1

Waddle – Waddington's epigentic landscape

Felicia Burtscher

***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. We implement the probability flux method (PFM) of constructing a landscape, additionally applying kernel density estimation (KDE). Stability analysis is conducted on dynamical systems and their corresponding landscapes. In conjunction to that second, a more accurate approach of the landscape is performed: Minimal Action Path (MAP) method. The latter one has a direct interpretation to the force that would need to be applied to cross the landscape between two chosen points. For high dimensional systems, the topic of dimensionality reduction is addressed, with corresponding tools developed support visualisation of these high-dimensional data. Reasonable results could be yielded for the 2-dimensional KDE as well as the PFM on the CPU; the PFM on the GPU posed a real challenge due to problems outlined in the group report and is still work in progress.**

I. Introduction

The following report will focus on different methods of dimensionality reduction (DR) and literature research as this was the part I have been most enaged with during the project and the part that has not given enough credit in the group report (in terms of where time and brain power was speant) as it went beyond the scope and goal of our project to simulate the epigenetic landscape. Other parts I contributed to was the Genetic Algorithm (GA) to approximate the MAP, implementing different kernels for the KDE in 2D, converting the MAP between two points into a quasi-potential (a now more elaborate final version has been developed), implementing different kernel methods for the kernel density estimation (KDE) as well as different information theory approaches to the given sample data set. However, this will not be the subject of the following report to avoid overlap with the other reports and due to the restrictive word count. For an overview we refer to the group project report, for the information theory approaches we refer to Madeleine Hall's report, for the GA as well as the KDE we refer to Lucas Ducrot's report. The final code can be found on *https://github.com/burfel/waddington-project*. The *README.md* will be kept self-explanatory and should give an overview of the main tools implemented.
Other important tasks of mine were documentation, understanding, adjusting and commenting other people's code, keeping our github repository nice and tidy and sharing links to interesting papers via a pad as literature research was a major part in the project.

Special thanks go to Michael PH Stumpf, Suhail A Islam, Rowan Brackston, Ivan Croydon Veleslavov and the whole Julia community out there.

It should be noted that in the following, I assume the reader to have read our group report. My following individual report is kept on a personal note and should reflect my main contributions to the research project and time allocated during the processes, challenges I faced and especially learnings thereof and is therefore not written in research-style.

For an introduction to the topic, we refer to the group project.

II. Literature research

Before I started with anything (except some toy models) I did some extensive literature research on the work that has already been done regarding Waddington's landscape. This was necessary as there was no specific task given, and no underlying biological question that needed to be answered. It was probably one of the major challenges of the whole project: To find a (convincing) motivation of the project and to come up with a specific goal. It is challenging to synthesise the work that has been done as very different approaches have been taken. In order to situate our work within the wider literature, I am going to highlight the major state-of-the-art methods that have been developed by other groups:

A. Wanderlust

Uses t-SNE — makes use of a non-linear method but does not consider the topography of the map when developing pseudotime orderings. The topography of the epigenetic landscape influences distances between cells on the 347 landscape, and therefore their effective relative positions to each other. Wanderlust [REFERENCE] makes use of a 345 non-linear method but does not consider the topography of the map when developing pseudotime 346 orderings. The topography of the epigenetic landscape influences distances between cells on the 347 landscape, and therefore their effective relative positions to each other.

B. Monocle

C. Topslam

uses PPCA

D. Hopland

uses BG-LVM, data-driven based on Hopfield network; emerging trend for integrating dynamical models with experimental data Want a probabilistic model of PCA that comes with a number of advantages.... TALK ABOUT BGLPVM here; too complicated/ involved approach that could not be achieved within the given time frame, but might be interesting to look into in the future –¿ MOVE TO OUTLOOK

For a more thorough review on the Hopland approach, we refer to Madeleine Hall's individual report.

E. Hadoop

??

F. Wishbone

?

G. SCUBA

?

[I WILL PROVIDE AN OVERVIEW OF THE LITERATURE REVIEW THAT WILL GO INTO THE GROUP REPORT]

As so much has been done already, and we did not want to reinvent the wheel, we defined our goal as follow: To develop a set of Julia tools and methods to simulate (via the Probability Flux Method (PFM), Kernel Density Estimation (KDE) and the Minimum-Action Path (MAP)) and analyse the given landscape including dynamical systems stability analysis; moreover, we wanted to provide the user with various methods to read in the data (including from sbml files), extract information from the given data set (statistical information, plots for visualisation) and based on this visualise specific genes or gene combinations after applying dimensionality reduction (DR). However, visualisation was not our focus as this has been tackled by last year's group already.

III. Sample single cell data set

A. Exploratory data analysis

Complementary to the information given in the group report on the data (see section [REFERENCE]) that was largely taken from [REFERENCE DATA PAPER], and the information obtained by measure-theoretic anlaysis, we provide the user with an initial exploratory data analysis: The user can obtain statistics on the whole sample set, such as mean and variance of the different genes, that can be visualised; also more detailed statistics such as percentiles by selecting individual genes can be obtained and subsequently visualised in a boxplot.

[INSERT FIGURES HERE: MEAN, VARIANCE, BOXPLOT] [INSERT TABLE HIGHEST MEAN, LOWEST MEAN, SAME FOR VARIANCE] BUT: OF ANY VALUE AT ALL??!

This can help the user to "get a feel" of the data set, discard outliers (COULD HAVE BEEN DONE HERE BUT WASN'T) and serve as a starting point for a subsequent more detailed analysis.

B. Feature selection

When we visualise the Waddington's epigenetic landscape in three dimensions, typically the x-axis and y-axis correspond to the expression level of two marker genes that represent cell states, while the z-axis represents the potential. To visualise global information, both in case of real world data or simulated data based on a model of more than two genes, DR techniques would be needed (see next chapter).

