Electronics and Computer Science
Faculty of Engineering and Physical Sciences
University of Southampton

Ashwinkrishna Azhagesh

25/03/2025

An AI Approach to Chaotic Physical Systems:

Project supervisor: **Adam Peugeot**

Second examiner: **David Millard**

Progress report submitted for the award of
Bachelors of Science

Abstract

Empirical laws are mathematical generalisations found through observing the physical world. It has taken us centuries of gathering data, keen research along with repeated experiments, and no doubt plenty of talented scientists to discover these laws. Leading us to understand everything from the mysteries that govern the collision of two objects to the shape of the path planets thread upon.

Recent advances in neural networks including increases in computational power permit us to train models, that replicate, fasten and automate our discovery of empirical laws. This extends to even noisy chaotic systems such as the double pendulum. Combined with white box models, symbolic regression and explanable A.I., we can peer into the ”mind,” of how such models, process data and conclude their observations. Human congition is inherently finite in its capacity for thought and observational ability, has been historically overcome through the development of new tools such as the microscope. Similarly, congitive biases can be mitigated, by utillising artificial intelligence, which is a rapidly emerging technology capable of expanding our perception and analysis.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.

- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.

- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for

this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

ECS Statement of Originality Template, updated August 2018, Alex Weddell aiofficer@ecs.soton.ac.uk

Abstract

I would like to thank my supervisors, Professor Adam Peugeot and Professor David Millard, for all the help and advise I received throughout this project.

Contents

1 Introduction:

1.1 Motivation:

As there is more data being generated than ever before and new experiments, we need a systematic and automatic way to deduce various mathematical patterns and laws in these data. Through the use of symbolic regression we can utilise these data, and in an explainable manner deduce various new physical laws. In this research I have also extended this beyond physics and have applied this to biological data sets which is a novel application of this method. Perhaps extend this beyond or add a sectionsaying this can also be applied to nlp and that it can learn the rules in language and writing etc.

Talk a little about the way this is used outside of this niche use case, and in research, so of course I need to look and research into this.

2 Previous Work:

2.1 Literature Review:

3 Noise:

In this section, I aimed to explore how noise affects the model, and potential ways to mitigate it. Continuing onwards from the previous model, in the data generation step, noise was artificially added, and the results were observed.

So in order to model the noise, I used the python random library, and generated random numbers between 0 and an ever increasing amount of randomness, in oder to guage the accuracy as noise increased for the model. I was also part

3.1 How noise affects the model:

So in order to add noise to the generated data set, I imported in random, and used the randn.int function. In order to vary the inputs, another function was created that incrememntally passes in higher numbers as parameters to the random function, allowing each set of generated data to incrememntally become more and more noisy. Then the symbolic regression model is run on these new data sets, and the resulting equations levels of noise are then plotted in a graph. Furthermore using the Time library to measure the amount of time it takes to run the model as the amount of random error increases.

3.2 How to mitigate noise in data:

Ways to mitigate the noise and it's affects on the model were explored. Functions such as "denoise," in the symbolic regression library helped to some extent. However after a certain point, such methods do not seem to offer much assistance.

I also made my own denoise algorithm. I implemented various different denoise algorithms to see what effects they had. Firstly I implemented a simple moving avergae as a way to mitigate the noise in the dataset. reword this -¿ " Simple and fast, smooths data well by averaging neighbors. However, it blurs sharp changes and is sensitive to extreme outlier values, pulling the average significantly and distorting the signal."

These were my results, this is the pseudo code, explain the alogrithm.

The second denoise algoirthm I implemented is a median filter, and this is what effects it has, and this is how i implemented it. Insert Pseudo code. reword: "Excellent at removing spikes and preserving edges better than averaging. Less affected by outliers. Can sometimes slightly distort the overall shape of the signal, especially with large window sizes."

Finally this is the third algorithm that I had implemented for denoising. Wavelet Denoising, this is the effects, and this is the pesudo code. Reword this -¿ "Transforms data to isolate noise, preserving both smooth and sharp signal features effectively. More complex to understand and requires careful selection of wavelet type and parameters for optimal results, which can be tricky."

