Statistical and Numerical Methods

Simulation of a Dynamical System:

This means that we use the differential equations of the system in order to predict the way it will behave in the future, given a specific initial state. The way it is usually done is by applying the Runge-Kutta methods and similar variants. The numerical solution of the differential equation can be a very powerful tool, which can help us study and understand the dynamical system, which in our case represents a game.

Linear Regression

This problem has been extensively studied for centuries. It could be stated as:

(Regression): *Given a set of observations,* $\{(\boldsymbol{x}, y)\}_{i=1}^{N}$ *,* y *continuous, find a proper model for the* $y = f(\boldsymbol{x})$.

In Linear Regression, the above formula is supposed to be of a linear form, i.e. $y = h(\boldsymbol{a}, a_0) = \boldsymbol{a}^T * \boldsymbol{x} + a_0$, where $\boldsymbol{a}, a_0$ describe the coefficients of the linear equation. The set of all possible candidate models that follow some form (in this case we have the linear form) is also called the *Hypothesis Space* . In linear regression we study the hypothesis space of the linear models and our goal is to find a model which satisfies some kind of optimisation criterion.

The solution to the above problem is traditionally given by the Ordinary Least Squares solution. By ordering our data in two matrices,

$$X_{aug} = \begin{bmatrix} \boldsymbol{x}_1^T & 1 \\ \boldsymbol{x}_2^T & 1 \\ ... & 1 \\ \boldsymbol{x}_N^T & 1 \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ ... \\ y_N \end{bmatrix}$$

we can get the *error* vector, which is the difference between the observed values of Y and the values predicted by some model of the hypothesis space.

$$\boldsymbol{e} = Y - X_{aug} * \begin{bmatrix} \boldsymbol{a} \\ a_0 \end{bmatrix}$$

The OLS solution is the hypothesis which minimises the function $J(\boldsymbol{a}, a_0) = c{*}\boldsymbol{e^T}{*}\boldsymbol{e}$. The optimal parameters $\boldsymbol{a}_{aug} = \begin{bmatrix} \boldsymbol{a} \\ a_0 \end{bmatrix}$ can be given by solving the equation

$$\boxed{X_{aug}^T Y = X_{aug}^T * X_{aug} * \boldsymbol{a}_{aug} \rightarrow \text{OLS Solution}}$$

Another path to find a solution would be *Maximum Likelihood Estimation.* Let us suppose that the measured $\{(\boldsymbol{x}, y)\}_{i=1}^{N}$values are measurements of Random Variables, $\boldsymbol{X}, \boldsymbol{Y}$. By using a model from the hypothesis space on the random variables, we can define the $P(\hat{\boldsymbol{Y}} = \hat{y_o} | \boldsymbol{X} = \boldsymbol{x}_0; \boldsymbol{a}_{aug}\}$, which is the distribution of the predicted variable.

By using the empirical distribution from the observed datasets, we can make the demand that the $fun(\boldsymbol{a}_{aug}) = P(Y_{observed} | X_{observed}; \boldsymbol{a}_{aug})$ be maximised. This is a form of Maximum Likelihood estimation, in the case of supervised learning problems. Maximum Likelihood Estimation is generally proven to be able to give solutions that satisfy many properties that are desired, namely estimations that are asymptotically: unbiased, consistent, achieving the Cramer Rao bound and many other properties.

When some certain assumptions hold true, mainly regarding the Gaussianity of the random variables, the solution obtained by the OLS is also the Maximum Likelihood Estimator on the hypothesis space. This means

that we can calculate the ML estimator, which is desired, using a simple and elegant algorithm derived from the Least Squares optimisation.

Ordinary Least Squares can also be easily applied to find regression models that belong on the *polynomial* hypothesis space (Polynomial Regression). Generally, in any model that can be described as a linear combination of basis functions, $h(\boldsymbol{a}, \boldsymbol{x}) = \boldsymbol{a}^T * \boldsymbol{\Phi}(\boldsymbol{x})$ OLS can be easily applied by creating the new $X^{'}$ matrix using the $\boldsymbol{\Phi(x)}$ functions. It should be noted however that while a hypothesis of gaussian distribution on the independent variables generally does hold true, the same cannot be said for the new independent variables, $\boldsymbol{\Phi}_i(\boldsymbol{x})$, because when $\boldsymbol{X}_i \sim N$ then the $\boldsymbol{\Phi(X_i)}! \sim N$ and therefore the OLS might not be able to approach the MLE in the new hypothesis space. Nevertheless, polynomial regression is able to give us an easy way to deploy decent non linear models using OLS, which can be quite helpful in a large variety of situations.

Logistic Regression

The classification problem statement is quite equivalent to the statement of regression, the only noteworthy difference is that the values of the dependent variable are discrete and not continuous. In traditional statistics, there have been many attempts to apply a calculus equivalent to the one used for Linear Regression (i.e. define an equivalent hypothesis space and optimise with an equivalent algorithm) in order to solve classification problems. A classification problem is called *binary* when the dependent variable is binary and *multiclass* when the dependent variable is discrete and not binary.

One first attempt to solve the binary classification problem was to use the Least Squares solution of a linear model to estimate the *probability* of the event that the output variable has either one of the two possible values. We create a Y matrix by assigning to y a different number, according to the corresponding class, for example (0,1). The linear model is trained with OLS by having this new Y dataset as output goals. To get the final decision regarding the class, a step function is applied to the output value.

While this attempt is simple, intuitive and straight forward is does come with some problems. The most noteworthy is that the probability estimate is, in most cases, not good enough and it is not rare to get values that do not belong in the $[0, 1]$ interval - this means that the input to the step function before the final decision is made could be negative. Such problems render the algorithm impractical and demand some kind of solution to be found.

This solution is given by using a special non linear function after the output of a linear model, one which could help us estimate the probabilities in a more robust and reliable way. In most cases, this function is the so called *Logistic Sigmoid* function,

$$f(y) = \frac{1}{1 + e^{-y}}.$$

where we can easily observe that the output variables of this function belong in the $[0, 1]$ interval, this signifies that by using this function we could improve our probability estimates. The new hypothesis space described as $h(\boldsymbol{a}, a_0, \boldsymbol{x}) = \frac{1}{1+e^{-(\boldsymbol{a}^T * \boldsymbol{x} + a_0)}}$ for the probability estimates, the final decision is taken by applying the output in a step function.

