

- eigendecomposition of the covariance matrix after standardising the data
- eigendecomposition of the correlation matrix
- eigendecomposition of the correlation matrix afer standardising the data.

For more computational efficiency, one often uses SVD for the eigendecomposition (see section 5.4.3).

Selecting the principal components

The eigenvectors form the new basis, ie the directions of the new axes and all have the same unit length 1. To decide onto which eigenvectors we want to project down to, ie which eigenvectors "to keep", we look at the corresponding eigenvalues. The

eigenvectors with the highest eigenvalues capture the most distribution or information of the data and the ones that should be kept. For this purpose, we sort the eigenvalues from highest to lowest in order to later choose the top k eigenvectors, where k is the number of dimensions of the feature subspace and $k \leq d$.
A frequently asked question is: What is the size of k that represents the data well?

For that, we construct the projection matrix P by "concatenating" the just computed d eigenvectors. Each of those eigenvectors comes with an eigenvalue (we will call them *eigenpairs*) which can be interpreted as the "length" or "magnitude" of the corresponding eigenvector.

By computing the "explained variance", ie amount of information each PC (eigenvector) captures, on our data set, we see that most of the information, approximately 21.85 % to be precise, can be explained by the first PC. The subsequent components each bear information between approximately 8.36% and less than 0.129%. Together, the first two PCs account for more than 30% (30.22%) of the information. The third PC explains less than a third of what the first components; as a result, three PCs capture 36.78% of the total variance. We examined that 75 dimensions can be seen as benchmark as the first 75 PCs are enough to retrieve almost all (99.06%) of the variance of the data. For details, we refer to Figure 14.

To come back to the question above, we leave that up to the user who can choose either a fixed number of PCs spanning the new feature space or a percentage of explained variance that the PCs represent in total based on the computation of explained variance.

Projection onto the new feature space

To project the data set onto the new feature subspace, we take top k eigenvectors of the just constructed projection matrix P which we will call P_k. Last but not least, we transform the original data set X via P_k into Y in the new k-dimensional feature subspace by computing $Y = X \times P_k$.

For a more rigorous mathematical definition the reader might want to consult

[20].

5.4.3 Singular Value Decomposition (SVD)

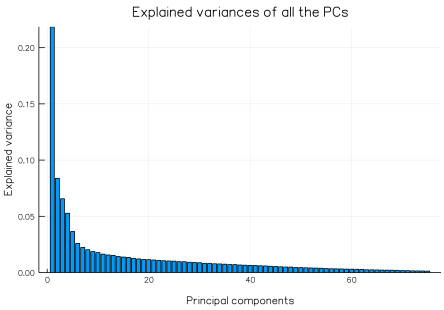
A bit less intuitive than the eigendecomposition of the covariance or the correlation matrix, is the SVD.

Let $A \in \mathbb{R}^{m,n}$ be a matrix of rank r. If $m \neq n$, the eigenvalue decomposition in 5.1 cannot be applied. However, we can perform another decomposition on A, the so-called *Singular Value Decomposition* (SVD) which makes use of the *SVD Theorem* [20].

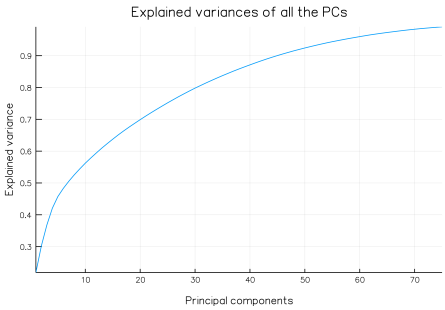
Theorem 5.2. *Let $A \in \mathbb{R}^{m,n}$ with rank r. Then $A = UDV^T$, where D is a $r \times r$ matrix with non-zero singular values of A and the columns of U, V are orthonormal left and right singular vectors of A. Furthermore, for all i, $D_{i,i}^2$ is an eigenvalue of A^TA, the ith column of V is the corresponding eigenvector of A^TA and the ith column of U is the corresponding eigenvector of AA^T.*

For the theory, we refer to [20]. In `generalPCA.jl`, another PCA method based on SVD was implemented.

As a proof-of-concept example, we apply our PCA method to the given sample data set 5.2. Figure 14 shows the explained variance of the principle components, computed from the singular values of the data matrix (rows correspond to features, ie genes, and columns correspond to the samples).



(a) Explained variance across all the single PCs

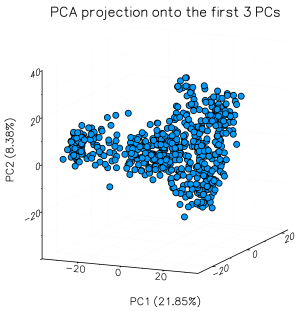


(b) Explained cumulative variance as we add more PCs

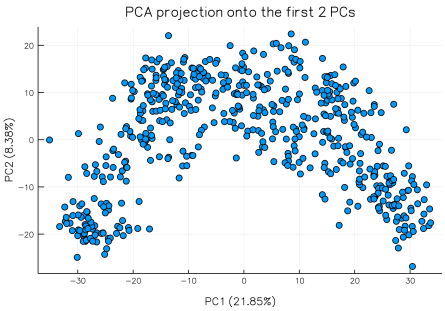
Figure 14: Explained variances of all the PCs: The first three components account for more than 30% of the total variance of the data; 75 dimensions are enough to reconstruct the data almost completely.

Higher dimensions might be interesting from a data compression point of view; as we are only interested in the visualisation, we only look at two or three dimensions.

Figure 15 shows the projection onto the first two principal components and on the first three principal components.



(a) Projection onto the first 3 PCs

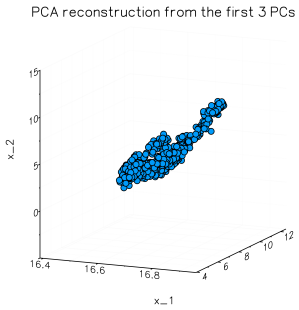


(b) Projection onto the first 2 PCs

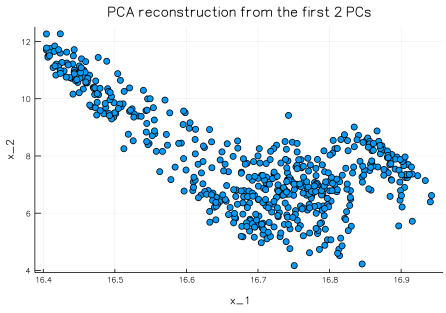
Figure 15: Projections after applying PCA: The blue dots are the 576 cells from our sample data set, the new axes are the PCs

As we expected and have seen from previous analysis, the third component does not contain much information (6.56%); a projection onto the first two principal components seems sufficient.

Additionally, we try to reconstruct the data after reducing the dimensions and compare it to the original data set.

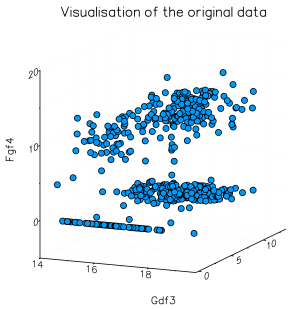


(a) Reconstruction from the first 3 PCs

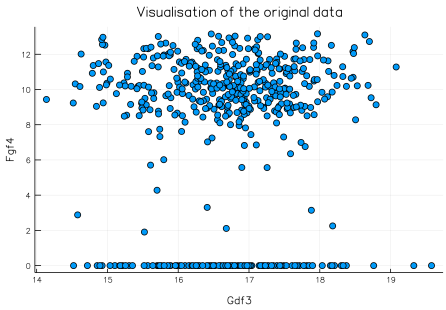


(b) Reconstruction from the first 2 PCs

Figure 16: Reconstruction of the characteristics of the original data set (first 3 dimensions) after projecting them down; again the blue dots are the 576 cells from our sample data set



(a) Original data (first 3 dimensions)

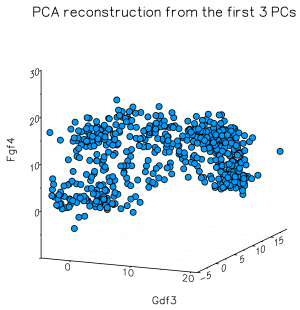


(b) Original data (first 2 dimensions)

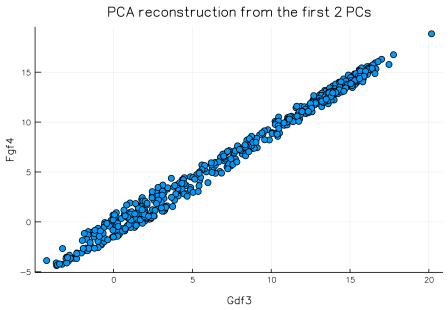
Figure 17: Visualisation of the original data set

Comparing the the reconstruction to the original data set (first three dimensions, namely genes *Actb*, *Bmil* and *Bmp4*), we can see that the characteristics of the original data can be retrieved. However, it is difficult to judge based on plots as we cannot visualise the high-dimensional original data set and with any three or two dimensions we visualise, we only show a small part of the data set. Therefore, it is questionable how meaningful it is to compare to the original data set, as it represents only a fraction of the original data set.

However, as the initial basis (dimensions, or genes) is not sorted by variance, it might also be interesting to reconstruct the dimensions, ie genes, with the highest variance. From the exploratory data analysis section, we know that these are the genes *Gdf3*, *Fgf4* and *Cldn6*. The reconstruction onto these dimensions is found in Figure 18.