For now, we want to visualise the landscape with respect to two genes. For that purpose, we wanted to choose genes that have an "important" relationship and would result in a distinctive landscape. Additionally to computing pairwise correlations of genes, we therefore used measure theoretic approaches implemented in the *InformationMeasures.jl* package to examine relationships between genes. These included eg mutual information (MI), conditional MI, entropy and conditional entropy; the program can be found on `https://github.com/burfel/waddington-project/blob`

A summary of the results can be found in table 1 of the group report; for the plots, a discussion and further details we refer to Madeleine Hall's individual report.

IV. Dimensionality reduction (DR)

If the number of dimensions, ie genes in our case, is large, the number of possible states in this space grows exponentially large. In other words, with increasing dimensionality it becomes harder to sample from the space; even for a small model of three genes this can be a major challenge. Simulations take very long to run (as discussed in section ..... in the group report); similarly, to train machine learning (ML) algorithms on such high-dimensional data sets is very time-intensive and can cause over-fitting. This problem is often referred to as *Curse of dimensionality*; DR can be seen as a potential preprocessing step for machine learning algorithms.

Therefore, it is necessary to reduce the data to fewer dimensions as a preprocessing step for ML algorithms. Another reason to apply dimensionality reduction is 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 direction) can be visualised in our 3D world, or respectively projected onto a 2D plane.

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

In the following linear as well as non-linear, including manifold learning algorithms, are presented that have (partially) been applied to our given data set in order to tackle the challenges discussed above.

DR algorithms can be categorised in two main groups: Those that try to preserve the distance structure withi the data (usually linear methods) and those that aim to preserve local distances over global distances [REFERENCE: https://arxiv.org/pdf/1802.03426.pdf] (manifold learning methods).

Principal Component Analysis (PCA) as a linear method and kernel PCA (kPCA) as a non-linear method will be explained in greater detail as they represent the simplest and most common, yet powerful, approaches to DR. Additionally, an interesting manifold learning algorithm, Laplacian Eigenmaps (LEM), is presented and discussed in comparison to other non-linear methods.

A. Linear methods and Kernel PCA

1) 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 originial feature space and still retain most of the information (variance).
We will see that the new basis can be obained 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 [REFERENCE].

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

$$Au = \lambda u. \tag{1}$$

It can be shown with basic 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.*

Theorem 4.1: 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 correpsond 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 we yield the same results for the following three approaches:

- 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 IV-A2).

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.

If some eigenvalues have a significantly larger magnitude than others, the reduction of the data set via PCA onto a lower-dimensional subspace by dropping ”less informative” eigenpairs seems reasonable.

By computing the ”explained variance”, ie amount of information each PC (eigenvector) captures, on our data set, we see that most of the information, ... % to be precise, can be explained by the first PC. The subsequent components each bear information between ...% and ...%. Together, the first two PCs account for ...% of the information.

Also to come back to the question above, we leave that up to the user that 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 $Y = X \times P_k$.

[MAYBE INSERT PSEUDOCODE HERE]

For a more rigorous mathematical definition we refer to [REFERENCE SHALEV-SHWARTZ BOOK].

2) 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 4.1 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*.

Theorem 4.2: 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, $D^2_{i,i}$ 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 [REFERENCE]. In `generalPCA.jl`, another PCA method based on SVD is implemented.

3) Probabilistic Principal Component Analysis (PPCA): PCA is not based on a probability model. However, we can extend PCA by determining the PCs of a data set by maximum-likelihood estimation of parameters in a latent variable model as described in [REFERENCE].

We can use a Gaussian latent variable model to derive a probabilistic formulation of PCA. The PCs result from maximum-likelihood parameter estimates which can be computed by the usual eigendecomposition of the sample covariance matrix and subsequently incorporated in the model [REFERENCE THESIS ABOVE].

Alternatively, it is natural to apply an iterative, and computationally rather efficient Expectation-Maximization (EM) algorithm to estimate the principal subspace, ie effecting PCA.

Intuitively it seems appealing to formulate PCA in a probabilsitic way, as having a likelihood measures enables us to compare it to other probabilitic methods. Also, it allows statistical testing and Bayesian methods to be applied. Moreover, PPCA has some practical advantages and can extend the scope of classic PCA [REFERENCE]:

- Multiple PCA models may be combined as a probabilistic mixture and PCA projections could be obtained even when some data values are missing.
- Besides its application to DR, PPCA can be used as a general Gaussian density model. This has the advantage that maximum-likelihood estimates for the parameters associated with the covariance matrix can be efficiently computed from the PCs and could be applied to eg classification.

4) ICA: [UNFINISHED] - tries to decompose a multivariate signal into additive subcomponents; we assume that the subcomponents are (1) non-Gaussian signals and that (2) they are statistically independent from each other - often used in signal processing; eg sound is usually a signal that is composed of additive signals from several sources at each time t - special case of blind source separation; - common example application is the *cocktail party problem* where one tries to filter out an underlying speech signal, eg a person’s speech, in a noisy room.

[INSERT TABLE THAT COMPARES PCA VS ICA]

- finds independent components (ie factors, latent variables) by maximising the statistical independence of the estimated components - there are many ways to define a proxy for independence, eg minimisation of mutual information or maximation of non-gaussianity. The first one uses the Kullback-Leibler Divergence and maximum entropy; the second one is motivated by the central limit theorem PCA: ...

preprocessing steps in order to simplify/ reduce the complexity of the problem to the actual iterative algorithm - uses centering (subtract the mean to create a zero mean signal) - whitening (usually with eigenvalue decomposition using PCA or SVD); ensures that all dimensions are treated equally *a priori* before the algorithm is applied. - actual dimensionality reduction

[IMPLEMENTATION JADE ALGORITHM FOR ICA – PSEUDOCODE]

5) Kernel PCA (kPCA): [UNFINISHED]

Linear methods can often be powerful, but do not detect non-linear structures in the data. Kernel PCA tries to generalise linear methods like PCA for non-linear data. It uses the kernel trick to transform non-linear data to a feature space were samples may be linearly separable.