It can be proven that the maximum likelihood solution for the hypothesis space specified above is given by optimising the *log-likelihood* or *log-loss* function. This function is actually the empirical cross entropy of our dataset (which describes a distributional difference between the actual probability distributions and the ones estimated by our model), by minimising the cross entropy we demand that the distribution estimate is "close" to the actual distribution of the measured y values. Unfortunately, no close form solution exists for this problem (unlike OLS) and we have to apply iterative algorithms (like gradient descend) to get a solution.

When the problem is multiclass, we can use a similar tactic. We use a larger number of linear models (as many as the number of different classes) and train each model so that it estimates the $\hat{y}_i = Pr(Class = C_i | \boldsymbol{x}_{in})$. By inferring one input pattern from every single linear model we get a $\hat{\boldsymbol{y}}$ vector, describing the probabilities for each different class. We create the goals by using one hot $\boldsymbol{y}$ vectors, where the position (index) of the $'1'$ digit

signifies the current class. In order to have a proper probability estimate we have to use a special function, like we did with the Logistic Sigmoid earlier on. This function is called the *softmax* function,

$$\sigma_i(\hat{\boldsymbol{y}}) = \frac{e^{\hat{y}_i}}{\sum_{\forall class} e^{\hat{y}_j}}, \forall i.$$

The algorithm described above is usually called *Multinomial Logistic Regression*. One can easily see that the model can actually be described as a *Multilayer Perceptron* with no hidden layers, using the softmax function after the output of the linear perceptrons to estimate the probabilities vector. The training is done by minimising the cross entropy function, in this case for the multiclass problem (i.e. this is not a binary cross entropy).

Just like before, we can use a Polynomial augmentation of the dataset in order to be able to create non linear models, with a higher capacity to understand harder relations and functions. This is one of the simplest ways to deploy a non linear multiclass classification model, without the need to write Backpropagation algorithms (like in the deep MLP) or create a large amount of binary models to solve the multiclass problem (like with the non linear SVM models).

Density Based Clustering Algorithms

The problem of clustering a specific set of points could be stated as:

(Clustering): *Given a set of points,* $\{\boldsymbol{x}_i\}_{i=1}^N$, *find divisions of the set so that elements of each subset are more similar to each other than elements of the other subsets.*

In density based Clustering, a cluster is a set of points where there is a significant amount of point density. A classic density based algorithm is DBSCAN, which classifies each element of the dataset in 3 different categories - either noise, border or core point. The core points belong in the "interior" of a cluster, the border points belong in the "exterior" of the a cluster and the noise points do not belong in any cluster. The decision about the nature of every point is based on *the amount of neighbours that can be found in an area near the point.* If the amount of neighbours of one point surpass a specific threshold then the point is a core point, if it does not surpass the threshold but belongs in the are of a core point then it is classified as a border point. If neither of these two statements hold true, the point is a noise point.

Unlike algorithms from traditional Pattern Recognition, like kmeans, DBSCAN has the impressive ability to find arbitrarily shaped clusters, with the only demand that the points are dense enough. However it does come with its problems, mainly the fact that we have to choose two parameters for the algorithm, the radius which defines what the interior of the area of each point is and the point threshold to decide on the category of the point - problems that become quite hard to solve in spaces with dimensions ≥ 4 where visualisation cannot be easily applied. Also, the algorithm cannot easily find clusters of data that have substantially different densities, even though it might be needed to do so.

An algorithm that can help with some of these problems is the improved OPTICS algorithm. This algorithm is quite similar to the DBSCAN algorithm, but requires a fewer amount of parameters and is able to give us a *reachability plot* which can help us study a hierarchical structure of the data. This plot reflects the points of the dataset in a way so that the points that are actually neighbours on the dataset also are near each other on the reachability plot. This has the effect that the points that naturally form a density based cluster appear as valleys on the *reachability plot.* The more deep a valley is, the more dense is the point distribution. By taking advantage of the y-axis, which has values of the reachability distance, we can choose some value for the radius parameter specified above and apply the dbscan algorithm. The hierarchical structure could allow us to deploy the algorithm multiple times or even recursively, without the need to pay an extraordinarily high computational cost.

Principal Component Analysis

The Principal Component Analysis is a method which analyses a set of statistical observation vectors, generally described using a matrix X where every row corresponds to one observation, to a new set of vectors. The new dataset has the property that the features (i.e. the columns) are uncorrelated, and that the direction of the i_{th} feature is the maximum variance direction among the set of all directions that are orthogonal to every single direction of the previous features. An orthonormal base using these directions can be easily constructed by calculating the principal components, which is done by solving the eigenvalue problem of the $X^T X$ matrix. The transform is finally done by projecting the original X matrix on the principal component matrix.

The method can prove to be quite helpful in a large variety of pattern recognition problems. The reason why we use this method is to have some insight on whether some specific set of points on an $n-$dimensional space belong on a surface, a curve or are just observations of one specific point. For example, if the principal component analysis of a set of points belonging in a 3D space returns a feature variance that is zero, then we can make the conclusion that the points are not observations of points of a surface.

Computational Analysis of a 2D dynamical system

In this point, I will try to use computational and statistical methods to have some insight on how the dynamical system describing the game actually works. I propose a methodology that could generally be used in many continuous time dynamical systems that describe games. The aim of this methodology is to help a researcher acquire some intuition on the dynamics of the game before trying to tackle the problems in a mathematically analytical way. My method can be useful for the researcher, especially in cases where the differential equations are too complex to be solved analytically and/or when the system has a large number of players (> 3) and therefore the visualisation of the state space cannot be achieved.

An abstract description of my method is the following:

- Compute the $\dot{x}$ vector on a set of points $\{x\}$ that you can define. It is frequent that this set of points is defined using a grid (e.g. sampling points on a vector space using a constant sampling rate for every simple vector component). After the computation, plot the $\dot{x}$ using a so-called quiver plot. This plot is called the phase portrait, and it can be quite helpful, especially in understanding what the steady state behaviour of the system is for a large variety of initial conditions. The downside of this step is that it actually works only in games where the number of players is less than 4, as the visualisation shows problems already for the 3D case.
- Simulate the system, using some initial conditions that you can define. By numerically solving the system for a specific initial condition we can have an understanding that is more thorough than the one we had with the quiver plot, without having any problems regarding the dimensionality of the space (i.e. the number of players). Unfortunately, gaining an insight on how the system works on a macroscopic scale would require simulating the system for a large variety of initial condition which can have significant computational problems (especially on a high dimensional space).
- Create a dataset for the statistical analysis of the dynamical system. This step is done by solving the system for a wide variety of initial states and getting what the final states are. This way, we can have **a dataset with observations for the steady state of the system.** This dataset can help us make quite important conclusions for the dynamical system. Of course, this step also has computational problems, as the number of simulations we have to do in order to create the dataset scales in an asymptotically inconvenient matter ($O(m^n)$, where n is the number of players). Fortunately, the problem is embarrassingly parallel in its nature and a high performance algorithm which uses the power of a GPU can greatly reduce the time needed for computation. A successful creation of the dataset is one which gathers an adequate amount of observations for the system, which covers an adequate percentage of the possible initial states of the space, and which uses an adequately large time limit for the simulations (so that the system actually reaches the steady state). Alternatively, we can minimise the $|\dot{x}|^2$ function (the square is used for differentiability),