(a) Reconstruction from the first 3 PCs



(b) Reconstruction from the first 2 PCs

Figure 18: Reconstruction of the characteristics of the original data set (genes Gdf3, Fgf4, Cldn6) after projecting them down; again the blue dots are the 576 cells from our sample data set

As expected we see a higher spread of the data points in three dimensions compared to the rather dense version in Figure 16. For the two dimensions we can see that it is very distinct to the first reconstruction in 16. Indeed, we discovered a highly linear relationship between these two genes (of highest variance amongst the samples) and looking in our sample data set, we can indeed see that the expression of these two genes are highly correlated between each other. A thing to further investigate would be, whether these two genes are eg controlled by the same transcription factor or found in the same pathway. As they highly vary across samples, we might conclude that they are of special importance or of special function and eg no housekeeping genes that are equally expressed across all cells.

As it is hard to capture how the new dimensions (PCs) lie in space, we try to visualise each of them by their unique "barcode", see Figures 19 - 21. As each PC, ie new basis, is a linear combination of the initial basis, we can think of this barcode as a function of the original dimensions:

$$PC_i := f(x_1, x_2, ..., x_N) = \sum_{i=1}^{N} w_i \cdot x_i \tag{16}$$

where x_i denotes the elements of the inital basis, ie in our case genes and w_i are the coefficients or weights of each initial component. In our case $N = 96$, as we have 96 in the input data set. From the barcode, we identified *Fgf4* as highest contributor,

ie gene with the highest absolute value of the w_i for the first PC (see Figure 19), *Cldn6* for the second PC (see Figure 20), and *Cdh2* for the third PC (see ref 21). We notice that both *Fgf4* as well as *Cdh2* are amongst the top 5 genes with the highest variance.

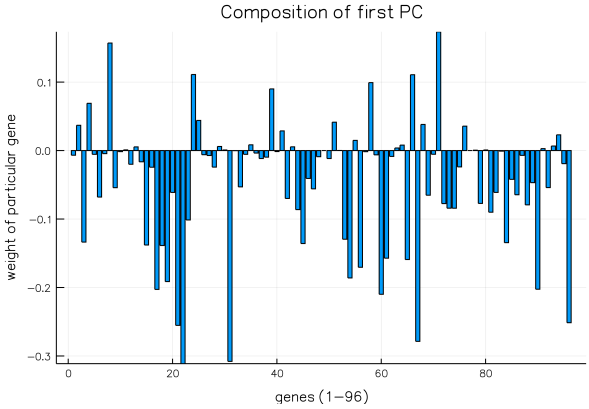


Figure 19: "Barcode" of PC1: Its composition represented by the weights each of the individual 96 genes contribute

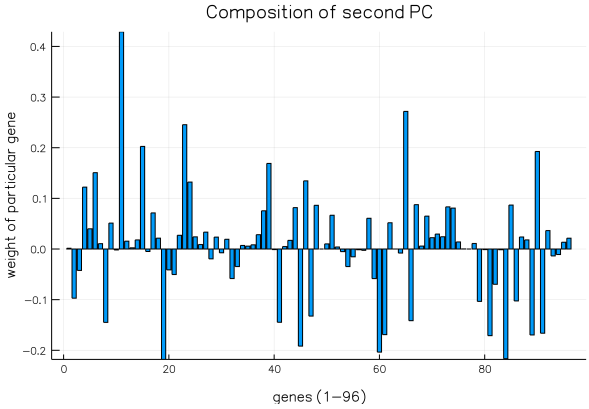


Figure 20: "Barcode" of PC2: Its composition represented by the weights each of the individual 96 genes contributes

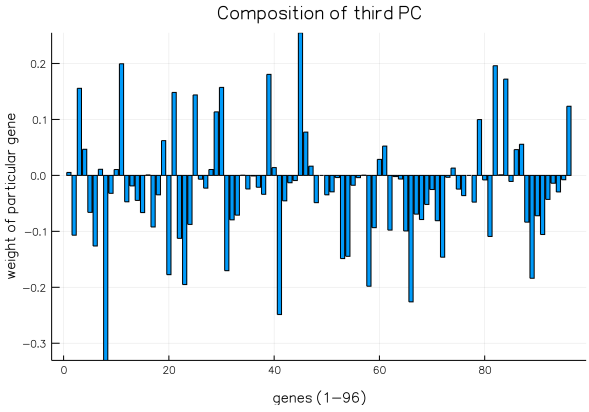


Figure 21: "Barcode" of PC2: Its composition represented by the weights each of the individual 96 genes contributes

Plotting the contribution of all genes summed over all PCs in Figure 22 showed that *Tubb3* contributed the most and *MBP* the least; *MBP* in fact shows no expression at all. We note, that this is in line with the initial exploratory analysis as we noticed that both *MBP* and *Smarca4* have zero mean and zero variance; we could have discarded both genes at that point. However, there is no significant variance which is in line with what we expected.

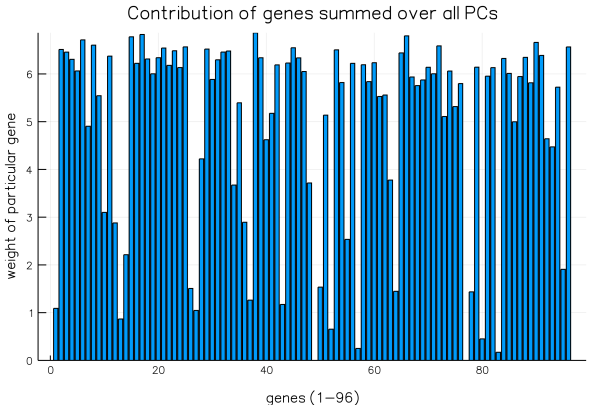


Figure 22: Contribution of all genes over all PCs represented as absolute values of the weights over all PCs

For further studies it might be interesting to segment the data according to cell types or time points and analyse them separately to detect patterns. Subsequent k-means clustering was shown to identify three different groups in the data set [21].

5.4.4 Manifold Learning

Non-linear DR methods also include Manifold Learning Algorithms; this it the part where a little bit of differential geometry comes into play. Certainly, manifolds are of great interest in differential geometry; for a (mathematically rigorous) definition of manifolds we refer to [24]. Unlike simple linear DR methods like PCA, manifold learning techniques consider the intrinsic geometry of the data; they are based on the assumption that the given data lies on an embedded non-linear manifold within the high-dimensional space. Instead of projecting the data onto the "right" set of directions, we look for a manifold close to the data, project it onto that manifold and "unfold" the manifold for representation. We do this in a way that each data point is assigned a low dimensional representation while keeping its essential geometric properties such as relative distances between points unchanged. The data, assuming the manifold is of low enough dimension, can then be visualised in this lower-dimensional space.

Another major difference between (most) manifold learning techniques and linear methods is that only local features of the data are considered opposed to global ones by eg taking correlations of the entire data set. The typical manifold learning problem is unsupervised: it learns the high-dimensional structure of the data from the data itself. However, supervised versions exist as well.

The main problem to which different methods suggest different solutions is: How to construct a representation for the data lying on such a low dimensional manifold embedded in a high dimensional space? What are criteria for "good" and "bad" representations?

One standard data set to compare manifold learning techniques is the *Swiss roll* plotted in Figure 23 (not in Julia).

Standard manifold learning techniques include t-SNE (which is largely applied to single-cell data), Locally-linear embedding (LLE) and various variants, Isometric mapping (Isomap), Diffusion maps, Laplacian eigenmaps, local tangent space alignments and many more and by which various manifold learning methods. For a

complete overview, we refer again to the github repository.

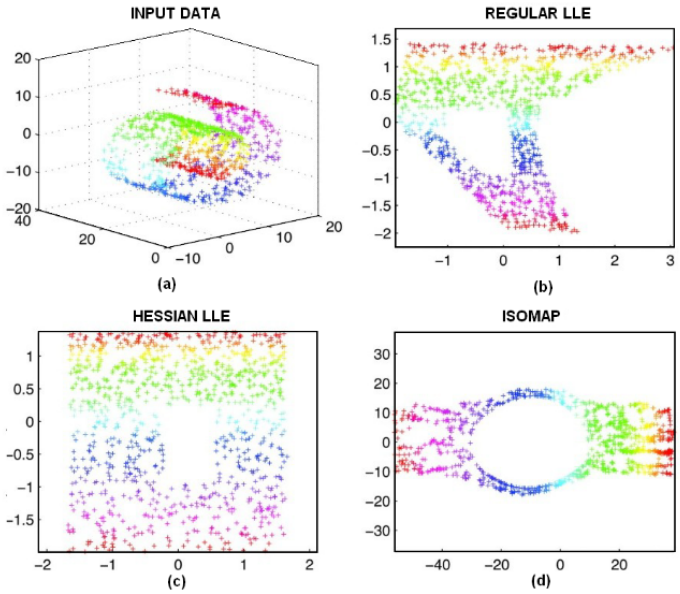


Figure 23: The *Swiss roll* unfolded: Only HLLE can handle non-convexity. Isomap and regular LLE find the hole but the set is distorted.

5.5 Simulations of a SDE model [LD]

As explained previously, both PFM and KDE are methods used to calculate the potential landscape. In both methods, the landscape is based on the density distribution of the data which can come from either single-cell experiments or simulations of a model. As a part of our package is to provide an epigenetic landscape from a SDE model, we are interested in the generation of a large number of simulations of such a model.

