1

Waddle – Waddington's epigentic landscape

Individual report by 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. 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 great 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, discussion and suggestions regarding dimensionality reduction (DR) and literature research regarding this as this was the part I was 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 Kernel density estimation (KDE) in 2D, converting the MAP between two points into a quasi-potential (a now more elaborate final version has been developed), as well as different information theory approaches to the given sample data set. However, due to the very restrictive word count and to avoid overlap with the other reports this will not be the subject of the following report. 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.
Besides literature research on and the application of DR, important tasks of mine were documentation, adjusting our code, keeping our github repository nice and tidy and sharing interesting papers via a pad as literature research was a major part in the project.

It should be noted that in the following, I assume the reader to have read our group report. My following individual report is not technical, it is kept on a personal note and should indicate my main contributions and time spent during the research project; it reflects challenges I faced and especially learnings thereof and is therefore not a formal report.

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

For an introduction to the Waddington's landscape and the scope of the project, we refer to the group project.

II. Literature research

Before starting with anything (except for 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, we highlighted the major state-of-the-art methods that have been developed by other groups in the group report. Here, I will point out the approach these methods took regarding dimensionality reduction (DR) which this report will be focusing on.

A. Wanderlust

Wanderlust [28] uses t-SNE. A non-linear method seems reasonable to apply; however, it does not consider the topography of the map when developing pseudotime orderings [27].

B. Monocle

Monocle [33] uses Independent Component Analysis (ICA). This method, however, assumes a linear map, which does not seem like a realistic assumption.

C. Topslam

Topslam [29] uses the probabilistic Bayesian Gaussian Process Latent Variable Models (BGLVM). It can be seen as an extension of PPCA – a probabilistic model of PCA that comes with a number of advantages.

D. Hopland

Hopland [27] also uses Bayesian Gaussian Process Latent Variable Models (BGLVM), it is data-driven and based on Hopfield network. We investigated BGLVM a bit further but found it to be beyond the scope of this project. For a more thorough review on the Hopland approach, we refer to Madeleine Hall's individual report.

Regarding current DR methods for single-cell data, a thorough overview can be found in [43]. Since 2016, however, other methods have been developed. The method *ZIFA*, based on the FA framework, addresses the problem of frequent dropout events in single-cell data, which leads to zero inflated data [7]. It does this by explicitly modeling dropout characteristics. However, genes with high degree of zero-inflation are problematic; equally, small gene sets should be avoided. In recent years we have seen a big development in neural networks or latent variable models for single-cell data, including *scvis* [12], which has some nice properties: It preserves both local and global neighbour structures in the data, it is robust even on small data sets and it is scalable as its probabilistic parameter mapping function learns to ad points to an existing embedding – this is a major advantage in current manifold-learning techniques; *scVI* performs equally well and beats the current state-of-the-art non-linear method t-SNE in this respect. Due to their computational complexity, almost always linear methods are applied, or at least as a preliminary step. These include PCA, Probabilistic PCA (PPCA), Factor Analysis (FA) or Independent Component Analysis (ICA) with PCA being the most popular one.

III. Sample single cell data set

A. Exploratory data analysis

Complementary to the information given in the group report on the data (see section 5.2) that was largely taken from [2], and the information obtained by measure-theoretic anlaysis, 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.
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 in various manners, see proof-of-concept examples in the group report.

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 mean to visualise the landscape with respect to two genes. For that purpose, we choose genes that have an "important" relationship and would result in a distinctive landscape. Additionally to computing pairwise correlations of genes, we therefore used information-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,

These methods based on information theory can be regarded as a generalisation of the ones based on statistics: They can capture non-linear relationships between variables and can handle interval and categorical data at the same time.

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.

For more on feature selection, we refer to [18], [19], [20].

IV. Dimensionality reduction (DR)

Complementary to our motivation to apply DR outlined in the group report, DR is often used as a pre-processing step for machine learning (ML) algorithms as training on high-dimensional data sets is very time-intensive and can cause overfitting. However, no matter how universal the application of DR is, the common goal is usually the same: To reduce the complexity of the data set to a more understandable representation without losing (much) information.