using a variant of the gradient descent algorithm that would satisfy the constraints of the problem (the states have to lie on the $[0,1]^N$ space). This can by computationally faster, but can only give us information about attractors that are equilibrium points (or a set of equilibrium points), as this method can only find points where the $\dot{x} = 0$ - we should not forget that the system might have limit cycles and/or other strange attractors that cannot be found using this optimisation technique.

- Use a density based clustering algorithm, in order to separate the dataset in a set of attractors. If the dataset is created successfully, then the observations should actually lie near the actual attractors of the dynamical system. A careful application of the density based algorithms, especially when done in a recursive manner, could separate the points based on the cluster they belong to. If the separation (i.e. clustering) is done successfully, then the dataset is partitioned in a way which can help us study every attractor on each own. This step is very useful, especially in the case where the visualisation of the state space is impossible.
- Use a classification algorithm which can help divide the state space in regions that correspond to different attraction basins. After labelling the steady states according to the attractor (cluster) they belong to, we can create a supervised learning dataset, where the independent variables are the initial states and the dependent variable is the attractor in which the system ends up to. If the dataset is large and able to represent the system dynamics in an adequate way, then a properly trained classification model could try to separate the state space in different attractor basins. This gives us a way to determine quickly and easily what the attractor of each initial state is.
- Use principal components analysis, or other more advanced techniques, to understand what the dimensionality of the attractors are. This means that we try to make a decision on whether the points of a cluster are on a surface, a curve, an equilibrium point etc. Again this step is only useful when the visualisation of the dataset is impossible.
- Try to model the attractors. After having observations of the attractors, we could try to use supervised learning techniques in order to have a mathematical model for them. If the dimensionality of the attractor is equal to the dimensionality of the state space and the attractor can be modeled using a function, then the problem can easily be solved using regression techniques. Of course, if these 2 statements do not hold true then the modelling of the attractor is much more difficult, but certain solutions could be applied to avoid the problem. For example, if the attractor is a curve which lies on a 3-D space, then we could try to use a model of the form:

  $$curve(u) = [function_1(u), function_2(u), function_3(u)],$$

  i.e. using a curve parameterization. The simplest way to continue would be to use a state variable x_i as the curve parameter and then make 2D scatter plots, x_i versus x_j for all the remaining j in order to see whether the $x_j = function(x_i)$ could be successful. Such a method is actually applied on my work later on.

After the abstract description, it is now time to apply the method in a simple, 2 Player game.

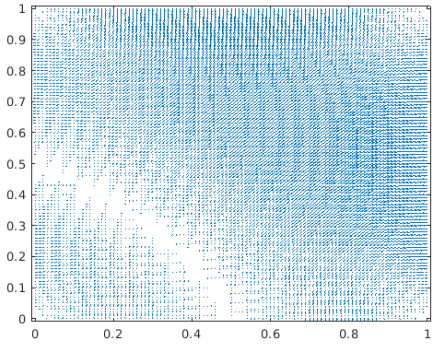
Results on the 2D system

In this section, I will study the dynamical system specified on the paper (check the readme), using $(a_{11}, a_{10}, a_{01}, a_{00}) = (-1, -1, -1, 1)$, so my system is

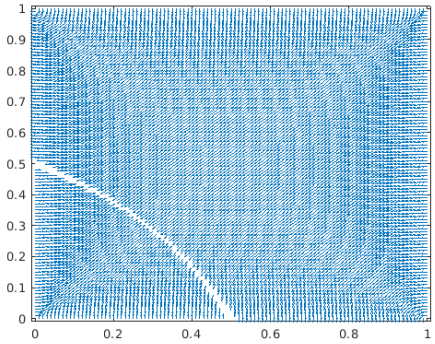
$$\frac{dx_1}{dt} = (-x_1x_2 - x_1(1-x_2) - (1-x_1)x_2 + (1-x_1)(1-x_2))x_1(1-x_1)$$
$$\frac{dx_2}{dt} = (-x_1x_2 - (1-x_1)x_2 - x_1(1-x_2) + (1-x_1)(1-x_2))x_2(1-x_2)$$

On the 2D Analysis directory, I have 4 .m files. The *system_ dynamics.m* file takes as input a specific x state and the 8 parameters of a 2D system (non symmetric) and returns the $\dot{x}$ value. The *simulate_ plots.m* file

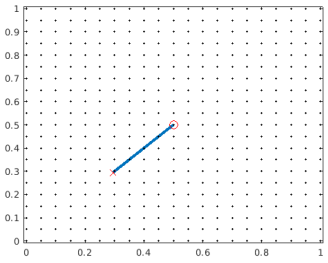
takes a specific initial state and solves the differential equation system numerically, it then creates time plots for every state variable and also a curve in the state space which shows the trajectory (acquired from numerical solving). The *create_phase_plots.m* implements the first stage of my methodology and creates the quiver plots (phase plots) of the system. Finally, in the *test.m* I define the parameters that the algorithm uses and call the 3 aforementioned functions in order to get my results. I highly advice you to use Matlab in order to run the scripts. These results can be seen on 1.