5.5.1 Approach

Implementation

The main idea was to provide a method allowing automatic generation of a large number of simulations. The first question we asked ourselves while tackling this problem is what should be the inputs and outputs of such a method. As it was the most natural output and the same format as the single-cell data, the method provides a matrix of the last result of each SDE simulation. This is then a convenient

input for the KDE and PFM methods. As inputs, the user provides the boundaries of the variable space, the SDE model to be tested, and the number of simulations required. Moreover Julia is well known for its *DifferentialEquations.jl* package, so it was quite natural to use it for the simulations.

Some other questions we considered included (1) how to define the initial conditions of each SDE, (2) should we let the user choose them, (3) should we allow a random or a fix time for the SDE, (4) should we provide other points than the last one of each trajectory.

We answered these questions considering what was the main objective of the method as part of the package, so with the idea that the final aim was the computation of the potential landscape from the data.

(1) As we wanted to have a final distribution of points without favoring any particular initial conditions, it is somehow natural in this context to choose uniform random initial conditions within the boundaries chosen by the user.

(2) Part of the answer can be found in (1) but except the boundaries, the user cannot choose a single or a set of initial points as we consider that the final aim is to provide quality data for the landscape.

(3) That was a debated question, we finally chose to use only fix time for simulations. A suitably chosen time allows convergence of the PDF. Times smaller than this will lead to a sample strongly dependent on initial condition while times larger than this lead to unnecessary computational cost.

(4) Like (3), the question was quite open, the main problem was still to think about the final aim : the potential landscape. In the context of building a landscape from a SDE model with either KDE or PFM, the simulations should be independent one from each other which is not the case if we consider taking multiple points along a SDE trajectory. Indeed, on a SDE trajectory, the next point is really dependent on the previous one. That is why we decided to use only the last point of each trajectory. An example implementation can be seen in Figures 24 and 25. Figure 24 defines a 2D model we wish to simulate. Figure 25 shows the function and its

inputs.

```
using Plots, DifferentialEquations

##### DEFINITION OF THE PROBLEM
nb_bin=10 #discretization of the space for the frequency matrix
nb_sim=10000 #number of simmulations
end_time= 10.0 #end time for each simulation

##### DEFINITION OF THE MODEL
a1=1
a2=1
b1=1
b2=1
k1=1
k2=1
n=4
S=0.5
lambda=0.01

##### 2D model
dim=2
boundaries=[[0.0,3.0] [0.0,3.0]] #boundaries for each dimension
tuple_dim=(nb_bin,nb_bin) #tuple of dimension of the discretized space
function model(du,u,p,t)
    du[1] = a1*u[1]^n/(u[1]^n+S^n)+b1*S^n/(u[2]^n+S^n)-k1*u[1]
    du[2] = a2*u[2]^n/(u[2]^n+S^n)+b2*S^n/(u[1]^n+S^n)-k2*u[2]
end

function sig_model(du,u,p,t)
    du[1] = sqrt(abs(a1*u[1]^n/(u[1]^n+S^n)+b1*S^n/(u[2]^n+S^n)-k1*u[1]))
    du[2] = sqrt(abs(a2*u[2]^n/(u[2]^n+S^n)+b2*S^n/(u[1]^n+S^n)-k2*u[2]))
end
```

Figure 24: Example of the definition of the 2D model problem, section 5.1, plus some parameters for the simulation method

```
##### SIMULATIONS
@time state=simulations_CPU(nb_sim,boundaries,end_time,model,sig_model,dim)
```

Figure 25: Call of the simulations method with the previously defined parameters and model

Limitations

One major issue we encounter is the number of simulations required. Both KDE

and PFM quality highly depends on this factor and even if the time of computation of the simulations increase linearly with the number of simulations, as shown in Figure 26, which is comparatively adequate, it is a main concern. The time of computation also increases slightly with the dimension of the problem. Moreover the higher the dimensionality of the problem, the more simulations are needed, mostly to cover well the space during the initialization phase. This is part of the curse of dimensionality (discussed previously). In particular, the number of simulations required rises exponentially with dimension d. For p points per dimension, you require p^d simulations. We needed to consider solutions to counter this.

We employed the idea of using more points on each trajectory, decreasing the dependency by taking the points far from each other. However, for us the best option was to increase dramatically the number of simulations without making time requirement concession. For this, we worked as far as we could on a SDE solver using GPU computing.

Number of sim	10	100	1000	10000	100000
2D model	0,05	0,48	4,53	45,23	473,72
3D model	0,07	0,66	6,04	61,95	598,92

Figure 26: Table presenting the time of computation required in seconds function of number of simulations for specific 2D and 3D models

5.5.2 Graphics Processing Unit (GPU)

We decided to use General-purpose computing on GPU as it has been a really hot topic for few years now and the results are generally impressive. The idea is to use the Graphic Card of a computer which basically contains dozens to hundreds of small processors, and parallelize the computation of methods and works that can be. In our case, each simulation is totally independent from another so each can be done in parallel on different threads. This drastically lowers the time required for the simulations and allows us to perform many more in the same amount of time.

It is a much easier thing to discuss than to implement. We decided to use CUDA which is a complement language widely used in GPU programming for NVIDIA Graphic Cards. It allows the programming of methods on the GPU using another language like C or Julia. We began with an implementation of a small solver in CUDA/C to familiarise ourselves with the process.

Sadly after many attempts, we could not use the Julia packages for GPU computing because of many compatibility problems. As Julia is evolving very quickly with a target on version 0.7.0 expected later this year, some problems may be fixed at such a time and Julia might provide a great GPU tool.

We still present some results comparing SDE simulations on Julia/CPU, C/CPU and C/GPU in order to get an idea of the power of GPU programming, and explain why we think it is a clear future focus for further implementations.

Nb of sims	10	100	1000	10000
Julia	0,05	0,48	4,53	45,23
C++	0,03	0,12	0,35	3,01
C/CUDA (GPU)	0,54	0,59	0,63	1

Figure 27: Table representing the time of computation required in seconds function of the number of simulations of the 2D model, written either in Julia (CPU), C++ (CPU) or C/CUDA (GPU)

The programs we provide in C/CUDA and C++ are entirely not optimal, and we have some problems of memory for high numbers of simulations with the GPU program. However, the results presented still show how powerful GPU programming can be and why we think it should be a main focus for future work.

Given the short evaluation of the results in Figure 27, we can see that Julia is clearly slower than the C++ program. There are many reasons behind this; it is not exactly the same algorithm, not the same random number generator, less outputs and so probably less calculus in our C++ code.

We can also discuss the difference between the C++ and C/CUDA programs; both have a really similar structure and differ mostly on the random number generation and the use of the GPU. We can see that the C++ code is quicker for low

number of simulations; that is natural due to the process behind general purpose GPU (GPGPU). Information are primarily sent from CPU memory to GPU memory, and then the computation is done with GPU variables. Finally the results are transfered again from GPU memory to CPU memory. This process takes time and should be put in balance with the time of computation itself. For high number of simulations, this time of transport becomes worthy thanks to the performance of the computation on GPU.

Please notice further issues with GPU programming that must be tackled in general or in Julia particularly:

- Probable need of implementation of a SDE solver in Julia: as we could not try the CUDAnative package or other Julia GPU packages, we cannot know how far we have to implement SDE solver or if we can use already existing methods on the GPU. In C, we had to go back to basic and code a solver following the Euler-Maruyama method, it might be the same for Julia.

- Problem of Random Number Generation: one main problem of GPU programming, as many random number generator are pseudo-random and use seed as initialization. This is a real problem on GPU as we must take care of each thread seeding etc. As future work, we should look further the Random 123 method which might be an interesting alternative possibility.

5.6 PFM and KDE implementations [LD]

As explained previously, we use the KDE and the PFM to access the quasi-potential from some data, either from single-cell experiments or simulations of a SDE model. Both methods approximate the density distribution of the data and link it to the potential according to the equation in (3).

Both methods share the same inputs and output. The main inputs are the two dimensions of interest, i.e. the dimensions the landscape will be plot from, the data either from single-cell experiments or simulations. Some parameters needed for these

methods are also defined in Figure 24, such as the boundaries of the problem which do not change and the discretization parameter. The output is a matrix containing the values of the potential U in some points in space which are also available via the span2D method (a method providing the line space values of each dimension from the boundaries and the discretization parameter).

5.6.1 PFM

The idea of PFM is to discretize the space considered by the user and compute the density distribution of the data by counting the number of points falling in each discrete part of the space.

It is possible to perform the PFM in N dimensions and provide the density distribution of the data as it results from a multidimensional random variables, but it is impossible to plot this then when $N > 2$. We chose to implement a method which takes two dimensions of interest and returns the 2D density distribution. We use the data considering the dimensions of interest, and other dimensions are not taken into account. Moreover, if (optional) dimensionality reduction is performed, the results can be in two dimensions (for example PC1 and PC2 from PCA), so this also suits the inputs of this method (by choosing PC1 as the first dimension of interest and PC2 as the second one).

```
##### PFM reduction 2D
dim1=1 #dimension of interest 1
dim2=2 #dimension of interest 2
U_pfm=PFM_reduction_2D(state,boundaries,nb_bin,dim1,dim2)
span_2D=Span_2D(boundaries,nb_bin,dim1,dim2)
x=span_2D[1,:]
y=span_2D[2,:]

plotly()
surface(y,x,U_pfm)
```

Figure 28: Call of the PFM function after, for example, the definition of the problem in Figure 24 and the simulations in Figure 25

5.6.2 KDE

KDE is a particular PFM. It differs from it by computing an approximation function of the density as seen in equation (10). The same question as the PFM can be asked regarding the reduction to a 2D problem to be able to plot the results. We implement an analogous solution.