3.3 Modelling the noise:

4 Writing my own symbolic regressor from scratch:

4.1 The core;

The core and essential part of any symbolic regression model, lies in the way it at the simplest level, generates and traverses the search tree of possible equations and expressions that may fit the data presented to it.

In order to save time, and to test if my expression generation was working as intended, i have started with simple 2 variable equaitons, and also pass in the speific operations used in the equation. Furthermore this is also extended to handle constants and more later on.

enter in the pseudo code here.

Then I further improved this, by designing a resursive way to generate these expressions, to allow to generate more robust equations from the given variables. Also this is dynamic, so it can

enter in pseudo code

4.1.1 exploiting physical properties

The next step is to then start to truncate these generated expressions as much as possible to prune the search tree. One of the ways you can do this is through exploiting the symmertrical property of physical equations and how they are mathematically equivalent. Such as removing duplicate expressions.

This is how I achieved that.
Give pseudo code here.

4.1.2 Dealing with constants:

Another way i further pruned the amount of expression, is through filtering all the expressions generated through the newer recursive generator, by removing all the expressions that did not contain all the specified variables. This is in order to save further time later on during the evaluation seciton.

Insert in pseudo code:

Then later on i designed it next to work with nested expressions and i wanted it to work with more than one constant.

insert in some code that has changed.

I had then filtered out any of the expressions generated that did not have both of the constants. This is mainy to save on some computation time in evaluating some of the redundant expressions. This is how I managed to do that.

Insert in pseudo code.

4.1.3 Dealing with powers:

So appling powers to expressions, I applied the power to the expression. This also allows you to prune the search tree further by not needed to generate redundant expressions with powers.
This is the pseudo code.

Then I filtered based on if the expression contained the power, this allows me to further prune the tree. In a more robust model, this is dervied from scratch, however for the sake of computation time, and flexibility, I decided to proceed with this approach as it saves some time.

this is the pseudo code.

4.1.4 Chaining powers and constants:

The next step is to chain together powers and constants, such that both are applied to the expressions. It can already be chained as with the design it already has. However it needs to be filtered poperly in order to maintain the least amount of expressions possible.

Te constants filter can be used, but the powers are inside the constants, and therefore the older power filter does not work as intended. Therefore i needed to redesign it such that it will function recursively.

This is the code.

However, sometimes this gives off constants that are chained, such as sin(sin()), and so to filter this futher, I want to filter out expressions with more than one instant of the constant that is chained.

this is the code - filter single constant

4.1.5 Loading data:

Next I needed a way to load the data I has typed up. At this point I was focused on testing as quick as possible and in order to proceed in a prompt manner I made up some dummy data values. Afterwards I made the decision to keep the data as a numpy array, because this will be faster then a text file, there are some various reasons for this, such as numpy arrays being stored in memory, the efficiency of the nderlying data format it is stored in (binary), and finally numpy uses c, and so it vectorises operations, making it far faster.

Insert in pseudo code.

As you can see I check if the number of variables entered matches the shape of the arrary in X, which here is the input data, and y being the target data, as in the final result. Ie x contains the mass and acceleration values, and y is the array of the result of the equation f = ma, so it only contains the value of f in it. This is a basic check to make sure the number of columns all have a corresponding variable.

4.1.6 Evaluating expressions:

Next I evaluate the expressions that I had generated, and i assign the variables to a column of the data, in increasing order. Then this is substituted into the equation, and the expressions are run, and there is an array of outputs of the expression. This essentially evaluates every generated expressions that has been pruned, and returns a np arrya of the results of those expressions based on the input data.

insert in pseudo code here.

Then like the paper suggested, insert in paper here, I used a medium error description length loss function, and have implemented it in the same way as in the paper. Using error squared, making all the errors positive, and added 1 as a constant to ensure that all the errors are greater than the value, when taking the log.

Insert in pseudo code.