MOTIVATION: NOT LINEARLY SEPARABLE DATA...., to generalise PCA through kernel based techniques. does not rely on structure of the manifold on which the data may possibly reside.

KERNEL TRICK. COOL!!..: WE NEVER ACTUALLY WORK DIRECTLY IN THE FEATURE SPACE BUT ONLY VIA DOT PRODUCT...

[MAYBE PROVIDE EXAMPLE BUT ON OTHER DATA SET?]

[NOT VERY USEFUL AS LINEAR DR PERFORMS WELL ENOUGH, EG WE HAVE NO CIRCULAR DATA SET.]

6) Other approaches: Other classical linear approach that have been examined included Multidimensional Scaling (MDS), Linear Discriminant Analysis (LDA) and Factor Analysis.

B. 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 [REFERENCE]. 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. We could think of DR as trying to "unfold" this manifold (embedded in a high-dimensional space) so each data point is assigned a low dimensional representation while still 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, without using predetermined classifications [REFERENCE scikit-learn]. However, supervised versions exist as well.

One 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? The methods proposed in the following will touch upon that.

There are various methods that generate non-linear maps, ie embedding maps of the data points to a lower dimension. Most of them are self-organising or based on a neural network (see [REFERENCE HOPFIELD] that we examined and was presented in Madeleine Hall's individual report); the problem is usually formalised as a non-liner optimisation problem that can be solved by eg gradient descent. However, the latter only guarantees to return a local optimum; global optima are in general difficult to obtain efficiently. Also, we do not know whether the points actually lie on a manifold of lower dimension – it's a mere assumption. Therefore, non-linear methods that do not rely on the assumption of a low-dimensional manifold structure, such as kernel PCA might be more useful in some cases. [MAYBE IN THE DISCUSSION PART?]

In the following, we want to give an overview of the main manifold learning algorithms applicable to our data set. However, we will keep the maths to the lowest level possible.

1) t-distributed stochastic neighbour embedding (t-SNE): [UNFINISHED]

Probably the most widely used manifold learning method is t-SNE. Unlike most of the linear methods, t-SNE that was developed by Laurens van der Maaten [REFERENCE] is probabilistic.

t-SNE converts affinities of data points into probabilities.

t-SNE tries to minimise the divergence between two distributions: the pairwise similarity of the points in the higher-dimensional space (represented by) and the pairwise similarity of the points in the lower-dimensional space.

The similarity is measured based on Student's t-distribution or Cauchy distribution which looks similar to a Gaussian distribution.

To measure the divergence between these two distributions, we use the *Kullback-Leibler divergence (KLD)* of the joint probabilities in the original space and the embedded space which will be our cost function. From there, we can minimise by eg gradient descent to train our model.

CONS:

- computationally expensive: it scales quadratically with the number of samples
- stochastic and therefore, multiple restarts with different seeds can yield different embeddings. One might choose the one with the lowest divergence.
- global structure is not explicitly preserved

CON: There are some modifications of t-SNE that already have been published. A huge disadvantage of t-SNE is that it scales quadratically with the number of samples (O(N2)

) and the optimization is quite slow. These issues and more have been adressed in the following papers:

Parametric t-SNE: Learning a Parametric Embedding by Preserving Local Structure Barnes-Hut SNE: Barnes-Hut-SNE Fast optimization: Fast Optimization for t-SNE

[STANDARD APPROACH]

[EXPLAIN STEPS: pseudocode]

2) Laplacian eigenmaps (LEM): [UNFINISHED]

LEM, in scikit-learn also called *Spectral Embedding*, finds a low-dimensional representation of the data using a spectral decomposition of the graph Laplacian [REFERENCE]. The graph can be considered as a discrete approximation of the low-dimensioanl manifold in the high-dimensional space.

As in the other manifold learning methods, LEM starts with the assumption that the data lie on or around a low-dimensional manifold in a (potentially) very high-dimensional space. Usually, this submanifold is unknown except for finitely many points sampled form some probability distribution.

1. Internally, LEM builds a graph representation incorporating neighbourhood information of the data set. The data points represent the nodes of the graph and the edges between nodes and, therefore, the connectivity of the graph is determined by the closeness of neighbouring points usually using k-nearest neighbour algorithm (KNN). We might, therefore, think of

this graph as a discrete approximation of the low-dimensional manifold in the high-dimensional space. To make sure that points close to each other on the manifold are mapped close to each other in the low-dimensional space, ie local distances are preserved, we use a cost function based on the graph that is to be minimised, similar to ......

It uses the correspondence between the graph Laplacian, the Laplace Beltrami operator on the manifold, and connections to the heat equation (see below). The graph Laplacian is a discrete approximation of the Laplace operator on manifolds and has been widely used for different clustering and partion problems [REFERENCE Shi and Malik, 2000, SIMON, 1991, NG et al, 2002]. (The eigenvectors of the Laplacian matrix are equivalent to eigenfunctions of the Laplace operator. The Laplace operator, in turn defines the inner-product on the tangent space for any point in the manifold. The inner product is used to define geometric notions such as length, angle, orthogonality.)

In geometry and spectral graph theory, the connections between the Laplace Beltrami operator and the graph Laplacian have been known for long [REFERENCE Chung, 1997; Chung, Grigoryan and Yau, 1997]; however, LEM was the first DR method that exploited this relationship.

2. Using the notion of a Laplacian of the graph, we then compute a low-dimensional representation of the data set that optimally preserves local neighbourhood information in a certain sense.