The only significant problem is the choice of the bandwidth h that we have not yet dealt with. Some papers propose methods to obtain the optimal bandwidth for a given problem, but they use the second derivative of the exact density function which is not available in our case. We decided to use an approximation of the optimal bandwidth found in [26]:

$$h \propto n^{-\frac{1}{4+d}} \tag{17}$$

where n is the number of points in the data and d the dimension of the problem. As we are always in 2D in the KDE implementation, we use:

$$h = n^{-\frac{1}{6}}. \tag{18}$$

```
##### KDE reduction 2D
dim1=1 #dimension of interest 1
dim2=2 #dimension of interest 2
U_kde=KDE_reduction_2D(state,boundaries,nb_bin,dim1,dim2)
span_2D=Span_2D(boundaries,nb_bin,dim1,dim2)

x=span_2D[1,:]
y=span_2D[2,:]

plotly()
surface(y,x,U_kde)
```

Figure 29: Call of the KDE function after, for example, the definition of the problem in Figure 24 and the simulations in Figure 25

5.7 Action-based Method [LD]

The aim of the Action-Based Method (ABM) is also to provide the potential landscape, but from a totally different perspective than either PFM or KDE. In this section, we present how we implement the computation of the MAP and further the computation of the landscape from the MAP function and the stable points of a dynamical system.

5.7.1 Genetic Algorithm [LD, FB]

The ABM computes the Minimum Action Path (MAP) between two points, which represents an optimization problem. An easy way to generate a solution to this problem is by using a Genetic Algorithm (GA), an evolutionary algorithm including natural selection inspired from biology. A GA is metaheuristics that achieves a trade-off of accuracy of results and computational intensity. Therefore, we can obtain fairly good results for an optimization problem in an acceptable amount of time. This is due to the fact that the algorithm only samples a set of solutions that would be too large to be completely sampled (see Figure 30). Minimal assumptions are made about the optimization problem at hand. However metaheuristics are not guaranteed to find the globally optimal solution but might get stuck in a local minimum.

Motivated by the issue that the number of paths to analyze is impossible to tackle, we developed a GA to compute solutions to our problem. The steps of a genetic algorithm are (1) creating an initial population of paths, (2) selecting the best paths from this generation (also called *parents*) according to some metric/fitness function, (3) mixing the selected paths (referred to as *mating*) to hopefully get better paths. These steps mimic natural selection. We begin with an initial population which represents a bad approximation of the set of possible solutions; the GA then uses tools to constantly change this set with the aim of finding better solutions.

One major problem of GA is the dependence of the result on the initial population. As metaheuristic algorithms, GA work on heuristics which are simplified

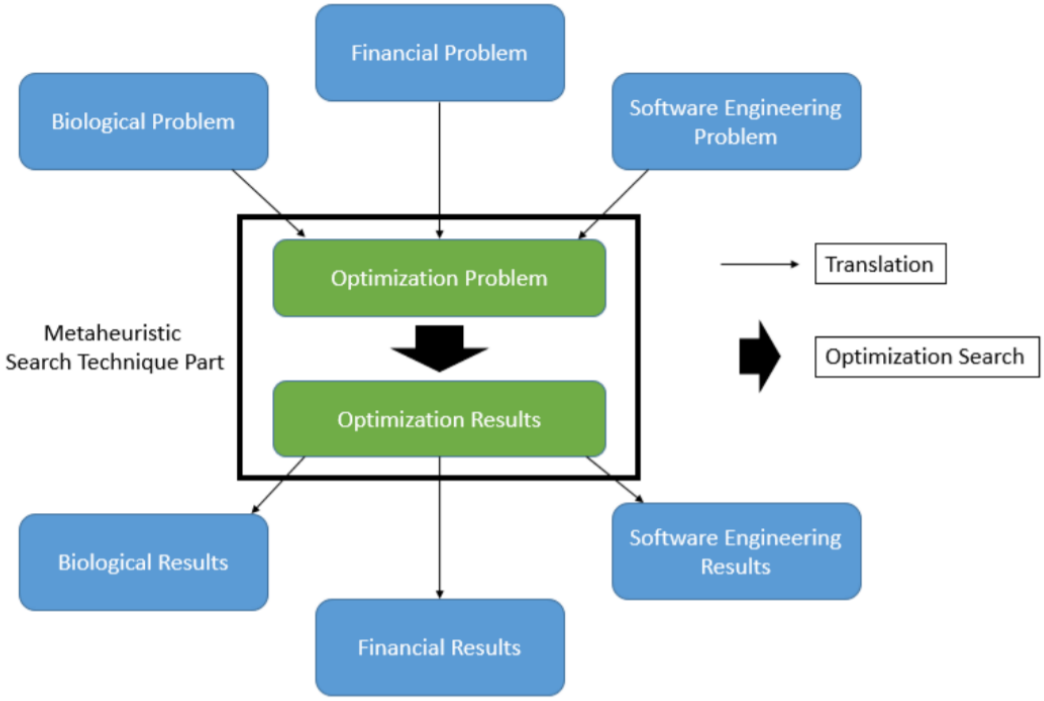


Figure 30: How metaheuristic search techniques work

versions of problems. This simplification only takes into account the significant characteristics for the optimization. Then, most of the hard work is to translate well our problem into a GA's one.

At the core of a GA is the natural selection process based on the fitness of the individuals. The latter is encoded by a *fitness function*: an individual with a higher fitness will have a higher chance to survive; our optimization problem then is a maximization problem: We aim to find the individual that has the highest fitness. Equivalently, we can formulate it as a minimization problem where the fitness function is to be minimized.

In our case, a good path is a path with a low action. Consequently, we frame it as a minimization problem and choose the action itself as a fitness function. Having searched through the chosen subsets of solutions, the GA returns the best individual i.e. the path with the lowest action after a long process of natural selection. Thus, this path is an approximation of the minimum action path.

5.7.2 MAP Landscape [LD]

The idea is to compute the landscape from the stable points and the Minimum Action Path function. The MAP basically provides a height between two points as, along a path, it increases for any ascension on the landscape and remains constant in descent. From this information, we can calculate the relative height between each stable point and find the lowest one, as in Figure 31. Indeed, the relative height (RH) of B from A is

$$\mathrm{RH}(A,B) = Action(\mathrm{MAP}(A,B)) - Action(\mathrm{MAP}(B,A)) \tag{19}$$

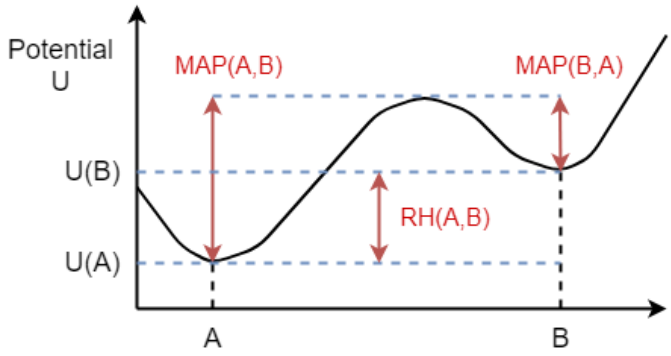


Figure 31: Computation of the relative height between A and B from MAP function in a 1D example.

For models with more than two stable points, we solve the problem by defining a matrix of relative heights

$$(\mathrm{RH}(A_i,A_j))_{i,j} = (Action(\mathrm{MAP}(A_i,A_j)) - Action(\mathrm{MAP}(A_j,A_i)))_{i,j} \tag{20}$$

Then the lowest point, A_k, is given by

$$k = \underset{i=1..N}{argmax} \sum_{j=1}^{N} \mathrm{RH}(A_i,A_j) \tag{21}$$

where N is the number of stable points. From this result, we can calculate the actual

action of the MAP, TrueAction, from a stable point A_j to any point in the space B:

$$\text{TrueAction}(A_j, B) = Action(\text{MAP}(A_j, B)) + \text{RH}(A_k, A_j) \tag{22}$$

Combining these results, we can now go further and consider the computation of the landscape. We discretize the space and compute the TrueAction from each stable point to each point of this defined space. The value of the potential U in any point of space B is then defined as:

$$\text{U}(B) \propto \underset{j=1..N}{min} \text{TrueAction}(A_j, B) \tag{23}$$

In our package, the user can implement an SDE model (and easily an ODE model from it), find the stable points through the solver, and further find the potential landscape by calling the function MAP.

```
### test of MAP_reduction_2D
nb_bin=20 #discretization of the space
Nsize=15 #number of points in each path
dim=2 #dimension of the problem
boundaries=[[0.0,3.0] [0.0,3.0]] #boundaries of the problem in each dimension
fixed_points=[[1.0 1.0], [0.0 2.0], [2.0 0.0]] #fixed points
dim1=1 #dimension of interest 1
dim2=2 #dimension of interest 2
Nb_ga= 10 #number of time the GA is repeated
Nb_gen=300 #number of generation for each GA
pop_size=100 #population size for each GA

@time U=MAP_reduction_2D_multiple_fixed_points(nb_bin,boundaries, dim1, dim2,
                                               Nb_ga,Nb_gen,pop_size,fixed_points,Nsize,dim,f,g)
span_2D=Span_2D(boundaries,nb_bin,1,2)
x=span_2D[1,:]
y=span_2D[2,:]

plotly()
surface(x,y,U)
```

Figure 32: Call of the MAP function for the 2D model.