Then furthermore I also implemented 2 other loss algorithms, specifically root mean squared loss as well as mean absolute error.

insert in pseudo code.

This was to help bridge and improve upon the loss algorithm used in the paper, as these two have their own advantages, and a combined hybrid approach seemed smarter.
Explain why later.

4.2 Polynomial Fit module:

Now that the simple, core of the algorithm works, and is adapted to take care of contants, powers, variables, generate expressions, and filter out the redundancy using physical properties of the world such as symmtery, I now aimed to futher extend the program by writing a polynomial fit module. The aim of the polynomial fit technique is to.

Why: Many functions in physics (or parts of them) are low-order polynomials.

Why: It's a computationally very cheap method for this specific function class.

How: It attempts to fit the given data to a sum of polynomial terms.

How: It generates all possible polynomial terms up to a specified low degree (e.g., degree 4).

How: For each data point, this creates a linear equation where the unknowns are the polynomial coefficients.

How: It solves the resulting system of linear equations using standard methods like least squares.

How: The Root Mean Squared Error (RMSE) of the fit is calculated.

How: If the RMSE is below a predefined tolerance $(_pol), thepolynomialisacceptedasasolution.$

Effect: It acts as a fast base case in the recursive algorithm, quickly solving problems that are simple polynomials.

Effect: It can also solve sub-problems that are transformed into polynomials by other modules (e.g., dimensional analysis, inverting the function).

4.2.1 Data Loading:

So to start, I began by creating the data loading function. The aim was to take in a numpy array, with the data, along with a list of variables. Then comparing the shape of the data column and the number of variables in order to make sure the input is sufficient.

This is the pseudo code.

4.2.2 Generating polynomial expressions:

Then the next step is to generate polynomaial expressions, and then it will return a list of polynomial expressions on the list.

Pseudo code.

This function, generate$_p olynomial_e xpressions, createsallpossiblesymbolicpolynomialexpressionsbycombiningalistofcoef f$

4.2.3 Filtering the Polynomial expressions:

The filter$_e xpressions function programmatically filters symbolic expressions using structural and semantic constraints. Le$
$suited for large - scale symbolic filtering tasks where strict mathematical structure must been forced.$

insert in pseudo code.

This initial version only worked for symblic constants, ie sin, cos etc, and didn't work for numbers, or nteger coefficients, i caught this error during testing and I rewrote the function so that it works for integer coefficients.

insert code:

4.2.4 Evaluating expressions:

Now I need to take the filtered expressions, and try fit the model to the dataset np array. The model fitting logic function fits polynomial expressions to input data by finding the best set of coefficients that minimize the error between the predicted and actual output values. It tests multiple polynomial degrees (up to a specified maximum) and selects the one that provides the lowest error, ensuring an optimal balance between accuracy and complexity.

I use root mean squared error to calculate the loss, and the funciton returns a list of loss, per expression.

So i take an expression, then I substitute in the variables using the input data, calculate the predicted y value of the said equation, then take it away from the true value of y the target and then use that to calculate the rmse.

pseudo code.

Best polynomial fit:
Now that I have the list of expressions and the corresponding rmse values, i pick the lowest rmse as the most accurate polynomial fit for the data.

insert in pseudo code

4.3 Dimensional Analysis:

Physics Constraint: Physical equations must be dimensionally consistent (units on both sides must match).
Strong Simplification: This dimensional constraint severely limits the possible forms of the unknown function.
First Step: AI Feynman applies dimensional analysis as the very first attempt to simplify the problem.
Unit Representation: Units of variables (like mass, length, time) are represented as vectors of integer powers.
Linear System: A linear system is set up based on the unit vectors of the input variables and the target variable.
Dimensionless Combinations: Solving this system and finding the null space reveals combinations of variables that are dimensionless.
Problem Transformation: The original problem is transformed into finding a function of these new, dimensionless variables.
Reduced Variables: This process typically reduces the number of independent variables the algorithm needs to search over.
Search Space Reduction: A smaller number of variables drastically shrinks the combinatorial search space for subsequent steps.
Efficiency Boost: It makes Polynomial Fit, Brute Force, and Neural Network-guided searches significantly faster and more likely to succeed.