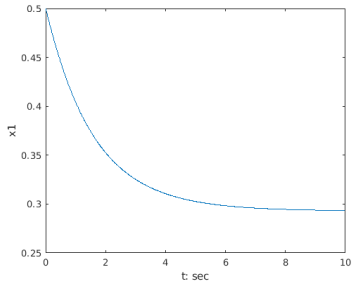
(a) Phase Plot, Unnormalised Vectors



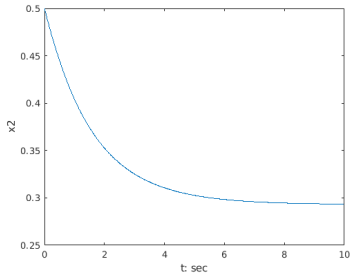
(b) Phase Plot, Normalised Vectors



(c) Simulation Results on the State Space



(d) x_1 vs t, simulation



(e) x_2 vs t, simulation

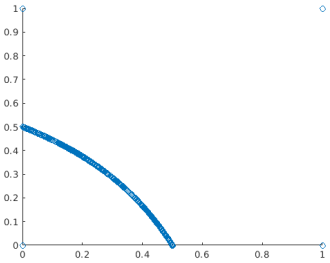
Figure 1: 2D Results

From the Phase plots, we can see that the system seems to have a curve that has the role of an attractor. From the simulation of the $x_{init} = [0.5, 0.5]$ point we can see that the trajectory ends up on a point that seems to belong in this attractor. This attractor is actually a set of equilibria, as the norm of the $\dot{x}$ that can be observed from the phase plots is zero on the attractor. The quiver direction is facing towards the attractor, which probably means that the equilibria are stable (if the directions were facing away from the attractor, then the equilibria would probably be unstable). By zooming in on the plot, we can see that the norm of the $\dot{x}$ is 0 on the four corners, which means that the 4 corners are equilibria (most probably unstable, since the quivers do not face towards these points).

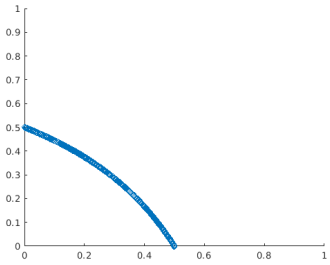
Going on, we can use the *2D Analysis/Stats* directory to apply the statistical analysis I have specified. In the *equilibria_study.m*, I create the dataset by using a grid on the 2D state space , with a space step of 0.01 between consecutive points on each one of the 2 coordinates. This gives us in total 10201 steady state observations. If the dataset is already created, we can use the *load_equilibria_flag* variable to avoid the unneeded computation and just load the dataset we have already saved. Since we are on 2 Dimensions, the first thing I do after creating the dataset is scatter plots, to see what the results were. These results can be viewed on 2.

As we can see, we have the equilibria on the corners and the ones on the curve, which is something we already expected from the phase plot. The next thing I decided to do is model the curve, as in this specific case it seems like an easy task.

At first, I made some simple hypothesis regarding the curve. The first is that the curve is an arc of the



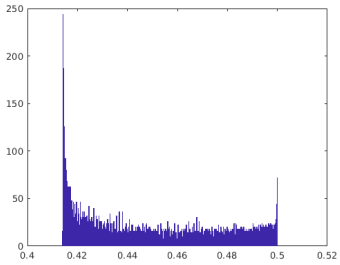
(a) All of the equilibria



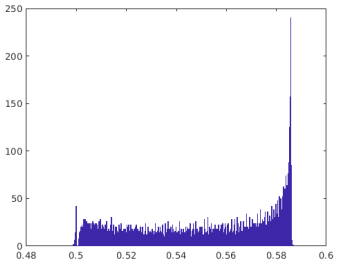
(b) The equilibria on the curve (got rid of the corners)

Figure 2: Scatter plots of the steady state observations

circle with a $[0, 0]$ as a center, and the second is that the curve is on the locus of points where the sum $x(1) + x(2) = constant$, where $x(1), x(2)$ are the two coordinates of the state vector. I decided to test on whether these two hypothesis are true by creating a histogram of the norm of the steady state observations (for the first one), and one with the sum $x(1) + x(2)$ for every state. The two histograms can be seen on figure 3.



(a) Histogram of the $\dot{x}$ for the points of the curve



(b) Histogram of the $x(1) + x(2)$ for the points of the curve

Figure 3: The two histograms that will help us check whether the hypotheses were true

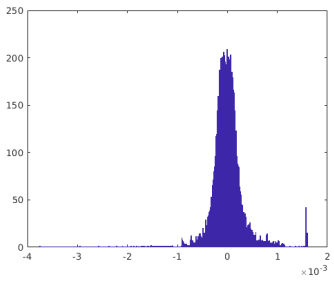
As we see, the histograms do not seem to indicate a distribution with zero variance, which will lead me to reject the hypothesis. To be fair, it was quite obvious that the 2 hypotheses were not true and the reason why I present this way of thought is to show that my methodology could allow the researcher to do some (informal) statistical hypothesis testing if needed.

Going on, I will make a third hypothesis - one where the premise actually seems much more plausible. I continue by assuming that the curve we are studying could be modelled using a $4-$th degree polynomial, one of the form $x_2 = regress(x_1)$. On my script, I apply the least squares solution after augmenting the dataset so that it can handle the polynomial regression. On 4 we can see the results.

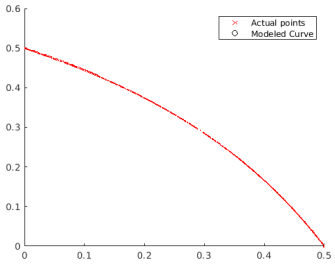
As we can see, the regression is quite successful and the modelling of the curve is near perfect. This means that we managed to model the curve successfully using just numerical and statistical methods. I could further decrease the error metrics by increasing the polynomial degree, but I believe that making the polynomial more complex is not worth once it already achieves remarkable results.

In order to solve the classification problem, we first have to solve the clustering one. Unfortunately, Matlab has no implementation of the DBSCAN/OPTICS algorithms, which forced me to use python and the sklearn package (which is open source) to solve this problem. I provide you with an *interactive python (.ipynb)* file, which already states the results of the script so that you do not have to actually run it. You just need to have an IDE to open the .ipynb file (or use google colab, jupyter etc).

This script opens the dataset, applies the dbscan algorithm and shows the clusters that are extracted. It also



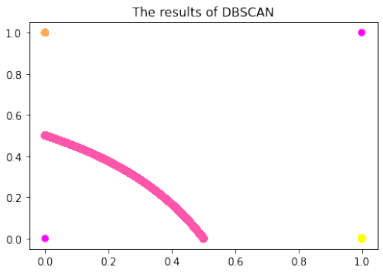
(a) Histogram of the regression residuals (errors)



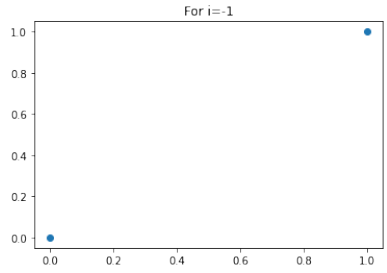
(b) Plotting the curve and the observed steady states on the same scatter plot.