DR algorithms can be categorised in two main groups: Those that try to preserve the distance structure within the data (usually linear methods) and those that aim to preserve local distances over global distances (most of the manifold learning methods) [14].

Principal Component Analysis (PCA) as a linear method has been explained in greater detail as it represents the simplest and most common, yet powerful, approach to DR. A proof-of-concept example was presented in the group report. Methods in Julia that have been applied include KPCA, Probabilistic PCA (PPCA) and Multidimensional Scaling (MDS). However, no great insights could be gained and PCA seems to be a good method on the specific data set. To effectively compare the different methods more data would be needed.

For an overview, we categorise the main DR methods available in Julia in the diagram found in the appendix. As no good overview of the plethora of DR methods available could be found we created one on the lowest level of maths for which we refer to *https://github.com/burfel/waddington-project/tree/master/papers/dimRed*.

V. Genetic algorithm (GA) for the Minimum Action Path (MAP)

To find the MAP we designed a GA, a metaheuristic 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). Intuitively we defined the action potential as our fitting function that was to be minimised.

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.

However, as hardly any methods in *MultivariateStats.jl* and *ManifoldLearning.jl* were actually running (eg only two of the six manifold learning techniques), 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 **??**. 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, KPCA 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. I implemented another PCA based on SVD.

An implementation of the popular manifold learning algorithm t-SNE is available in Julia; it hasn't been added to the *ManifoldLearning.jl* package yet since the latest Julia fails to build them. Also, they depended on the package *Gadfly.jl* that was unstable or *RDatasets.jl* that was not compatible with other packages we were using at the time respectively.

On the other hand, in Python all the major manifold learning methods are implemented 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.

It should be mentioned that 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 was 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. 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*.
For the second one, I was working on the other parts outlined in the introduction which went smoothly, the Julia packages were largely 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 an exciting research field. For comments regarding Julia, I refer to the appendix.

B. When to use which DR method?

Following the *No free lunch theorem*, there is no best method; it depends on the specific data set. Eg Linear Discriminant Analysis (LDA) would make more sense than PCA if you have a linear classification task; however, empirical studies have shown that it is not always the case [36] Although KPCA can separate concentric circles, it fails to unfold the Swiss Roll, for example; here, locally linear embedding (LLE) would be more appropriate [36].
As we have seen, there are various methods that generate non-linear maps, ie embedding maps of the data points to a lower dimension. Others are self-organising or based on a neural network (see [27] 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 KPCA might be more useful in some cases.
It is reasonable to say that the right DR method should be selected on the following criteria based on this priority order [37]:

1) Its 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, see figure 1)
4) Computational cost

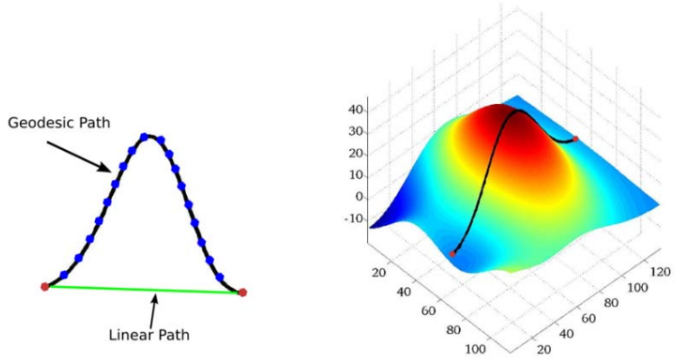

Figure 1: Euclidean vs geodesic distance. The geodesic distance between two points is the length of the path belonging to a given manifold that joins the two points whereas the Euclidean distance is the length of the linear path between the two points [38]

Regarding popularity in the literature, by far the most popular techniques are still classic linear techniques such as PCA or ICA – probably due to its simplicity and intuitiveness. However, due to the non-linear nature of experimental data in many fields, we identified a trend towards applying manifold learning techniques. This might be due to the rise of non-linear experimental data in many fields and its capability of keeping the local structure of the data.