5.8 Method election criteria [LT]

5.8.1 PFM Landscape Evaluation

Motivation

The main desirable feature of quasi-potential landscapes is their visual and intuitive information about the potential driving the main part of the system dynamics. However, this information is only approximate to a degree that varies depending mainly on the computational challenges summarized above. It then follows naturally the complementation of the generated landscape with its corresponding visual information about its quality.

The quality of the landscape constructed through the action-based method relies almost exclusively on the accuracy of the genetic algorithm, and it is easy to compare to the results of other superior optimization approaches suitable for and tested in our toy models. The quality increases as the accuracy of the optimization and the discretization of the space increase.

However, we were more focused on evaluating the PFM landscape, as we did not establish any clear quantitative relationships between the computational challenges and the theoretical consistency with the Lyapunov-stability criteria. For this reason, we attempt to assess these landscapes empirically. We provide a tool for visually distinguishing the zones of the landscapes that does not meet such conditions.

Structure of the tool

We provide a command-line function (called 'PFM_Lyap_filter+plot.2.jl') that takes as input a quasi-potential landscape with its corresponding coordinates region, calculates $\frac{\Delta U}{\Delta t}$ at every point, classifies them as greater than zero or not, and returns the landscape as a scatter plot with the two types of points in different colors. Also, we provide an additional function (that is extracted from the previous one) that returns the 3D vector field $\vec{F} =< \frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t}, \frac{\Delta U}{\Delta t} >$, since it might be useful for the user.

The analytical expression for the time derivatives of the system position is directly retrieved from the deterministic part of the model, $f(x)$. An approximation of the time derivative of the quasi-potential is calculated using equation (12).

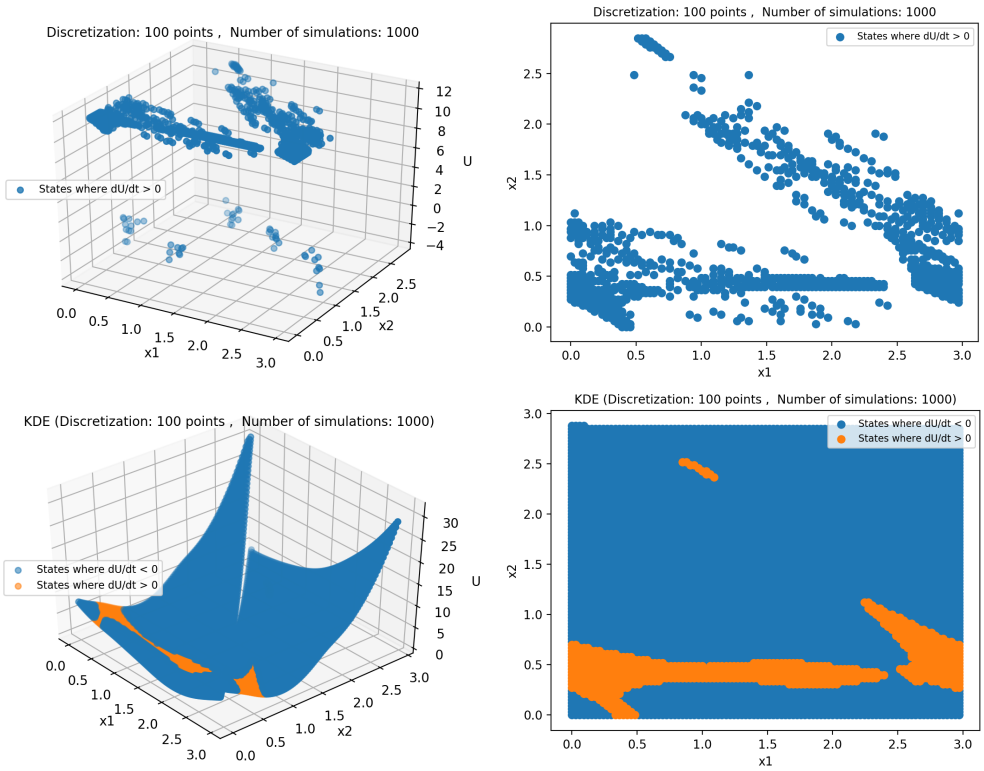


Figure 33: Upper row: for clarity purposes, we only show the subset of vectors from the landscape whose $\frac{dU}{dt} > 0$, both in 3D (in the left) and in 2D (in the right). Lower row: subsets of vectors from the landscape whose $\frac{dU}{dt} > 0$ (orange) and whose $\frac{dU}{dt} < 0$ (blue).

Limitations

∇U was calculated with a function of the linear algebra Julia tool-set (the 'diff' function) that takes the differences between consecutive points in a specified dimension, and therefore is only approximate.

The number of simulations required for obtaining a good landscape depends on the bin-size used for recording their end-positions. Intuitively, the smallest the bin-size is, the higher number of simulations are needed in order to counter the increase in variance at each bin.

Examples

We provide an example of the assessment of landscape of the 2D repressilator model (defined in the next section) obtained by the PFM, both with and without KDE. (See individual report by Luis Torada for a more detailed discussion).

6 Landscapes [MH]

6.1 2-Dimensional Model

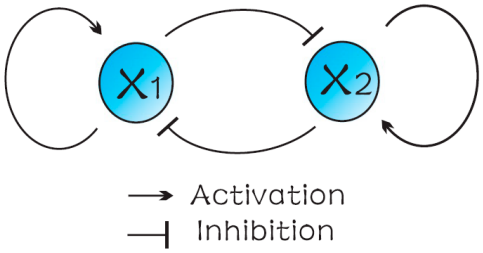


Figure 34: Illustration of a simple interaction between two genes (x_1 and x_2) self-activation and mutual inhibition [25].

Figure 34 can be described by a pair of ordinary differential equations (ODEs):

$$\frac{dx_1}{dt} = \frac{a_1 x_1^n}{S^n + x_1^n} + \frac{b_1 S^n}{S^n + x_2^n} - k_1 x_1$$

$$\frac{dx_2}{dt} = \frac{a_2 x_2^n}{S^n + x_2^n} + \frac{b_2 S^n}{S^n + x_1^n} - k_2 x_2$$

where a_i is the self activation, b_i is the basal expression, and k_i is the inactivation (degradation), for $i = 1, 2$. S parameterizes the inflection point of the sigmoidal functions which dictate the activation and inhibition, and n is the Hill coefficient determined by the steepness of S [25]. This type of interaction is found biologically in, for example, the interaction between the genes *Gata1* and *Pu1*.

As we need an SDE system for the simulations, we define the stochastic part as the square root of the absolute value of the deterministic one, which is common in chemical equations.

$$dX_1 = (\frac{a_1 X_1^n}{S^n + X_1^n} + \frac{b_1 S^n}{S^n + X_2^n} - k_1 X_1)dt + \sqrt{|\frac{a_1 X_1^n}{S^n + X_1^n} + \frac{b_1 S^n}{S^n + X_2^n} - k_1 X_1|}dW_t$$

$$dX_2 = (\frac{a_2 X_2^n}{S^n + x_2^n} + \frac{b_2 S^n}{S^n + X_1^n} - k_2 X_2)dt + \sqrt{|\frac{a_2 X_2^n}{S^n + x_2^n} + \frac{b_2 S^n}{S^n + X_1^n} - k_2 X_2|}dW_t$$

This system can be implemented in Julia by the code given in Figure 24. The

culmination of techniques described in this project yields the potential landscape for this model (with parameters as given in the implementation). This is shown in Figure 35. The basins of attraction arise in the darker, lower parts of the landscape. This particular landscape arises from the KDE method, with Gaussian kernel.

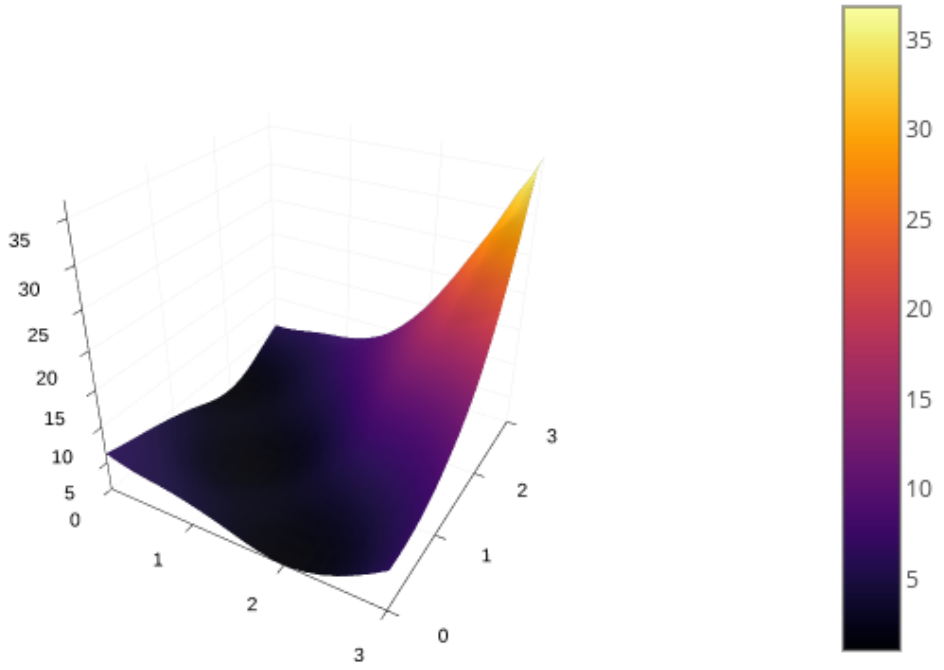


Figure 35: Waddington epigenetic landscape for the 2D system of self-activating mutually inhibiting genes.