Figure 4: Results of the regression, MSE=0.000000 Rmse=0.000315, polynomial parameters [0.5002, -0.4875, -0.7093, 0.5490, -2.3460]

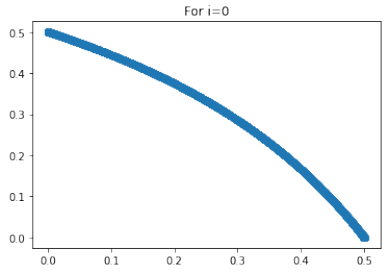
saves the labels in a csv file. I follow a fairly simple approach (I completely avoid OPTICS), as the game is on a fairly low dimension and I can easily visualise the results. This is also the way I chose the hyperparameters, I set the $min\_samples = 4$ because it is a fairly common heuristic ($2 * \text{num\_features}$) and then chose $eps = 0.01$ because this seems to give decent results. On 5 we can observe the dbscan results.



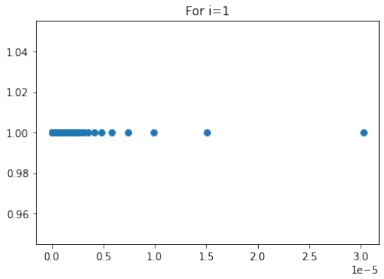
(a) All of the clusters



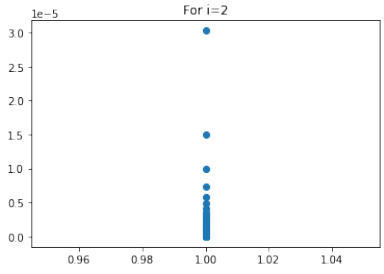
(b) Points classified as noise (label=-1)



(c) The first cluster (label=0).



(d) The second cluster (label=1)



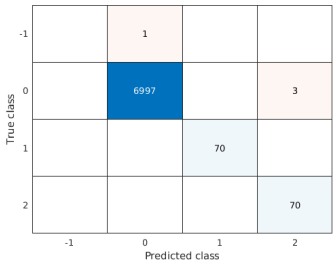
(e) The third cluster (label=2)

Figure 5: DBSCAN results

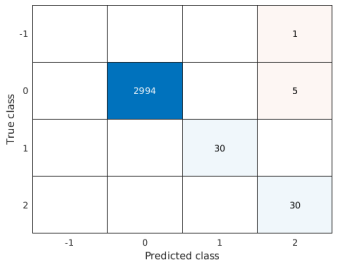
As we can see the algorithm did quite a good job. The lower left and upper right corners are classified as noise (this means the corresponding points do not have enough neighbours to actually create their own cluster), while the other two are classified as separate clusters. This is mainly due to numerical reasons - we see that there are some points in the neighbourhoods of the equilibria that have not managed to actually reach the equilibrium, either due to a fairly short simulation time limit or numerical errors. The important part, which is the separation of the curve from the rest of the points, is done successfully.

The last step that we can implement using the statistical methodology would be to create a classifier to help us study the basins of attractions. To do this, I use my dataset (with the labels I acquired from the *games.ipynb*) on the *equilibria_study.m* and run the *classify* function (can be found on the same file). This function separates

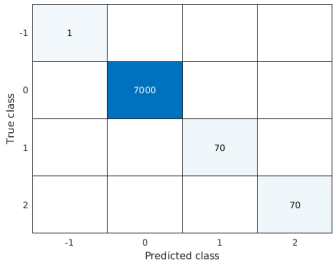
the dataset into two separate parts, a train and a test set. The separation is done using stratification, this way we make sure that the class distribution remains quite the same on the 3 datasets (original, train, test). I use both a linear and a quadratic multinomial logistic regression. The results can be viewed on 6



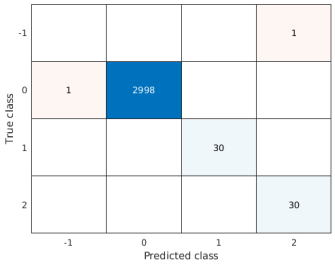
(a) Linear Model, Train set, Acc=0.99944



(b) Linear Model, Test set, Acc=0.998



(c) Quadratic Model, Train set, Acc=1



(d) Quadratic Model, Test set, Acc=0.99934

Figure 6: Classification Results

As I see, both of the models did a fairly good job. The results on the train set are quite equivalent to the ones on the test set, this is due to the fact that the 2 datasets are actually created using the same mechanism and therefore the empirical distributions do not differ greatly. Also I see that the non linear model is more accurate than the linear one, which is obviously due to the fact that it boasts a higher capacity.

As we can see, the statistical methodology that we proposed could actually give us some interesting results. Of course, in this specific case the dynamics are fairly simple and the whole numerical analysis might seem unnecessary. Later on, I apply the methodology on a 3D system, where the theoretical analysis is harder to be executed.

Numerical Analysis of a 3D system

In this section I decided to use my statistical methodology to study a 3 Player system. First I define the dynamics of the said system as:

$$\dot{x}_1 = [a_{000}(1-x_1)(1-x_2)(1-x_3) + a_{001}(1-x_1)(1-x_2)x_3 + a_{010}(1-x_1)x_2(1-x_3) + a_{011}(1-x_1)x_2x_3$$

$$... + a_{100}x_1(1-x_2)(1-x_3) + a_{101}x_1(1-x_2)x_3 + a_{110}x_1x_2(1-x_3) + a_{111}x_1x_2x_3]x_1(1-x_1)$$

$$\dot{x}_2 = [b_{000}(1-x_1)(1-x_2)(1-x_3) + b_{001}(1-x_1)(1-x_2)x_3 + b_{010}(1-x_1)x_2(1-x_3) + b_{011}(1-x_1)x_2x_3$$

$$... + b_{100}x_1(1-x_2)(1-x_3) + b_{101}x_1(1-x_2)x_3 + b_{110}x_1x_2(1-x_3) + b_{111}x_1x_2x_3]x_2(1-x_2)$$

$$\dot{x}_3 = [c_{000}(1-x_1)(1-x_2)(1-x_3) + c_{001}(1-x_1)(1-x_2)x_3 + c_{010}(1-x_1)x_2(1-x_3) + c_{011}(1-x_1)x_2x_3$$

$$... + c_{100}x_1(1-x_2)(1-x_3) + c_{101}x_1(1-x_2)x_3 + c_{110}x_1x_2(1-x_3) + c_{111}x_1x_2x_3]x_3(1-x_3)$$