Likewise, new methods in this fields have been developed in the last two decades.

C. Regarding DR also on the simulated data set

As a group we discussed at which point in time DR should or could be applied: 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. For where in the project of DR feeds into we refer to the workflow diagram in the group project.

Another challenge was the biological interpretability that has been found to be a problem elsewhere as well [13]. In the case of PCA, the PCs could be interpreted as linear combinations of genes as we have seen. As PCA tries to find uncorrelated variables, we could hypothesis that if there could be gene pairs identified that contribute to (always) the same PCs, they are significantly not uncorrelated, ie correlated and somehow ”linked” in their expression levels. This is another way to think of the ”explained variance” or information that PCA tries to preserve. With the method to identify PCs via their ”barcode”, they could be easily detected visually as well. Biologically, they might be controlled by the same transcription factor or occur in the same pathway, or if they have opposing ”weights” in the linear combinations, ie PCs, have antagonistic effect on each other, eg one is upregulated while the other one is downregulated. By our method to identify the PCs via their barcode, we would easily track.

On projected simulated data we could apply k-means or another clustering algorithm and compare the centroids of the cluster with the basins of attraction or fixed points from the stability analysis.

We could then compute these points back into the higher-dimensional output space which would give us an idea of where approximately differentiated cell states may lie.

Still stability analysis can be done and the basins of attraction (or any other point in the lower-dimensional space) can be computed back into the original space and suggest where in the higher-dimensional input space attractor and potential differentiated cell states lie.

D. One problem of Manifold Learning techniques: Scalability

During literature research we noticed that manifold learning isn't widely used in papers but people seem to still rely on standard PCA and other linear DR methods, especially for single-cell data. One reason for that might be that manifold learning techniques depend on the shape of the data, ie the mapping is implicit, and do not scale well [38]; also computationally these methods are expensive. One of the main problems of the manifold learning techniques explained here, is that they are only defined in a neighbourhood of the input data and they usually scale very poorly. In section VII we refer to a recent technique that tries to solve this issue.

E. Genetic Algorithm

The GA seemed like a pretty straight forward approach and a good alternative to traditional optimisation methods. Eg compared to non-linear programming models, less variables and constraints are necessary (as the variables largely depend on each other). We mainly chose the GA for its flexibility and for practical reasons to encode the action potential in the fitness function. The results we got were largly dependent on the inital population and the way we designed the mating process where we experimented a bit.

F. Different kernels in Kernel Density Estimation (KDE)

Other kernels implemented for the KDE in 2D including the linear kernel, cosine kernel, tophat kernel and epanechikov kernel have been implemented but not evaluated and compared yet.

VII. 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 C. Provided the data is available, the easiest as fastest implementation could probably be achieved by 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 improvements (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.

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.

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 Linearly-Local Embedding (LLE) as it addresses the regularisation problem which leads to distortions in LLE maps outlined in the theory part in the github overview. 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 is less computationally expensive [35]. 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 in case of 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 locality-preserving. [6] 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: Manifold learning in scikit-learn have proven to be good and there is also a scalable CPU-bound manifold learning package available in Python (*Megaman* [39]). However, especially for the application on single-cell data, better manifolds learning methods need that improve scalability and reduce the computational cost need to be developed. The Neural networks/ deep generative models approaches mentioned in the literature research section, especially *scvis* in [11] and also *scVI* in [12] are highly interesting and might replace the state-of-the art method t-SNE soon. Other methods based on neural networks hat are scalable in terms of training when using GPU are being developed.
Another interesting and novel manifold learning technique, not yet applied to single-cell data, found recently is the *Uniform Manifold Approximation and Projection (UMAP)* [14]. 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. With reference to [14] its running time is indeed astonishing. It is also 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. This project holds great potential and could be applied to a any type of data set, including biological data.

C. Genetic Algorithm

One might look into other methods to compute th MAP such as simulated annealing or other evolutionary algorithms..

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 determines their performance. And even though the results were not optimal, the GA suited our 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 the appendix.

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); this did not seem any relevant for our project.

VIII. Conclusion

For a general conclusion regarding the project, we refer to the group report. A personal conclusion can be found in the appendix.

APPENDIX

A. Personal motivation and conclusion

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 excited about immersing myself in the research in this field. As I have had a similar project 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 easier and faster if we used CuPy [40] and our project was based on Python; I think, initialising kernels in CUDA can be a lot of work and there might be more convenient tools available.
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).
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 Julia as it is now again for a bigger project where the main purpose is not to contribute to the community; for this project the programming language was a great limitation. However, I am very grateful for having had this learning experience. 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.

B. (Personal) challenges and comments

1) 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 situate it in the wider literature.

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

- **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 different DR methods on the data set. In another programming language such as Python this would have not been the case, as eg scikit-learn [35] 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.
- **Conflicts between packages.** Conflicts between packages like *DifferentialEquations.jl*, *MultivariateStats.jl*, Gadfly.jl 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.
- **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. The excellent documentation for data types and standard packages can serve as a model.

C. 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, the language stable, documentation great and many tools are available. 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 these Python benchmarks were actually written – probably in native Python and not in an optimised version. It might be more insightful 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, numerous people online have shown that it is possible to achieve the same benchmarks when one uses Python better and its tools. In fact, 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 could have been used for 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 is already 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 from 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.

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

The following 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.

E. *My top 7 learnings:*

The plan was "to look before we dived deep". We wanted to get a feel of what's already "out there", and whether our ideas have been previously approached. Literature research 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 [27].

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, if not, find out why not, and the graphics okay (the editor *Atom* often seemed to be sluggish) 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, some of us 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 a specific 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 stressed enough. This avoids time-consuming merging phases and avoid double-work which often happened as we were working on the same code at the same time. Personally, I am strongly in favour of 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 reproduced when the code isn't reviewed or even shared.

4) Learning #3: Comment code: Commenting code is especially necessary when working together on code. Not only 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 recently been addressed by a startup from Imperial (*Codeplot*); commenting therefore prevents time drains and errors. However, comprehending commented and readable code can be enriching and help you develop empathy as well. 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.

5) Learning #4: Do not get lost in literature research, keep track of your readings: This might especially be a personal learning. Constant alerts from *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 10-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.

7) Learning #6: Don't procrastinate: You can start your report early; don't leave things until the last minute. Maybe that's just something for (active) procrastinators like me.

Maybe these top learning are not only useful for me but for someone else as well (as reports from previous year's seemed to be shared sometimes).

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] Jenkinson G et al. *Potential energy landscapes identify the information-theoretic nature of the epigenome*. Nature Genetics 49, 719-729, 2017.

[5] scikit-learn. *https://plot.ly/ipython-notebooks/principal-component-analysis/*, 15 March 2018.

[6] Li Y and Lu R. *Applying Ricci Flow to High Dimensional Manifold Learning*. arXiv:1703.10675v4, 26 March 2018.

[7] Pierson E and Yau C. *ZIFA: Dimensionality reduction for zero-inflated single-cell gene expression analysis*. Genome Biology, 16:241, 2015.

[8] Lopez R et al. *A deep generative model for gene expression profiles from single-cell RNA sequencing.* arXiv:1709.02082v4, 16 January 2018

[9] Lopez R et al. *A deep generative model for gene expression profiles from single-cell RNA sequencing*. arXiv:1709.02082v1, 7 September 2017.

[10] https://github.com/epierson9/ZIFA, 27 February 2018.

[11] Lin C et al. *Using neural networks for reducing the dimensions of single-cell RNA-Seq data*. Nucleic Acids Research, Volume 45, Issue 17, 29 September 2017.

[12] Ding J, Condon AE, Shah SP. *Interpretable dimensionality reduction of single cell transcriptome data with deep generative models*. *https://www.biorxiv.org/content/early/2017/09/01/178624*, 1 September 2017.