The representation map generated by the algorithm can be thought of as a discrete approximation to a continuous map that naturally arises from the geometry of the manifold.

PRO: Key aspects of the algorithms are the following [REFERENCE LAPLACIAN EIGENMAPS FOR DR, 2002]: - The core algorithm is very simple: There is one eigenvalue problem to solve and a few local computations. The solution reflects the instrinsic geometric structure of the manifold. However, we do need to search for nearest neighbouring points in the high-dimensional space; since there are several efficient approximate techniques for that available, this is no major drawback. - The algorithm's justification is based on the Laplace Beltrami operator being an optimal embedding for the manifold [see ....]. While the manifold is approximated by the adjacency graph computed from the data points, the Laplace Beltrami operator is approximated by the weighted Laplacian of the adjacency graph with weights chosen appropriately. Since the Laplace Beltrami plays a key role in the heat equation, we can use the heat kernel to choose the weight decay function. Therefore, the embedding maps for the data approximate the Eigenmaps of the Laplace Beltrami operator which are maps intrinsically defined on the entire manifold.

The Laplacian of a graph is analogous to the Laplace Beltrami operator on manifolds;

- The locality preserving character of the LEM makes it relatively insensitive to outliers and noise. PRESERVE LOCAL INFORMATION IN THE EMBEDDING, ie points that map....conneted points stay as close together as possible. Another consequence of preserving local structures is that natural clusters in the data are "weighted" more and we can see a connectio to spectral clustering algorithms developed in learning and computer vision [REFERENCE LAPLACIAN EIGENMAPS FOR DR, 2002]. In this sense, DR and clustering can be regarded as two sides of the same coin and this connection has been explored in greater detail in [REFERENCE LAPLACIAN EIGENMAPS FOR DR, 2002]. This is in contrast to global methods, eg in ..LLE.... where pairwise geodesic distances between points are preserved.

However, not all data sets necessarily have meaningful clusters; in these cases other methods such as Isomap or classic PCA might be more appropriate.

Steps of algorithms:

1. Constructing the adjacency graph, eg based on ϵ-neighbourhoods or n nearest neighbours. The first is geometrically intuitive; however, it often leads to graphs with several connected components and, as usual, it may be difficult to choose the right ϵ. N nearest neighbour on the other hand is easy to choose and does not tend to lead to disconnected graphs but is less geometrically intuitive. 2. Choosing the weights for the edges, eg by using a heat kernel. 3. Compute the eigenmaps (eigenvalues and eigenvectors for the generalised eigenvector problem)

- LEM uses these connections to interpret DR algorithms in a geometric way

In M Belkhin's PhD thesis [REFERENCE] the problem of learning a function on manifold given by data points is discussed in greater detail. The space of functions on a Riemann manifold has a family of smoothness functionals and a canonical basis associated to the Laplace-Beltrami operator. It can be shown, that the Laplace-Beltrami operator can be reconstructed with certain convergence guarantees when the manifold is only given by sampled data points [REFERENCE PHD THESIS]. This is very useful, as we can then apply techniques of regularisation and Fourier analysis to functions defined on data.

The Julia implementation of the algorithm in the package *ManifoldLearning.jl* provides a computationally efficient approach to non-linear DR that has locally preserving properties.

[PLOTS PROJECTION, TRANSFORM]

3) Isometric Mapping (Isomap): Isomap can be seen as an extension of MDS. As the name suggests, it tries to find a lower-dimensional embedding which maintains geodesic distances between all points [REFERENCE]. It uses *Dijkstra's algorithm* or the *Floyd–Warshall algorithm* to compute the pair-wise distances between all other points; uses the Floyd–Warshall algorithm to compute the pair-wise distances between all other points and hence, estimate the full matrix of pair-wise geodesic distances between all of the points. The algorithm then uses MDS the reduced-dimensional positions of all points [REFERENCE].

4) Locally-linear embedding (LLE): [UNFINISHED]

LLE aims to find a lower-dimensional projection of the data which preserves distances within local neighbourhoods. We can think of it as a series of local PCAs which are globally compared to find the best non-linear embedding [REFERENCE: scikit-learn].

LLE was presented at approximately the same time as Isomap. As shown in [REFERENCE], it has several advantages over Isomap, including faster optimization when implemented to take advantage of sparse matrix algorithms, and better results with many problems. LLE also begins by finding a set of the nearest neighbors of each point. It then computes a set of weights for each point that best describes the point as a linear combination of its neighbors. Finally, it uses an eigenvector-based optimization technique to find the low-dimensional embedding of points, such that each point is still described with the same linear combination of its neighbors. LLE tends to handle non-uniform sample densities poorly because there is no fixed unit to prevent the weights from drifting as various regions differ in sample densities. LLE has no internal model.
Unlike most other methods, LLE uses the barycentric coordinates of a point based on its neighbours, which have some nice benefits compared to existing planar coordinates [REFERENCE].

However, one well-known issue with LLE is the regularisation problem. When the number of neighbours is greater than the number of input dimensions, the matrix defining each local neighbourhood is rank-deficient.

.

5) Hessian Locally-Linear Embedding (Hessian LLE): [UNFINISHED]

Hessian LLE is a variant of LLE

and as the name suggests, it adapts the weights in LLE to minimise the Hessian operator, ie it uses a hessian-based quadratic form at each neighborhood to recover the locally linear structure.

Con: Like LLE, it requires careful setting of the nearest neighbour parameter; it poorly scales with data size. Unfortunately, it has a very costly computational complexity, so it is not well-suited for heavily sampled manifolds. It has no internal model.

Pro: The main advantage of Hessian LLE is the only method designed for non-convex data sets [REFERENCE]. It tends to yield results of a much higher quality than LLE.