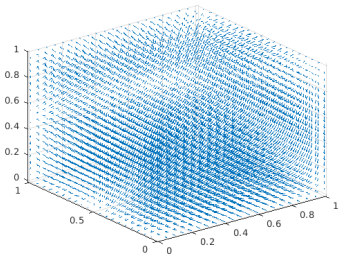
I used the *rand* function of matlab to get values $(1, -1)$ for the parameters above (if the result is > 0.5 then we set it $= 1$, else $= -1$). The parameters that I got are

$$a = [-1, -1, -1, 1, 1, 1, -1, 1]$$

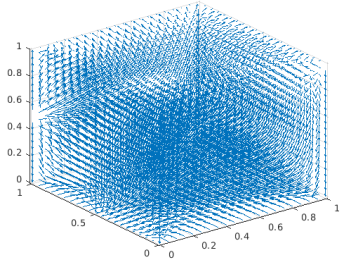
$$b = [1, -1, 1, 1, -1, 1, -1, 1]$$

$$c = [-1, -1, -1, -1, -1, 1, 1, -1]$$

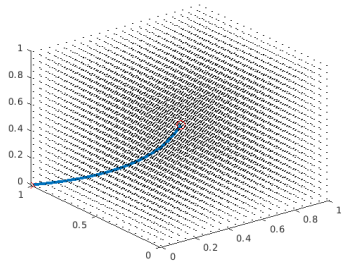
Going on the *3D Analysis* directory we can see the *test3D.m*, *create_phase_plots3D.m*, *simulate_plots3D.m*, *system_dynamics3D.m* which are all generalisations of the ones we had on the 2D analysis too (phase plots, simulation). The results can be seen on 7.



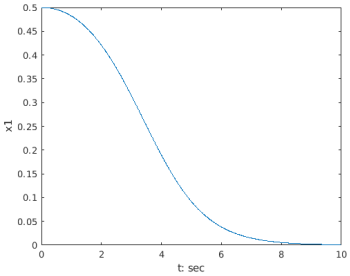
(a) Phase Portait, unnormalised vectors



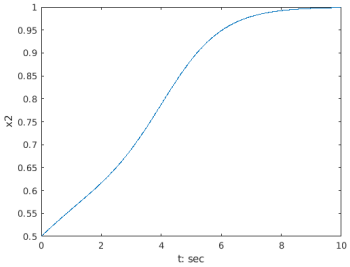
(b) Phase Portait, normalised vectors



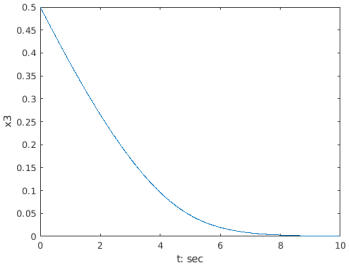
(c) Simulation, results on the 3d space



(d) Simulation (x_1 vs t)



(e) Simulation (x_2 vs t)



(f) Simulation (x_3 vs t)

Figure 7: 3D Phase Portait, simulations with [0.5, 0.5, 0.5] as initial condition

As we see, the phase portait is not that easily interpretable (as the one on the 2D problem). I will continue with the statistical analysis, but I will pretend that I cannot visualise the results - this is done so that I can test how my methodology could be applied in a high dimensional space where visualisation is practically impossible. In the end, I will present the plots so that we can see how well my results are.

Going on with the *3D Analysis/Stats* scripts, we can see that the scripts are generalisations of the ones we had at the 2D case. To create a dataset I understood that I had to apply some different method to get the initial states, because if I just used a 3D grid then the algorithm would be too complex, both in terms of time and memory. I decided to use 2D grids on the 6 borders of the $[0, 1]$ cube and then just acquire a large amount of initial states from a uniform distribution on the interior of the cube. The reason why I did this is because intuitively the border of the cube seems to have important information and therefore it has to be represented in a more thorough way than the interior. The number of interior points is 300.000 which gives us a total of 361206

points. The process of the dataset creation takes several hours, but I believe that it can be greatly reduced if one could use a massively parallel implementation of the Runge Kutta Methods.

The next thing I did, after creating the dataset, was to apply the OPTICS algorithm. Again I use the sklearn package of python, and I present the results in the *games.ipynb* file so that you do not have to run the script (it takes several hours too). The reachability plot I get from the OPTICS algorithm can be seen on 8.

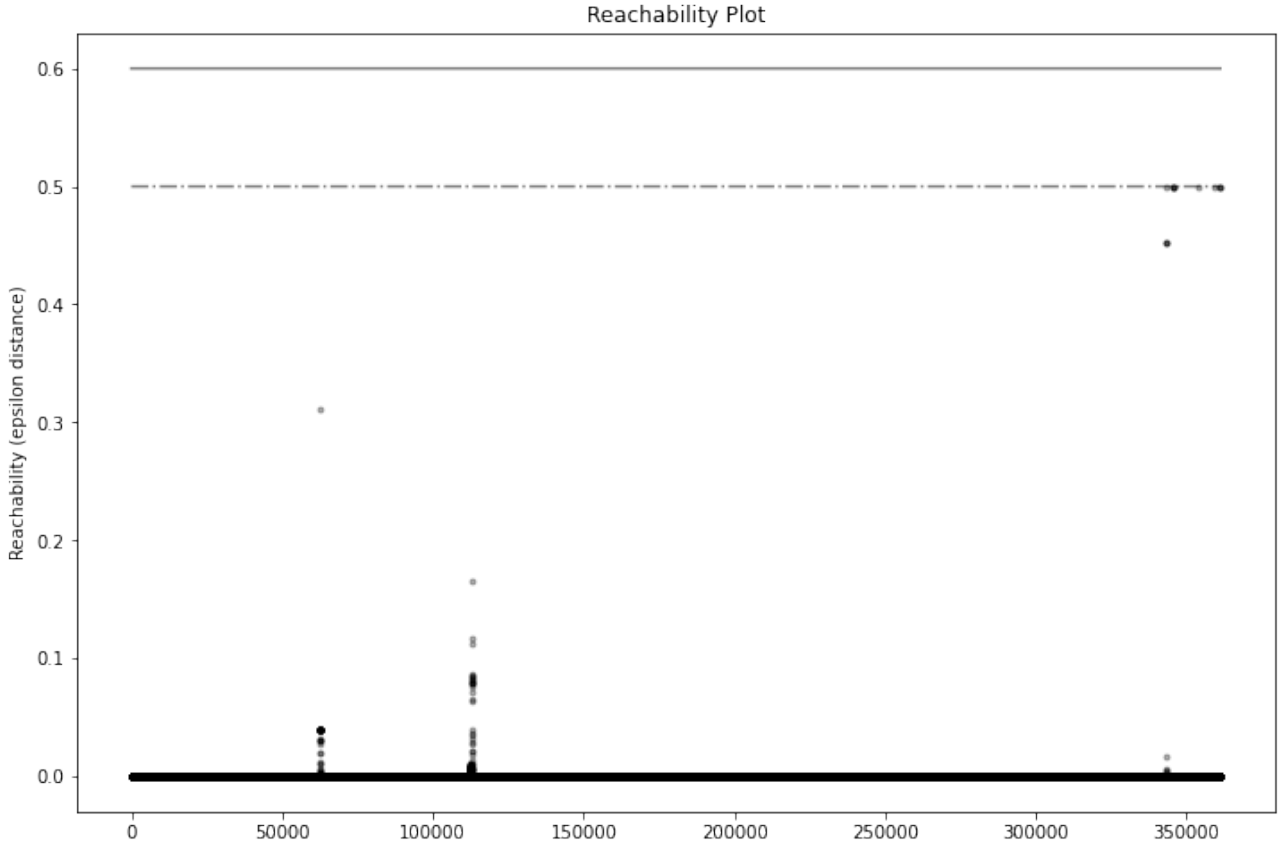
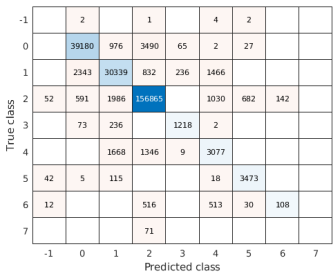


Figure 8: Reachability Plot, Optics

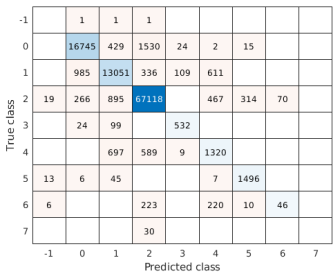
In this plot, the points that generally belong to the same cluster have a small reachability distance to their nearest neighbor and generally the clusters tend to show up as valleys on the plot. What we want is to choose an eps value (horizontal line) so that the points below form deep valleys. I begin with a simple application of the OPTICS to get the reachability plot, and then using the DBSCAN ($min\_samples$=8, eps = the one I choose)to get the clusters (i.e. the optics only helps in choosing the eps parameter). In the more general case, I could use the reachability plot to detect a larger amount of deep valleys and perform the clustering algorithm recursively - an experienced data analyst could work with this dataset and find the clusters with better results. The clustering results will be revealed in the end.

I return on the *equilibria_study.m* file and apply the classification analysis (this is done to help me get an easy way to study the basins of attraction). Again I use both a linear and a quadratic multinomial logistic regression model. The training phase requires more than half an hour, if you do not want to train the model (and run the classification algorithm) set the *classification_analysis=0*.

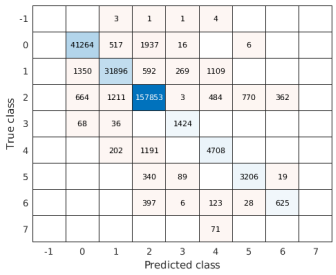
As we see from the classification analysis 9, the results are worse than the ones we had previously on the 2D case, and here the superiority of the non linear model is more observable. I believe that the classification results could be further improved if we had a larger dataset and/or used a Deep Multilayer Perceptron instead of the polynomial multinomial logistic regression. Using a higher degree polynomial would increase the training time, which might not be worth it.



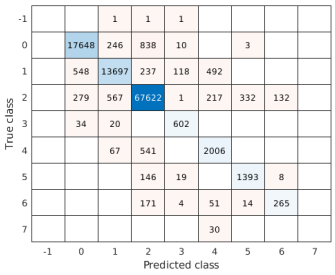
(a) Train Set, Linear Model, Acc=0.9264



(b) Test Set, Linear Model, Acc=0.9256



(c) Train Set, Quadratic Model, Acc=0.953



(d) Test Set, Quadratic Model, Acc=0.9526

Figure 9: 3D Classification Results

The step I try to do before going on with the curve modelling is to apply PCA in order to understand the dimensionality of the attractors. The results from the PCA can be seen on the matlab terminal, and 10.

The points with label==-1 are classified as noise by OPTICS.

As we see, in most of the clusters, the variance is less than 10^{-6} . But in the label==1 the variance is clearly larger than the others. This leads me to the conclusion that all of the clusters, except the one with label==1, must represent points. This means that the points are either equilibria, or points that due to the computational nature of the network could not reach their steady state. The cluster with label==1 should be either a surface or a curve. Looking at the percentage of variances explained by each variable on the PC analysis,I see that the first variance on the PCA analysis is clearly larger than the other two. This leads me to the final conclusion, that according to the PC analysis the cluster with label==1 might be a curve! I therefore make the hypothesis that this cluster represents an attractor that is a curve, and all the others are point equilibria. I continue by trying to model this specific curve.

To do this, I will try to model the curve using parameterisation (as I have described earlier). I can make 2D scatter plots between the various pairs of x_i variables in order to check and choose which pairs could be used in order to create the parameterisation of the curve. Let me remind you that our goal is to simply find a description:

$$r(u) = [fun_1(u), fun_2(u), fun_3(u)]$$

I will try to use a x_i as the parameter, and the other two functions will be found using regression analytics.

The 2D Scatter plots between the x_i, x_j pairs can be viewed on 11.