In Figure 36 and 37, the results from the different methods and the control found in [25] are presented.

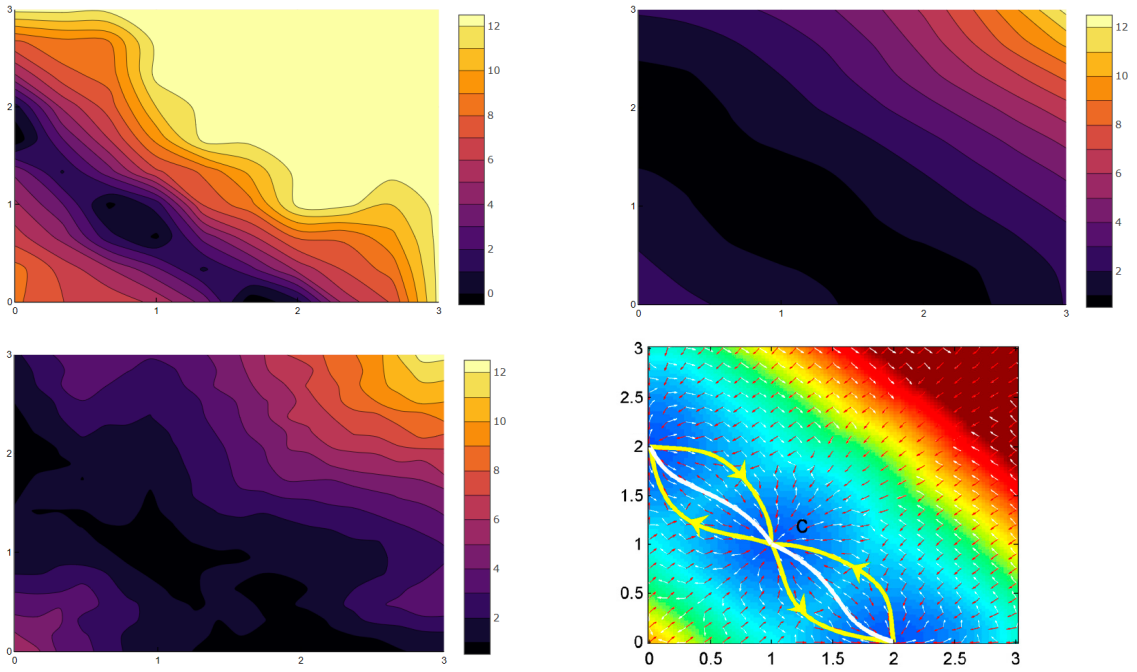


Figure 36: Colored maps of the 2D model potential computed from PFM (top-left), PFM-KDE (top-right), ABM (bottom-left) and control plot from [25] (bottom-right). X1 in X-axis and X2 in Y-axis.

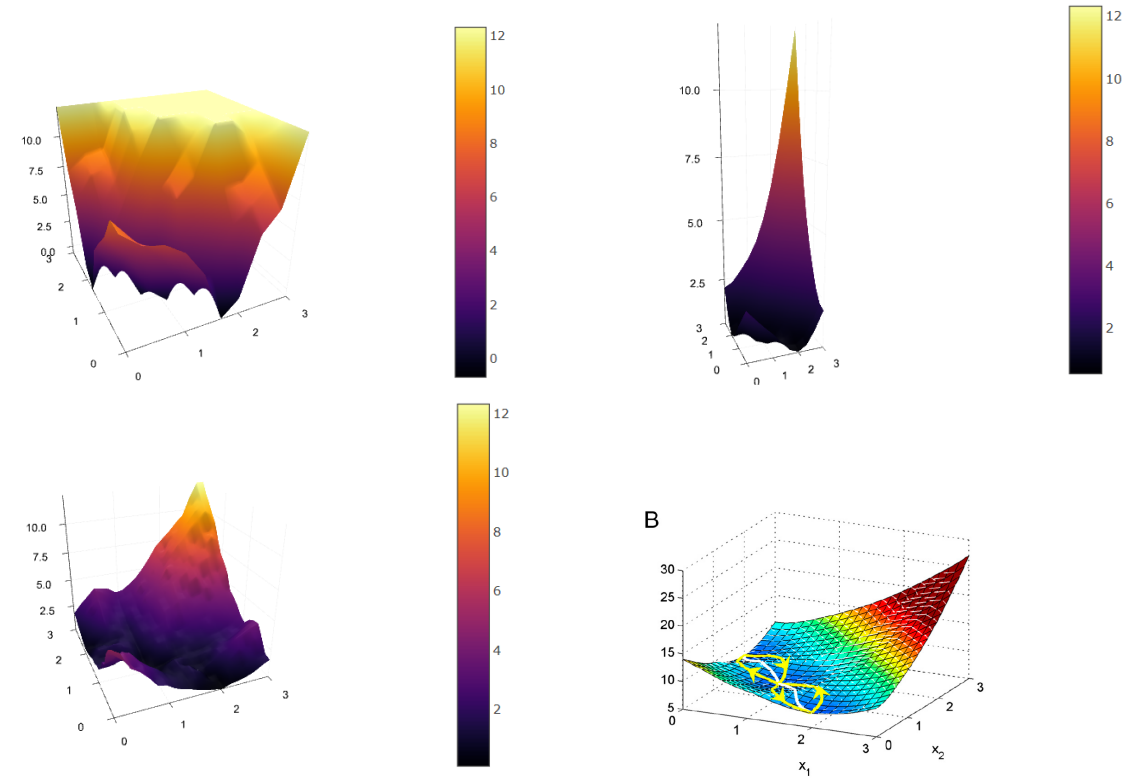


Figure 37: Potential landscape of the 2D model potential computed from PFM (top-left), PFM-KDE (top-right), ABM (bottom-left) and control plot from [25] (bottom-right).

6.2 Data

We developed functions in Julia that allow the user to view the landscape between any two genes in the dataset. The function input arguments are the data of type `DataFrame`, and optionally the two genes of type `String`. An example of how the user may specify the two genes they wish to plot is show below.

```
>julia (s,U)=two_genes(data)
Gene 1: Fgf4
Gene 2: Pou5f1

>julia (s,U)=two_genes_input(data,"Fgf4","Pou5f1")

>julia (s,U)=two_genes_skip_zeros(data,"Fgf4","Pou5f1")
```

The outputs `(s,U)` correspond to a scatter plot of the data between the selected pair of genes, and the potential landscape respectively. Observing the scatter plot is a useful option to give the user as it can give an indication of where basins of attraction may occur in the landscape (see Figure 38). The function `two_genes_skip_zeros` performs the same computation as the previous two functions, but ignores measurements in the data with a zero value. The most common form of measurement noise in this type of data is false zeros. Therefore it is useful to give the user the choice to plot the landscape neglecting any zero values. The choice of whether to ignore zero values in the data or not is left to the user. For comparison, Figures 39 and 40 show the use of the functions `two_genes` and `two_genes_skip_zeros` respectively, for the genes Fgf4 and Pou5f1.

Given the data can be partitioned based on the time point at which measurements were taken, we can in fact visualise the time evolution of the landscape between two genes. This visualisation can be given by simply observing the landscapes alongside one another (see Figure 41). Alternatively, animations can be created that

cycle through the increasing time points and corresponding epigenetic landscapes. The software ImageMagick was used to generate such animations in this project.

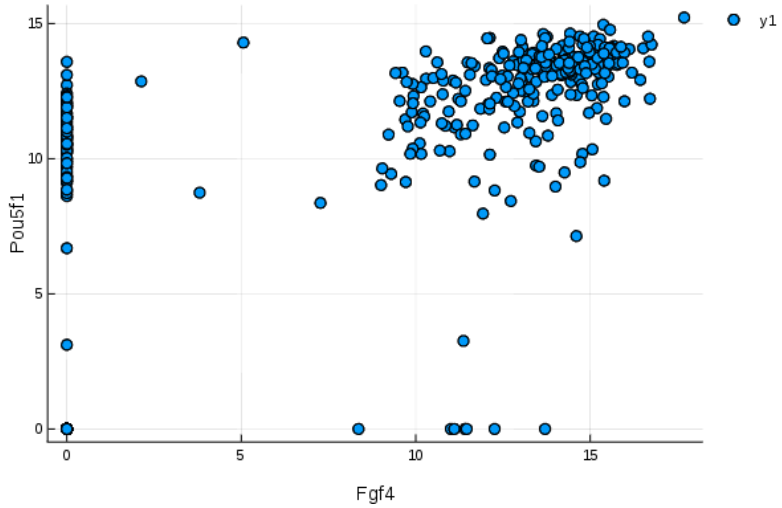


Figure 38: Scatter plot of the data between genes Fgf4 and Pou5f1, the genes in the data with the highest amount of mutual information.