6) Local tangent space alignment (LTSA): Unlike LLE that tries to preserve neighbourhood distances, LTSA tries to capture the local geometry at each neighbourhood via its tangent space, and performs a global optimization to align these local tangent spaces to learn the embedding, ie the global coordinates of the data points with respect to the underlying manifold [REFERENCE].

It is based on the intuition that when a manifold is correctly unfolded, all of the tangent hyperplanes to the manifold will become aligned. The algorithm can be sketched by the following steps:

- compute the d-first principal components in each local neighborhood
- based on that, compute an approximation of the tangent space at every point
- optimise to find an embedding that aligns the tangent spaces

We present a new algorithm for manifold learning and nonlinear dimensionality reduction. Based on a set of unorganized data points sampled with noise from a parameterized manifold, the local geometry of the manifold is learned by constructing an approximation for the tangent space at each data point, and those tangent spaces are then aligned to give the global coordinates of the data points with respect to the underlying manifold. We also present an error analysis of our algorithm showing that

7) Diffusion maps and other methods: Other methods such as *Diffusion maps* [REFERENCE] and *Diffeomorphic Dimensionality Reduction*, or *Diffeomaps* or other variants of the LLE, have been development. Diffusion maps exploit the relationship between the heat diffusion and a random walk or Markov Chain; they seem to be robust to noise in the data as opposed to methods that use geodesic distance and its calculation of distance seemes to serve as a better notion of proximity than just Euclidean distance or even geodesic distance [REFERENCE].

Diffeomaps try to learn a smooth diffeomorphic mapping for the data from the high-dimensional onto the lower-dimensional space. However, these need a little bit more explanation and understanding on the mathematical side.

Also Gaussian process latent variable models (GPLVM), that were used in [REFERENCE: HOPLAND PAPER], are other interesting probabilistic DR that use Gaussian processes to find a lower-dimensional non-linear embedding of high-dimensional data. For more information, we refer to Madeleine Hall's individual report.

V. Genetic algorithm

[UNFINISHED]

other methods discussed

[REFER TO GROUP PROJECT]

[PLAYING AROUND WITH DIFFERENT STARTING POINTS, BUT MAYBE LEAVE THAT UP TO [LD] AS HE WAS IN CHARGE OF IT]

[IMPLEMENTATION DIFFERENT KERNEL METHODS, MAYBE COMPARE THEM? –¿ COULD GO IN GROUP REPORT AS WELL, FUTURE WORK]

see `KDE_reduction_2D.jl`

The GA is are a probabilistic search approach which are based on the ideas of evolution: We develop generations of solutions based on the Darwinian principle of survival of the fittest using a specific fitness function (eg an objective function in mathematical optimisation)

However, because of the probabilistic development of the solution it does not guarantee (global) optimality of the solution.

An initial population is created containing a predefined number of individuals (or solutions), each represented by a genetic string (incorporating the variable information). Each individual has an associated fitness measure, typically representing an objective value. The concept that fittest (or best) individuals in a population will produce fitter offspring is then implemented in order to reproduce the next population. Selected individuals are chosen for reproduction (or crossover) at each generation, with an appropriate mutation factor to randomly modify the genes of an individual, in order to develop the new population. The result is another set of individuals based on the original subjects leading to subsequent populations with better (min. or max.) individual fitness. Therefore, the algorithm identifies the individuals with the optimising fitness values, and those with lower fitness will naturally get discarded from the population.

[PSEUDOCODE]

VI. Extended discussion

A. DR and the process over the project

As there are packages for DR available in Julia already, I was planning on comparing and evaluating different methods of DR, linear and non-linear, on a certain data set (real data and simulated data) — as performance of different algorithms heavily depend on the nature of the data — and evaluate which ones work best. Following the *No free lunch theorem*, there is no best method; it depends on the specific data set. Eg LDA would make more sense than PCA if you have a linear classification task; however, empirical studies have showed that it is not always the case [REFERENCE: https://github.com/rasbt/python-machine-learning-book/blob/master/faq/dimensionality-reduction.md]. Although kernel PCA can separate concentric circles, it fails to unfold the Swiss Roll, for example; here, locally linear embedding (LLE) would be more appropriate [REFERENCE: https://github.com/rasbt/python-machine-learning-book/blob/master/faq/dimensionality-reduction.md].
Juan M. Banda] It is reasonable to say that DR should be selected on the following criteria based on this priority [REFERENCE]:

1) Their popularity in the literature
2) The availability of a mapping function or method to map new unseen data points into the new dimensional space
3) The particular properties between the data and the type of distances between the data points (eg Euclidean vs geodesic)
4) Computational cost

However, as hardly any methods in *MultivariateStats.jl* and *ManifoldLearning.jl* were actually running, even with the help of the Julia community, my initial turned out to be over-ambitious due to the major challenges in Julia explained in [REFERENCE]. They methods have been partially flagged for fixing by the time of writing this report. Therefore, we were limited to few methods such as PCA, PPCA, PCA and LEM and LTSA to test our given sample data set. However for the latter two, the returned objects no user recipe functions including plotting have been implemented and are therefore of no practical use. Another PCA based on SVD was implemented.

Two implementations of t-SNE are available in Julia [REFERENCE, REFERENCE]; none of the packages have been added to the *ManifoldLearning.jl* package since the latest Julia fails to build them though. Also, they depended on the package *Gadfly* that was unstable or *RDatasets* that was not compatible with other packages we were using at the time respectively. Also, a documentation of how to call the function needs to be added.

On the other hand, in Python all the major manifold learning methods are implemented (see http://scikit-learn.org/stable/modules/manifold.html) and worked smoothly; also the package *ScikitLearn.jl* had major flaws and even though I often did, any Python-package in Julia was recommended not to be used by some members of the community also due to instability.

As there was a tradeoff for me between working on DR and contributing to our actual project. For the first one, I could have reimplemented already existing, but broken, methods to maybe gain a better understanding for the algorithms or implemented other methods, eg the JADE algorithm for ICA or a rather involved manifold learning algorithm, such as an optimised version t-SNE. The latter could be another project, certainly beyond the scope of our project. However, I didn't see much point in trying to "reinvent the wheel", and the risk that it might not work for future Julia versions if it depends on other packages was not motivating for me. Also, we had not enough data available to test the algorithm on that would have justified an implementation; sample data sets from *RDataset* were difficult as the package was not compatible with *DiffentialEquations.jl* and all in all, it was of little use for our actual project. I therefore based my understanding of the algorithms, the comparison and their benefits and drawbacks on literature as well as the implementation in Python using *scikit-learn* [REFERENCE]. For the second one, I was working on the other parts outlined in the preview which went smoothly, the largely Julia packages were reliable and the code can be found in the repository.

Facing these major obstacles, DR (in Julia and also the theory, especially manifold learning) is an entirely different project; DR itself is a whole research field. For comments regarding Julia, I refer to the appendix.

B. Regarding DR on the simulated data set

Also, the point of time at which DR should be applied has been discussed: As we saw no way to apply DR before the actual simulation of trajectories and the estimation of landscape thereof, we could not save any computational power here. Therefore, we would treat the simulation similar to a real data set in a lower-dimensional space and motivate DR by visualisation purposes only.

Another challenge was the biological interpretability. In the case of PCA, the PCs would be interpreted as linear combinations of genes. Due to the way PCA is computed these are genes that correlate positively and could eg be regulated by the same TF or be found in the same pathway. One could

try to further investigate that experimentally.
Another idea would be to compare the centroids of the clusters detected by a standard PCA with the basins of attraction or fixed points from the stability analysis and wether they correspond.

1) Genetic Algorithm: To us the GA seemed to be a pretty straight forward approach and a good alternative to traditional optimisation methods; especially for non-linear problem and unconstrained (or largely feasible) problems. Also, compared to non-linear programming models, less variables and constraints are necessary as the variables largely depend on each other. We also chose GA because of its flexibility and the type of models that can be considered. A comparison between the non-linear programming problem and its description as a GA can be found in [REFERENCE].For a discussion of the results and problems encountered, we refer to the group report and Lucas Ducrot's individual report.

2) Different kernels in Kernel Density Estimation (KDE): Other kernels implemented for the KDE in 2D including including the linear kernel, cosine kernel, tophat kernel and epanechikov kernel taken from [REFERENCE] could be evaluated and compared.

VII. (Personal) challenges and comments

A. Motivation

As already outlined, one of the main challenges for me was the fact that we were lacking a clear motivation for the project; there was no real (biological) question we were trying to be asnwered. The more literature research we did and encounters solutions like *Netland*, *Hopland* and *topslam*, the less motivating and harder it got to come up with something "novel". This might sound similar to the development many people undergo during their PhD and is probably a major part of research; for me, it was unlike previous projects and I took it as part of the learning process. However, we can claim that we are the first group to produce something in Julia and we finally did achieve to accomodate our work and suitate it in the wider literature.

B. Julia

Julia and the contribution to the Julia community, to produce something that would actually be used, was one main personal motivation for the project. Unfortunately, Julia turned out to be a major drawback to our work, and to my part especially.

1) High volatility: Julia is still a teenager and therefore unstable; version 1 isn't out yet. The volatility, especially in light of the probably soon to be published version 0.7 caused methods in packages to run at some points of the project (see VI-A). However, this is only a snapshot of the current situation; people keep chaging, moving and renaming methods which makes it very likely that our code needs fixing at some later point in time as well. Attention to bug fixing is Also the Julia community, even though very responsive especially on *https://discourse.julialang.org/* including developers, refer to the instability of the packages themselves and could therefore be of limited help. This meant, I had to redefine my initial goal of comparing differnent DR methods on the data set. In another programming language such as Python this would have been not the case, as eg [REFERENCE PACKAGE SCIKITLEARN] contains ready-to-be-used stable functions of most of the major DR algorithms; however, using these would have been contrary to our actual goal to develop something in Julia.

2) Conflicts between packages: Conflicts between packages like ...... to only name a few was probably the biggest time-consuming aspect; packages needed to be checked out, added, pulled from the master, cleared from the cache over and over again; workarounds needed to be found. However, here the julia community could sometimes help out. Additionally, precompiling packages was a major time-consuming aspect. All in all, Julia's good performance comes at a very high price.

3) Documentation: As the documentation of specific packages including *MultivariateStats.jl* and *ManifoldLearning.jl* are very sparse and to large extent deprecated, it was a challenging to not only get things running but also to understand the implementation including accessible properties of the methods.

VIII. Future work

A. Run DR methods on multiple data sets

Still an interesting task to do is to run different DR methods explained here, eg by using the scikit-learn methods in Python, on a large pool of data sets; for other methods available in Python that could have been useful for other parts of the project we refer to A. Provided the data is available, the easiest thing to do this would be by using the scikit-learn methods in Python. This way, we can get a feel for when or on what kind of data some methods work best, maybe identify emerging patters, and develop an understanding that is hard to do by just reading literature and understanding the theory. One can then come up with improvement, as there is always room for improvement, test it on sample data sets and work out the maths behind it. A good review where the different DR methods are summarised would be useful, too. Which methods to select is often not easy and highly depends on the data under study.
As mentioned before, feature selection is another motivation for comparing different DR methods.
Also, comparing the running time of different methods and cross-check it with results found in literature would be worth doing.

As single cell data are often very noisy which might affect DR techniques, it might be recommendable to preprocess data using some single-cell analysis technique, eg *PAGODA* [REFERENCE].

B. Manifold learning

Comparing different manifold learning algorithms seems especially interesting as it is a vibrant research field.

1) MLLE: The *Modified Locally-Linear Embedding (MLLE)* algorithm has shown to be another improvement of LLE [REFERENCE] as it addresses the regularisation problem which leads to distortions in LLE maps outlined in the theory part. For that, it uses multiple weight vectors in each neighbourhood. This method would be interesting as it produces robust projections similar to Hessian LLE, but without the significant additional computational cost [REFERENCE scikit-learn]. Apart from the many manifold learning method that are (currently) not working in Julia, MLLE does not yet exist in Julia and might be a suggestion to implement in the future.

2) Ricci Flow when different curvature: We have seen that the standard Manifold Learning algorithms bear the assumption that the local neighbourhood of a point in the high-dimensional space is roughly equal to the tangent space at that point. Even if the neighbourhoods of points on the manifold have very different (Gaussian) curvature, they would still be treated equally and projected down to the lower-dimensional space in the same way. However, as this often is the case, standard DR are poorly neighbourhood preserving. [REFERENCE] suggests a solution to this: We can perform an operation on the higher-dimensional space using Ricci flow before applying manifold DR. With Ricci Flow we transform each local neighbourhood of the higher-dimensional manifold to a patch with constant curvature. Then, the higher-dimensional manifold as a whole is transformed to a subset of a sphere with constant positive curvature. They could show that this method outperforms other manifold learning algorithms with better neighbourhood preserving rate.

3) Scalability: UMAP: During my literature research, I noticed that manifold learning isn't widely used in papers but people seem to still rely on standard PCA and other linear DR methods. One reason for that might be that manifold learning techniques depend on the shape of the data and do not scale well [REFERENCE]. Also, the maths can be deterring sometimes. Manifold learning in scikit-learn are good enough and there is even a scalable CPU-bound manifold learning package available in Python (*Megaman*). A GPU implementation might come out soon.
A very promising and novel manifold learning technique I found recently is the Uniform Manifold Approximation and Projection (UMAP) on [https://arxiv.org/pdf/1802.03426.pdf]. Mathematically more involved as it draws from techniques in algebraic topology and Riemannian geometry, it scales incredibly well. It is superior to the wildly-used t-SNE in terms of visualisation quality, preserves more of the global structure and runs faster; referring to [https://arxiv.org/pdf/1802.03426.pdf] its running time is indeed astonishing. It is widely applicable as it makes no restrictions regarding embedding dimension. If it could be implemented to run on a GPU even, the running time would massively increase even more. I believe that this projects holds great potential and could be applied to a any type of data set, including biological data.

C. Genetic Algorithm

Other evolutionary algorithms, simulated annealing or divide and conquer techniques could be worth looking into. In the end, there is a no free launch theorem. Also for other metaheuristics, it is the design of that algorithm that determine their performance. And even though the results were not optimal, the GA suited the purpose well.

D. (

Further work on Julia

If one wants to contribute to the Julia community, one could update outdated documentations and make them more user-friendly; also many methods need bug fixing, see challenges explained in VII.

Apart from MLLE, a rather low hanging fruit would be to implement another ICA method based on the JADE algorithm in Julia or fix the existing one (which fails to converge on our data set) [REFERENCE]; this did not seem any relevant for our project.

IX. Personal motivation and conclusion

1) Why Dimensionality Reduction?: The high-dimensionality of the problem, especially in the real data set, was an initial motivation to apply dimensionality reduction to save computational power but especially in order to visualise data. My interest in applying DR in the project stemmed the fact that I have hardly worked with it before this project but I was keen on diving deeper into the research in this field. As I had similar projects on simulations in the past, I wasn't exactly excited about the simulation (PFM) part and the effort to get it running on a GPU in CUDA. My C skills aren't that great and I believe things could have been implemented far easier and faster if we used CuPy [REFERENCE] and our project was based on Python; in my opinion, initialising kernels in CUDA is a lot of work and I would say not state-of-the-art anymore and there are more convenient tools available.

2) Why so much literature research?: I was predominantly looking for an interesting research question that we could try to answer or maths that got me "hooked". Also, I wanted to do read the relevant literature (and ask communities) that explain these methods and their advantages and/or disadvantages to first gain an understanding of which methods are appropriate. Also, especially some manifold learning algorithms are rather involved (mathematically).

3) Personal conclusion: I saw this project and the great part of literature research to it as a learning project. I learned how to use and tried to understand Julia; however, I wouldn't use it again for a bigger project where the main purpose is not to contribute to the community as for this project, the programming language was a great limitation. The project's code is solely written Julia (except the C parts in CUDA) and we developed something new regarding Waddington's landscape. I encountered an exciting research field (DR), some interesting papers and ideas for future projects (see [REFERENCE]). For a general conclusion regarding the project, I refer to the group report.

APPENDIX

A. *Suggestions regarding Julia*

Having mentioned all the challenges with Julia, I would like to stress that I am not trying to downplay Julia.

Julia has potential and is an exciting, intuitive and promising language to contribute to and I feel our work can be seen as an investment. I like that it's open-source and all available on GitHub. Julia can even be fun to code in (once all necessary packages have been installed and you have learned how to decipher her error messages). Also, for developing single packages/ functionalities it is suitable. However, for a rather big research project that depends on multiple (unstable) packages, I question its practicability. Clearly, using Python for the project would have made life easier; also, only considering the time constraint, I would have been more reasonable to stick with Python. The community is large, documentation great and it is state of the art. Having said that, I will certainly keep monitoring it as it is further developed and improved.

I can see why people get excited Julia. It is an easy language, has good support for functional programming and its raw speed is close to those of a compiled language like C convincingly presented in the cross language validation on *http://julialang.org/*. However, I was asking myself how the se Python benchmarks were written – very likely in native Python and not in an optimised version. It might be more meaningful to compare the performance based on tasks and once experts in the specific languages have produced the best code they can to perform these tasks. Indeed, as shown in [REFERENCE] it is possible to achieve the same benchmarks when one uses Python better and its tools. Also regarding running time, I think it is less of a question of whether to use Python, C or Julia but the major difference can be achieved by GPU instead of CPU. Still, there are plenty of tools available to optimise Python including *numpy* based on C or the C extension *Cython*, *Numba* that contains high performance functions written directly in Python and also *CuPy* (a NumPy-like API accelerated with CUDA) which I would have preferred for the simulation. The latter might also be handy to run some computations, eg DR methods on different data sets, and compare them to get a feel and start to understand rather than relying on one's understanding from reading literature. One might also try to use Numpy/Scipy with Intel MKL and Intel Compilers to optimise Python code; but often it's lready enough to write critical parts in Cython.

PyTorch as an machine learning library for Python that also runs on the GPU might have been useful to get manifold learning algorithms in *scikitlearn* onto a GPU. PyTorch already optimises standard optimisation algorithms.

As things move very fast in this field that it is almost hard to keep up, and given the enormous brain power including money of companies that are behind tools like *PyTorch* and also *TensorFlow*, the numerus libraries that are available in Python, it is very unlikely that a few Julia developers will outcompete the developments currently happening in Python.

In the end, the optimisation achieved by Julia does not seem very relevant to me; however, it can be a language fun to play around with.

B. ***On a personal note – reflection and learnings***

The following is part is a reflection on learnings that will be useful for the projects to follow and especially other group or open-source projects in the future.

Working in a group can often be challenging but also a great opportunity to develop soft skills useful elsewhere in life. Over the course of the project, I could develop project management skills and I found it great to work in a diverse team, with people from different disciplines. We had to establish means of communication (between group members as well as group and supervisor). Since we were locally close to each other most of the time, a pad (*http://pad.spline.de/*), emails and a messaging app were sufficient for exchanging ideas including sharing links to interesting papers.

C. *My top 6 learnings:*

The plan was "to look before we dived deep". We wanted to get a feel of what there is out there, what and whether our ideas have been previously approached. This can sometimes be overwhelming, as there are not only many papers but also software repository such as *GitHub* or *SourceForge* available. As often engaging with the network of researchers in the field can be invaluable, we also tried to contact authors when we found code that was not running (eg matlab code in [REFERENCE Hopfield paper].

1) Learning #0: Get all the set-up done quickly.: Install or ask to have all software installed right from the beginning; this saves emails and time. Also make sure that the computer is sufficiently fast and the graphics okay or switch to a laptop. Make sure you can access the computer from remote, so you can work independently.

2) Learning #1: Divide tasks and define interfaces clearly.: Even though we could not clearly define functionalities required and interfaces (input/output) between certain components of the work from the beginning, we did manage to split work to explore different parts of the project early-on. However, we carried our "exploratory phase" (toy models, literature research etc) until very far into the project, kept brainstorming ideas and adjusting (individual) goals as we went along; this might have been largely due to the fact that we could not start off with a clear-defined goal and our work wasn't based on one single paper.
Regular meetings to discuss progress and inform other group members about one's work and keep everyone in the loop was proven to be useful.

3) Learning #2: Always pull and push.: We chose *GitHub* as a version control repository to share, backup and edit each other's code as well as overleaf to work on the report; our final code can be found on *https://github.com/burfel/waddington-project*. One take-away from working with GitHub, especially when working on one single branch, is: *Always pull first and push regularly!* This cannot be repeated often enough. This spares time-consuming merging phases and avoid double-work which often happened as we were working on the same code at the same time. I promote an open approach of sharing source code as I hold this to be important for an sustainable future in science; too often results cannot be reproduces when the code isn't reviewed or even shared.

4) Learning #3: Comment code.: Commenting code is especially necessary when working together on code and for documentation reasons for users or developers to build upon your code but also to make it understandable to you and "why" you implemented it that way — as one wise man once said to me: "Each line of comment is a conversation between you and your future self". Also using meaningful variable names and formatting the code consistently can have great benefits. Understanding other people's (uncommented) code often takes more time than starting from scratch — a major problem in industry that has been addressed by businesses like [CODEPLOT ETC]; however, comprehending commented and readable code can be enriching and help develop empathy for people. In the end, writing code is like writing poetry. I had to learn this lesson in the past the hard way and was trying to get my team members stick to that rule as well. This prevents time drains and errors.

5) Learning #4: Do not get lost in literature research, keep track of your readings.: This might especially be a personal learning. Constant alerts by *Google scholar* and *pubmed* were constantly throwing me back into reading mode, even during the implementation part, but most of the time not relevant. It is easy to get dragged away and find things you might be more interested in working on (eg topological data analysis tools are interesting but only remotely relevant for our 12-week project), especially if you lack a clear-defined goal. Literature research often can be overwhelming. Therefore, it is good to find a way how to best keep track of the readings and resources and tools used for literature research; otherwise, it can make the write-up especially painful.

6) Learning #5: Keep track of your achievements/ problems/ ideas.: Also, to reflect weekly on the work like writing down your weekly achievements and problems and ideas and steps for the following week, seems like a motivating and useful thing to do. This might help you to refocus and prevent to go overboard with the reading (see learning #5) and fits my big-picture thinking. Also, it might prevent you from jumping between tasks too much. Setting milestones for oneself or the group might be useful too.

References

[1] Waddington CH. *The Strategy of the Genes*. Routledge, 2014. First published in 1957.
[2] 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
[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] SEE PAPERS ON PAD ADN GITHUB.