By looking at the scatter plots, I see that the $x_3 = fun(x_1)$ would be a good choice, as a polynomial should be able to describe this line fairly easily. However, in the other two cases we might have serious problems, as we see that there is a vertical line (which means that the relationship between the two variables is not functional). I will make a test by taking the x_1 variable to be the parameter (independent variable) and try to use regression

```
For label=0:

Perc Variance Explained:
   94.0395
    5.9602
    0.0003

Variances:
   1.0e-08 *

    0.6260
    0.0397
    0.0000

For label=1:

Perc Variance Explained:
   97.8065
    2.1881
    0.0054

Variances:
    0.0092
    0.0002
    0.0000
```

(a) Labels 0,1

```
For label=2:

Perc Variance Explained:
   95.9435
    4.0560
    0.0005

Variances:
   1.0e-07 *

    0.2580
    0.0109
    0.0000

For label=3:

Perc Variance Explained:
   78.3620
   21.6380
         0

Variances:
   1.0e-07 *

    0.4725
    0.1305
         0
```

(b) Labels 2,3

```
For label=4:

Perc Variance Explained:
   50.7060
   49.2940
         0

Variances:
   1.0e-07 *

    0.1718
    0.1670
         0

For label=5:

Perc Variance Explained:
   64.1977
   35.8023
    0.0000

Variances:
   1.0e-13 *

    0.4049
    0.2258
    0.0000
```

(c) Labels 4,5

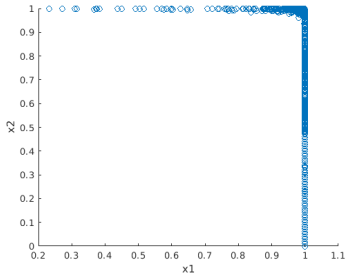
```
For label=6:

Perc Variance Explained:
   67.0381
   32.9619
         0

Variances:
   1.0e-13 *

    0.4098
    0.2015
         0

For label=7:

Perc Variance Explained:
   100
     0
     0

Variances:
   1.0e-07 *

    0.3713
         0
         0
```

(d) Labels 6,7

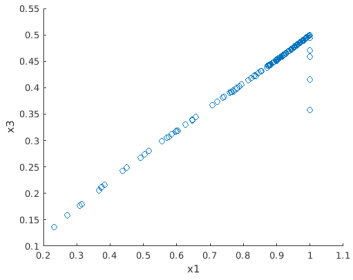
Figure 10: Results from the PCA

analytics to find the other two functions. In 12 we can see some results.

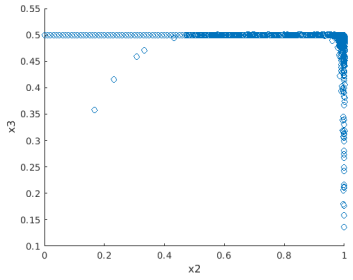
As we see, the polynomial regression cannot be used to describe this curve, because the $x_3 = fun(x_2)$ has a vertical line. The best thing that I could think of is to just separate the curve into two different subcurves, so that I can then apply regression analytics to find models. My calculus is described below.



(a) x_2 vs x_1

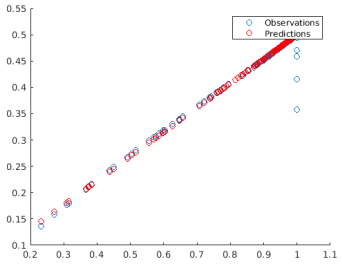


(b) x_3 vs x_1

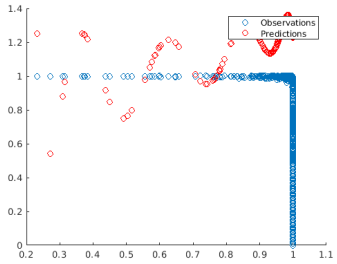


(c) x_3 vs x_2

Figure 11: The 2D scatter plots



(a) x_3 vs x_1 linear regression coeffs ->(0.0375, 0.4626)



(b) x_3 vs x_2 polynomial regression (degree=20)

Figure 12: The regression results.

If the $x_1 = 1$ (computationally *if* $x_1 > 0.99$), then we have the r_1 curve, which is modelled by

$$r_1(x_2) = [1, x_2, 0.5], x_2 \in (0, 1)$$

The $x_3(x_2) = 0.5$ was found using the x_3, x_2 scatter plot.

If the $x_1 \neq 1$ (computationally *if* $x_1 < 0.99$) then we have

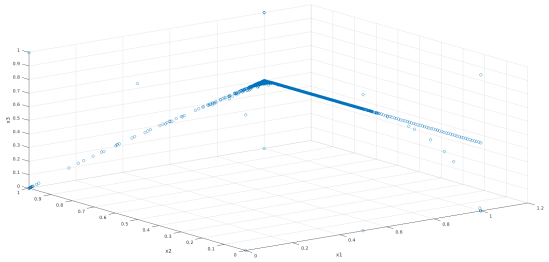
$$r_2(x_1) = [x_1, 1, regression(x_1)], x_1 \in (0.23, 1)$$

Where the $x_3 = regression(x_1)$ is already calculated as $0.0375 + 0.4626 * x_1$. The bounds of the $x_1$variable are this way because these are the minimun/ maximum values that had been observed. This is the final model of our only curve. It is now time to reveal the 3D nature of our data 13.

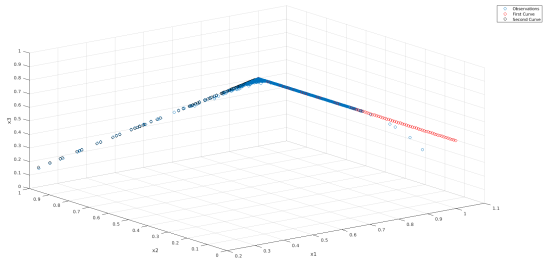
As we see, the method managed to go quite well in describing the curves and classifying the unknown data points, even though it is clear that further improvement could be achieved. In this specific case, I believe that the total number of equilibria calculated might have been inadequate, as we see that the left part of the curve cannot be considered continuous. Also, there is a large amount of points whose position makes no sense as steady states, which might imply that I should have run the simulation algorithms for a longer time frame. Of course, these two increase the time needed for computation, which as we have already described could be reduced by using massive parallelism.

The classification algorithm could be improved by further dividing all of the noise points into separate clusters (which makes sense, because they should belong in different classes) and also by training a non linear model with higher capacity. I used the quadratic polynomial just to demonstate that a non linear model can do better than a linear one.

The curve modelling problem is the hardest one in the whole methodology. We might need to use more



(a) The equilibria calculated



(b) The curves that are described from the models plotted with the equilibria

Figure 13: The 3D results. You can see the results of the OPTICS algorithm in the .ipynb file, we do not show them on the paper because it already is too large.

advanced methods to estimate the dimensionality of the attractors (Grassberger and Procaccia have proposed an algorithm for the estimation of the correlation dimension of attractors , used mainly in the field of time series analysis based on the Taken's embedding theorem - we might be able to apply some variant of the algorithm for our situation), and maybe even methods to model the curves in a more reliable and generalisable way.

In spite of all these difficulties I believe that my methodology can actually prove to be helpful, especially when studying dynamical systems that are hard to understand analytically.