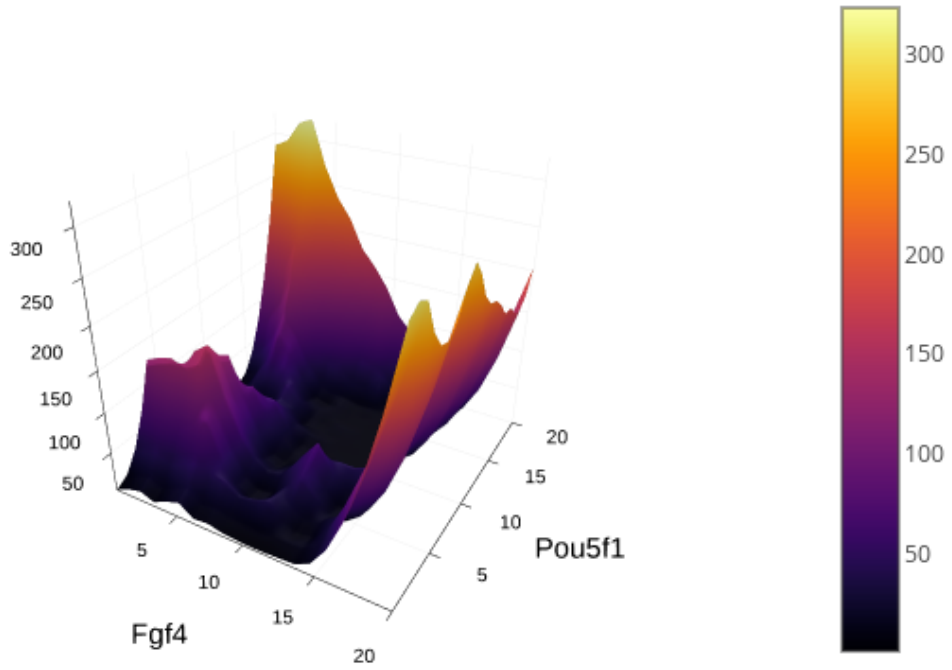


Figure 39: Waddington epigenetic landscape for the genes Fgf4 and Pou5f1, derived from the entire dataset.

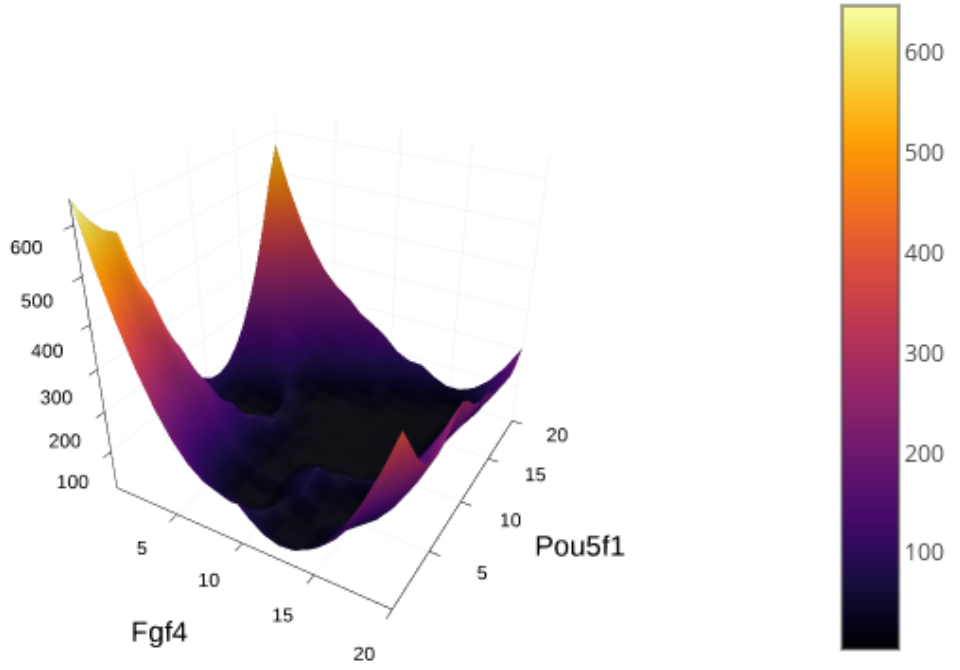


Figure 40: Waddington epigenetic landscape for the genes Fgf4 and Pou5f1, derived from only the nonzero data points.

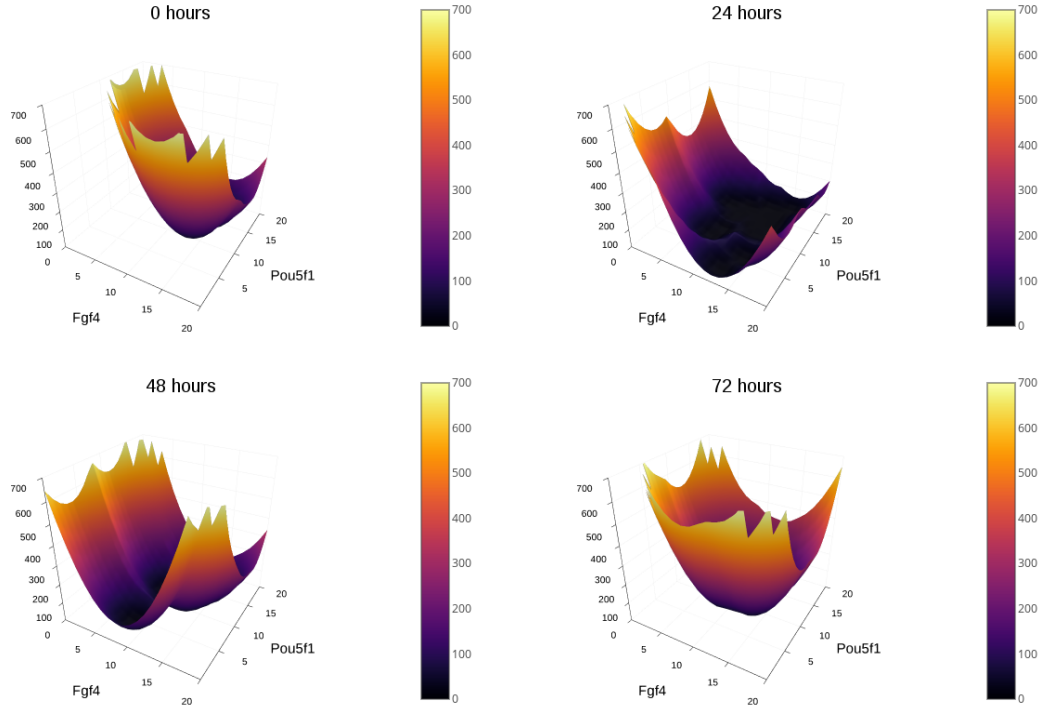


Figure 41: Time evolution of the Waddington epigenetic landscape between the genes Fgf4 and Pou5f1. It is informative to observe the basins of attraction shifting as the system develops in time.

7 Discussion and Outlook

7.1 Achievements and Wider Context [FB, MH]

No group has previously implemented the methodologies that we have outlined in this report in Julia. We have explored different approaches to generate a landscape, provided tools and methods for analysis, and provided information that may be informative to the user. In addition, we have performed an extensive review of dimensionality reduction techniques, which are widely applicable to this topic. This has resulted in a well-documented Julia package that can be downloaded, that is ready to be used and added by the open source community.

It is challenging to synthesise the work that has been done as very different approaches have been taken previously. To situate our work within the wider literature, we highlight the major state-of-the-art methods that have been developed by other groups. Extensive work on the generation of epigenetic landscapes has been performed, resulting in a number of tools for this purpose: Monocle, Topslam, Netland, Hopland, to name a few. However, none of these tools have made use of the Julia programming language.

7.2 Limitations and Future Work [LD, MH, LT, FB]

The work that we have produced has some limitations. Here is a list of areas that could be considered for future work:

- Increase the number of simulations while not making any time concession thanks to Julia GPU environment. It should be available soon.
- Find computational solution for the KDE's bandwidth which is a major factor in the quality of KDE's results.
- Look again for different dimensionality reduction techniques (including non-linear ones) as the package was not working during the project. It might be the same problem as the GPU environment so we should wait for Julia

0.7.0. The plan is to compare different methods, both linear and non-linear including manifold learning techniques on multiple data sets in order to make valid recommendations.

- In the meantime, tools in other languages, such as Python for both the manifold learning as well as GPU implementation could be used, including *CuPy* (a NumPy-like API accelerated with CUDA) and the scikit-learn library for manifold learning. For making manifold learning tools faster one we could look into *PyTorch* as an machine learning library for Python that also runs on the GPU.

- The GA is not performing well currently. As a future focus, we must find new techniques to compute the MAP and also think about a cleverer way to use the MAP over the grid (drawing areas around the fixed points so we could consider computing the MAP only from one point and not three as done currently). Work on the GA's parameters setting might be good idea as well. It could allow to reduce the time of computation while providing better results.

- Providing a method that computes the probability density function in more than 2 dimensions can be interesting. One of our PFM methods do it but the main problem is the quality of this pdf in high dimension. The added value of this method would be to allow the user to plot the a 2D landscape considering two variables while fixing the other variables values.

- Directly links to the previous paragraph, it would be interesting to provide evolving landscapes from dynamical systems. It is already done for data from single cell experiments (see section 6.2). With a pdf of more than 2 dimensions, it would be possible to plot the evolving landscape for two variables with a third one moving.

- The tool for extracting parameterized ODE functions from SBML files is limited to a certain degree of complexity in the models definition. Decreasing the

number of assumptions and testing the tool with a greater number of SBML files would increase its functionality and its robustness.

- We have not tested the stability analysis tool in more than three dimensions. Further testing and potential error corrections would make it a robust general-purpose tool for our application and for the Julia community.

7.3 Julia [MH, FB]

During the course of this project the Julia community were preparing for the upcoming release of v0.7.0. The necessary preparations for the release caused disruptions in the current release that lead to unforeseen and unavoidable hurdles in the development of our code. Updating packages lead to malfunctions and errors too deep for the capacity of our group to overcome. Most of the documentation found online was deprecated, this was especially a problem for the packages *Gadfly.jl, ManifoldLearning.jl, MultivariateStats.jl* and a lot of packages were conflicting with each other e.g. *Gadfly.jl* with almost any packages, *RDataset.jl* with most of the used packages such as *DifferentialEquations.jl*. The latter was particularly prone to conflicts. Work-arounds were necessary and will be addressed in some individual reports.

8 Concluding Remarks [MH, FB]

The overarching aim of this project was to develop and implement a computational framework in Julia that generates and explores Waddington's epigenetic landscapes. We explored numerous different approaches to generating landscapes, how to tackle the problem of DR when using single cell data, investigating different input methods and analyses that can be performed.

Developing our tools in Julia was an excellent opportunity to design and implement software for a clear scientific tool as part of an interdisciplinary group. Additionally, integrating ourselves in the Julia community and using such a new

and fast-developing programming language has been highly beneficial for our growing computational abilities; also it gives us the chance to practise another computing language besides Python.

The time allocation of this research project was only ten weeks, and the depth of understanding acquired in this short amount of time is something we are proud of. We look forward to presenting our work in due course.

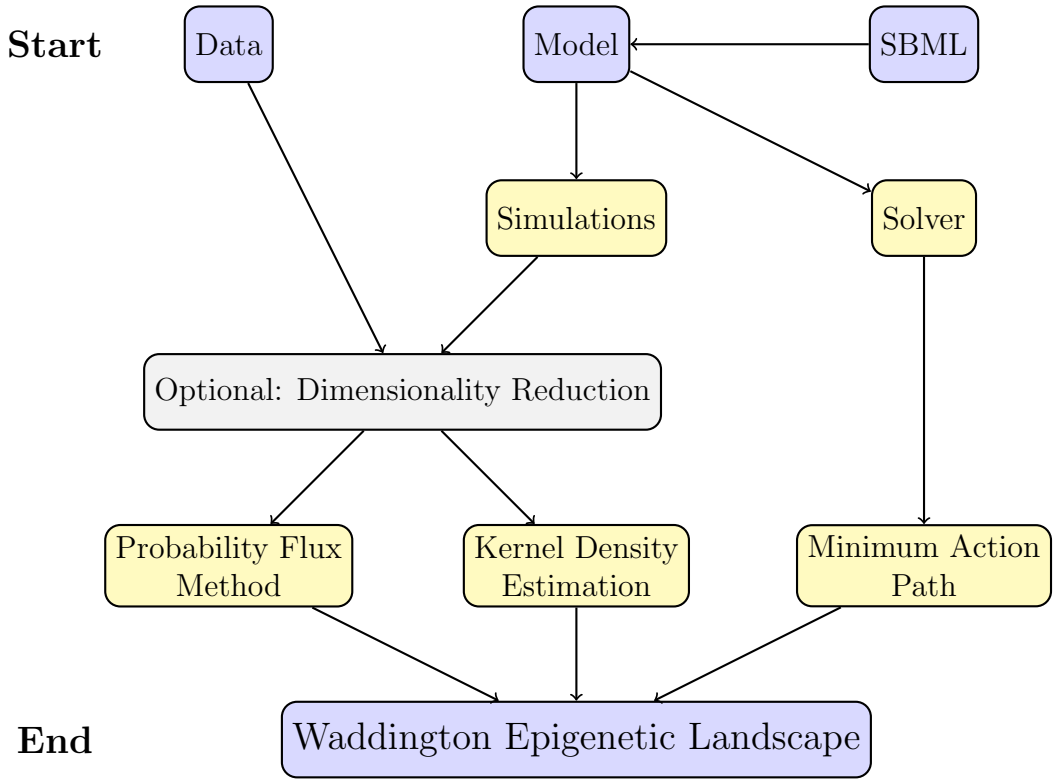


Figure 42: A summary of the progression of steps taken from three possible starting points - data, model or SBML. The three explored methods of landscape construction - PFM, KDE and MAP - are applied after an optional dimensionality reduction step.

References

[1] Bendall SC et al. *Single-cell Trajectory Detection Uncovers Progression and Regulatory Coordination in Human B Cell Development*. Cell, Volume 157, Issue 3, 24 April 2014.

[2] Bornstein, B. J., Keating, S. M., Jouraku, A., and Hucka M. (2008) *LibSBML: An API Library for SBML*. Bioinformatics, 24(6):880881.

[3] Chan TE, Stumpf MPH, Babtie A. *Gene Regulatory Network Inference from Single-Cell Data Using Multivariate Information Measures*. Cell Systems Volume 5, Issue 3, p251-267.e3. 27 September 2017.

[4] Elowitz MB *et al. A synthetic oscillatory network of transcriptional regulators*. Nature 2000 Jan; 403(6767): 335-338.

[5] Endre T. *et al. libRoadRunner: a high performance SBML simulation and analysis library*. Bioinformatics, Oct 2015; 31(20): 33153321.

[6] Freidlin MI, Wentzell AD, Szücs J. *Random perturbations of dynamical systems*. Number 260. Springer, Heidelberg [u.a], 3. ed edition. 2012.

[7] Gillespie DT. *The chemical Langevin equation*. The Journal of Chemical Physics, 113(1):297306. 2000.

[8] Guo J, Zheng J. *NetLand: a Software Tool for Quantitative Modeling and Visualization of Waddingtons Epigenetic Landscape*. Last updated: 7 Dec, 2016.

[9] Guo J, Zheng J. *HopLand: single-cell pseudotime recovery using continuous Hopfield network-based modeling of Waddington's epigenetic landscape*. Bioinformatics, 33, i102-i109. 2017.

[10] Haghverdi L, Buettner F and Theis FJ. *Diffusion maps for high-dimensional single-cell analysis of differentiation data*. Bioinformatics, Volume 31, Issue 18, 15 September 2015.

[11] Heymann, M *et al. The geometric minimum action method: A least action principle on the space of curves*. Communications on Pure and Applied Mathematics, Aug 2008, Vol.61(8), pp.1052-1117

[12] Hucka M. *et al. The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 2 Core*. sbml.org, December 2016.

[13] Hucka M. *et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models*. Bioinformatics, Oct 2002; Vol. 19 no. 4 2003, pages 524-531.

[14] Kholodenko BN *et al. Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades*. Eur. J. Biochem. Mar 2000; 267(6): 1583-1588.

[15] Liepe JC *et al. ABC-SysBio Approximate Bayesian Computation in Python with GPU support*. Bioinformatics, Jul 2010; 15;26(14):1797-9.

[16] Machn R *et al. The SBML ODE Solver Library: a native API for symbolic and fast numerical analysis of reaction networks*. Bioinformatics, Jun 2006; 1;22(11):1406-7.

[17] Moris N, Pina C, Arias AM. *Transition states and cell fate decisions in epigenetic landscapes*. Nature Reviews Genetics Volume 17 693-703. November 2016.

[18] SCUBA: "Single-cell Clustering Using Bifurcation Analysis." https://github.com/gcyuan/SCUBA. Last accessed: 27 March 2018.

[19] Setty M et al. *Wishbone identifies bifurcating developmental trajectories from single-cell data*. Nature Biotechnology 34, 637-645, 2016.

[20] Shalev-Shwartz S and Ben-David S. *Understanding machine learning. From theory to algorithms*. Cambridge University Press, 2014.

[21] Stumpf PS, Smith RCG, Lenz M, Schuppert A, Müller FJ, Babtie A, Chan TE, Stumpf MPH, Please CP, Howison SD, Arai F, MacArthur BD. *Stem Cell*

Differentiation as a Non-Markov Stochastic Process. Cell Systems Volume 5, Issue 3, p268-282.e7. 27 September 2017

[22] Trapnell C et al. *The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells*. Nature Biotechnology 32, 381-386, 2014.

[23] Waddington CH. *The Strategy of the Genes*. Routledge, 2014. First published in 1957.

[24] Walter F. *Foundations of Differentiable Manifolds and Lie Groups*. Springer, 1983.

[25] Wang J, Zhang K, Xu L, Wang E. *Quantifying the Waddington landscape and biological paths for development and differentiation*. PNAS vol. 108 no. 20 8257-8262. May 17 2011.

[26] Wasserman L. *All of statistics : a concise course in statistical inference*. Springer Texts in Statistics. 2005.

[27] Zhou JX, Aliyu MDS, Aurell E, Huang S. *Quasi-potential landscape in complex multi-stable systems*. J. R. Soc. Interface (2012) 9, 3539-3553. 29 August 2012.

[28] Zwiesele M and Lawrence ND. *Topslam: Waddington Landscape Recovery for Single Cell Experiments*. bioRxiv, https://doi.org/10.1101/057778, 8 June 2016.