4.3.1 Handling Units:

So i went to the ai feynman database website, and downloaded their units.csv to get a better idea of all the units in the dataset, I was dealing with. Then i had a look at all the required units, and then made a unit table, array, so that each unit corresponds to a unique power of the bsaic si units which I also implemented, as an array/list.

code:

4.3.2 Construct Matrix and Target Vector:

This function constructs the dimensional matrix M and target vector b essential for dimensional analysis. It accepts lists of independent and dependent variable names and a dictionary mapping variable names (keys) to their unit vectors (values).
Technical Implementation: Unit vectors for independent variables are retrieved via dictionary lookup, using lowercase variable names (var.lower()) to ensure case-insensitivity. These vectors are efficiently assembled into the columns of matrix M using numpy.column$_stack.Thedependentvariable's unitvector forms vector b.try...except KeyErrorbl$
Design Rationale: Case-insensitivity enhances usability. numpy.column$_stack offers performance.Explicit error handling prever$

code:

4.3.3 Solving Dimension and Basis Units:

This function determines the exponents for dimensional scaling and dimensionless groups by solving the linear systems Mp = b and MU = 0. It takes the dimensional matrix M and target vector b as input. The function first converts these NumPy arrays into SymPy matrices (sp.Matrix) to leverage symbolic computation capabilities. It then attempts to find an exact particular solution p for Mp = b using SymPy's LUsolve method, chosen for its ability to yield rational solutions. Robust error handling via try...except addresses potential issues like inconsistent systems. Finally, it calculates the null space basis U of M using $M_sym.nullspace(), which identifies the combinations forming dimensionless groups. The function returns the symbolic solu$

Think of it like this: you have a target physical quantity (b, like Force) that depends on several input quantities (M, like mass, length, time).
Finding the "Unit-Fixing" Part (p = $M_sym.LUsolve(b_sym)$) :
The function first figures out the specific combination of powers (p) of your input variables (x) that you need to multiply together (x^p) $so that the result has the exact same physical units as your target variable(b).$
For example, if the target is Force ([M L T²]) and inputs are mass ([M]), length ([L]), and time ([T]), it would find p corresponding to mass¹ * length¹ * time².
It uses LUsolve from SymPy to try and find an exact (often simple fraction or integer) solution for these powers p.
Finding the "Dimensionless Combinations" (U = $M_sym.nullspace()$) :
After accounting for the basic units, any remaining relationship must involve combinations of input variables that have no units at all (they are dimensionless numbers, like Reynolds number).
The function finds all the fundamental ways (U) you can combine powers of the input variables (x^u) $such that the units completely can$
In essence, the function:
Separates the part of the formula responsible for getting the units right (p).
Identifies all the core dimensionless building blocks (U) that the rest of the formula must be made from.
This allows the main algorithm to later focus on finding the relationship between these dimensionless quantities, which is a simpler problem than the original one involving various physical units.

4.3.4 Data Transformation Function:

his function converts original physical data ($data_x, data_y$) $into a dimensionless form using previously calculated exponents(p, U).$
How it Works: First, it calculates a scaling $_factor for each data point by raising input variables(data_x) to the powers specified in p(u$

4.3.5 Symbolic Transformation Generator

This function generates the symbolic mathematical expressions corresponding to the dimensional analysis transformation. It accepts the original independent variable names (independent$_vars$) $and the exponent vectors for scaling(p) and dimens$
How it Works: It first creates SymPy symbolic objects for each input variable name using sp.symbols. Utilizing these symbols and the scaling exponents p, it constructs the symbolic expression symbolic$_p$ $representing the unit-fixing scaling factor(x^p) via sp.Mul. Subsequently, it iterates through each exponent vector u in U, building the corresponding s$

insert code:

4.4 Neural Network Fitting:

The next step crutical component, is using neural networks in order to simplify the expressions further.
In order to get a smotth and differentiable approximation of the function we are looking for, this is what the neural network allows us to do.
The neural network provides a powerful, universal function approximator (f_NN) $capable of learning intricate patterns directly fro$

$inspiredsimplificationtechniqueslikesymmetrydetection, separabilityanalysis, andcompositionalitychecks, whichrelyo$

4.4.1 SymbolicNetwork:

This class defines the neural network architecture used as a universal function approximator within the symbolic regression framework. It inherits from torch.nn.Module, the base class for all neural network modules in PyTorch.
Technical Implementation: The constructor ($_{i}nit_{)}initializesthenetworkstructure, acceptingthenumberofinputfeatures(n_input)anddefaultingtooneoutpu$
Design Rationale: This multi-layer perceptron (MLP) architecture provides significant expressive power. The Tanh activation was chosen as specified in the reference papers, offering a smooth, differentiable non-linearity suitable for approximating complex physical functions and enabling reliable gradient computation for subsequent analysis steps.

4.4.2 Preparing the data:

This function preprocesses raw input and output data (data$_x$, $data_y)intoaformatsuitableforPyTorchmodeltrainingandvalidati$
Technical Implementation: It begins by converting the input NumPy arrays data$_x$$anddata_yintoPyTorchtensorsoftypetorch.float$
Design Rationale: This design adheres to standard PyTorch practices. Tensor conversion is mandatory for framework compatibility. The train/validation split is critical for monitoring model generalization and preventing overfitting. TensorDataset provides a convenient pairing of inputs and targets. DataLoader is essential for efficient training, enabling batch processing (managing memory and potentially parallelizing computation) and automating data shuffling, which improves model robustness and convergence.

4.4.3 Training the Network:

This function orchestrates the supervised training process for the provided PyTorch neural network model. Its primary goal is to iteratively adjust the model's parameters (weights and biases) to minimize the difference between its predictions and the true target values using the training data, while monitoring performance on unseen validation data.
Technical Implementation: The function begins by transferring the model to the specified compute device ('cpu' or 'cuda'). It initializes the Adam optimizer, a common adaptive learning rate optimization algorithm, linking it to the model.parameters() and setting the learning$_rate.TheMeanSquaredErrorloss(nn.MSELoss), suitableforregression, is$
Design Rationale: This structure represents a standard PyTorch training loop. Using DataLoader enables efficient batch processing. The Adam optimizer provides robust convergence properties. MSE loss directly minimizes the squared prediction error. Explicitly setting model.train() and model.eval() ensures correct behavior of layers like dropout or batch normalization (if present). The torch.no$_grad()contextduringvalidationpreventsunnecessarygradier$

4.4.4 Predict Function:

This function performs inference using a trained PyTorch model, generating output predictions for a given set of input data. It is designed to take a trained model, input data as a NumPy array (x$_numpy), andthetargetcomputationdevice('cpu'or'$
Technical Implementation: The function first sets the model to evaluation mode using model.eval(). This is crucial as it disables layers like dropout or batch normalization that have different behaviors during training and inference, ensuring deterministic output. The input NumPy array x$_numpyisthenconvertedintoatorch.Tensorwithdtype =$
$torch.float32andtransferredtothespecifieddevice.Thecorepredictionstepoccurswithinawithtorch.no_grad() :$
$contextmanager.Thisdisablesgradientcalculation, significantlyreducingmemoryconsumptionandspeedingupcomputati$
Design Rationale: This design follows standard PyTorch inference practices. model.eval() ensures correct prediction behavior. Tensor conversion and device handling manage data compatibility with the model. The

torch.no$_g rad()context optimizes performance for inference. Returning a NumPy array provides a user-friendly output format.

4.5 Pareto Frontier Optimisation:

iThe Pareto frontier provides a principled way to manage the inherent trade-off between a formula's accuracy and its simplicity (complexity) in symbolic regression. Instead of seeking a single "best" formula, the goal is to identify all Pareto-optimal formulas: those for which no other discovered formula is simultaneously simpler and more accurate.