[13] Ding J, Condon A and Shah SP. *Interpretable dimensionality reduction of single cell transcriptome data with deep generative models*. *http://dx.doi.org/10.1101/178624*, 1 September 2018.

[14] McInnes L and Healy J. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. arXiv:1802.03426v1, 9 February 2018.

[15] McCarthy D and Wills QF. *Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R*. Bioinformatics, 33(8), 2017, 1179-1186, 14 January 2017.

[16] Bioconductor. *https://www.bioconductor.org/help/course-materials/2017/BioC2017/Day2/Workshops/singleCell/doc/workshop.html*, 25 March 2018.

[17] Debahuti M et al. *Feature Selection in Gene Expression Data Using Principal Component Analysis and Rough Set Theory*. Software Tools and Algorithms for Biological Systems, pp 91-100, 15 March 2011.

[18] Dash M and Liu H. *Feature selection for classification*. Intelligent Data Analysis, Volume 1, Issues 1-4, 1997.

[19] Guyon I and Elisseeff A. *An Introduction to Variable and Feature Selection*. Journal of Machine Learning Research 3, 2003.

[20] Saeys Y, Inza I and Larranaga P. *A review of feature selection techniques in bioinformatics*. Bioinformatics, Colume 23, Issue 19, 1 October 2007.

[21] Marco E et al. *Bifurcation analysis of single-cell gene expression data reveals epigenetic landscape*. PNAS, https://doi.org/10.1073/pnas.1408993111, 30 December 2014.

[22] Ferrell JE. *Bistability, Bifurcations, and Waddington's Epigenetic Landscape*. Current Biology, Volume 22, Issue 11, 5 June 2012.

[23] . Asp P et al. *Genome-wide remodeling of the epigenetic landscape during myogenetic differentiation*. PNAS, https://doi.org/10.1073/pnas.1102223108, 31 May 2011.

[24] Maze I and Nestler EJ. *The epigenetic landscape of addiction*. Annals of the New York Academy of Sciences, https://doi.org/10.1111/j.1749-6632.2010.05893.x, 27 January 2011.

[25] Lord J and Cruchaga C. *The epigenetic landscape of Alzheimer's disease*. Nature Neuroscience, Volume 17, Numer 9, September 2014.

[26] Morris MR and Latif F. *The epigenetic landscape of renal cancer*. Nature reviews Nephrology 13, 47-60, 2017.

[27] Guo J and Zheng J. *HopLand: single-cell pseudotime recovery using continuous Hopfield netowrk-based modeling of Waddington's epigenetic landscape*. Bioinformatics, Volume 33, Issue 14, 15 July 2017.

[28] 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.

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

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

[31] 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.

[32] https://github.com/NetLand-NTU/NetLand

[33] 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.

[34] SCUBA. *https://omictools.com/single-cell-clustering-using-bifurcation-analysis-tool, 25 February 2018.*

[35] Scikit-learn. *http://scikit-learn.org/stable/modules/manifold.html*, 25 March 2018.

[36] *https://github.com/rasbt/python-machine-learning-book/blob/master/faq/dimensionality-reduction.md*, 25 March 2018.

[37] Banda JM, Angryk RA, Martens PC. *Quantitative Comparison of Linear and Non-linear Dimensionality Reduction Techniques for Solar Image Archives*. 25th International FLAIRS Conference, 16 May 2012.

[38] Sorzano COS, Vargas J and Pascual-Montano A.*A survey of dimensionality reduction techniques*. *https://arxiv.org/ftp/arxiv/papers/1403/1403.2877.pdf*

[39] Megaman. *https://github.com/mmp2/megaman*, 25 March 2018.

[40] CuPy. *https://github.com/cupy/cupy*, 25 March 2018.

[41] Numba. *https://numba.pydata.org/*, 25 March 2018.

[42] Cython. *http://cython.org/*, 25 March 2018

[43] Veys S. *A comparative review of dimensionality reduction methods for high throughput single-cell transcriptomics* Universiteit Gent, 2016.