I am going to implement this by plotting on a 2d plane, where x represent complexity and y represents the mean error description length. As the algorithm generates candidate formulas (through brute-force search, transformations, or recursive combinations), each candidate is evaluated and potentially added to the set of points forming the frontier.

Crucially, any candidate formula that is dominated – meaning another formula on the frontier has both lower (or equal) complexity and lower (or equal) loss (with at least one inequality strict) – is discarded. This Pareto pruning significantly reduces the search space, especially when combining solutions from subproblems, improving computational efficiency and robustness against noise by inherently favouring simpler explanations for a given accuracy level. The final frontier presents the user with a spectrum of optimal solutions.

4.5.1 Points:

This Point class serves as a fundamental data structure within the Pareto frontier analysis framework. Its primary purpose is to encapsulate the essential characteristics of a single candidate formula evaluated during the symbolic regression process.
Technical Implementation: The constructor ($_i nit_{)} initialize each Point instance with three attributes: complexity (a numerical score representing the formula
Design Rationale: This class is designed to bundle the key metrics (complexity, accuracy) required for Pareto dominance checks with the associated formula (expression). By grouping these related pieces of data into a single object, it simplifies the management and comparison of candidate solutions. Functions operating on the Pareto frontier can easily access the necessary complexity and accuracy attributes from each Point object to determine if one solution dominates another, thereby facilitating the efficient maintenance of the non-dominated set of optimal formulas.

4.5.2 Pareto Point Set:

The update$_p areto_p oints function maintains a Pareto frontier of symbolic expressions, balancing complexity (models implicit$
The function filters the combined list to retain only non-dominated points: a point A dominates point B if A is both no more complex and no less accurate, with at least one being strictly better. This ensures only the best trade-offs remain.
In symbolic regression, this is critical because we aim to discover expressions that are not just accurate but also interpretable — meaning low complexity. The Pareto frontier lets us visualize and choose among optimal models, avoiding overfitting (too complex) or underfitting (too simple).

4.6 Plotting:

I wanted ways to visualise and understand in a visual sense what was happening behind the scenes in my symbolic regression program. Humans are visual creatures and looking at plots offers a more powerful way to

understand data rather than staring a strng of numbers or expressions, trying to manually find patterns or mistakes.

I decided to use plotly, instead of the usual matplotlib, as I personally think it is more asthetically pleasing and it has a better interactive environment.

4.6.1 Plot for Pareto front:

Visualizing the Pareto frontier is highly beneficial for interpreting the results of symbolic regression. It provides an intuitive graphical representation of the fundamental trade-off between model complexity and predictive accuracy (or loss).
Code Description: The provided code utilizes the plotly.express library to generate this visualization. A Pandas DataFrame (df) structures the data, holding columns for 'Complexity' (x-axis), 'Loss' (y-axis, representing inaccuracy like MEDL), and the corresponding symbolic 'Expression'. The px.scatter function creates a scatter plot mapping complexity against loss, crucially using the 'Expression' data to label each point directly on the plot via the text argument. Further customization using fig.update$_t racesenhancespointvisibility(markersize, color)andlabelpositioning$
Utility and Rationale: This visualization allows researchers to readily identify the set of non-dominated solutions. By plotting quantitative metrics, it elucidates points where significant gains in accuracy require substantial increases in complexity (an "elbow" region), facilitating informed model selection based on the specific balance desired between interpretability/simplicity and predictive power. The direct labeling of points with their expressions provides immediate context for each optimal solution discovered.

4.7 Testing:

Every module, function and file, were thoroughly tested using dedicated tests. The boundary conditions and inserted tests to make sure the function behaved as envisioned. There was also significant robust testing functions written when chaining together various techniques and models, in order to ensure everything was working smoothly.

5 Applying the model to Biological Data:

6 Broder Use Cases:

7 Project Management:

8 Conclusion:

9 Project Planning: