Report for exercise 5 from group F

Tasks addressed:	5
Authors:	Yuxuan Wang (03767260)
	Shihong Zhang (03764740))
	Mengshuo Li (03792428)
	Augustin Gaspard Camille Curinier (03784531)
	Qingyu Wang (03792094)
Last compiled:	2024–06–24

The work on tasks was divided in the following way:

Yuxuan Wang (03767260)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Shihong Zhang (03764740))	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Mengshuo Li (03792428)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Augustin Gaspard Camille Curinier (03784531)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Qingyu Wang (03792094)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%

Report on task 1, Approximating functions

In this task, we implemented both linear and nonlinear functions approximations. To approximate a linear function, we simply had to perform a linear regression (Ordinary Least Squares problem).
We define $\hat{f}(x) = Ax$ for some matrix $A \in \mathbf{R}^{d \times n}$. Here, $\hat{f}$ will be our approximate of the true underlying function F mapping our data X to some other vectors Y. Therefore we wanted to find the matrix A that maximizes the similarity between $\hat{f}$ and f or in other words minimizes loss $e(\hat{f}) = \|F - f(\hat{X})\|$ that is find A such that $A^T = (X^T X)^{-1} X^T F$. We solve this problem using *scipy.linalg.lstsq* in python.

For the nonlinear approximation, we used the linear decomposition into radial basis functions. We define $\hat{f}(X) = C\phi(X)$ with ϕ a vector of L radial basis functions whose elements are $\phi_l(x) = \exp -\|x_l - x\|^2/\epsilon^2$ with x_l called centers, uniformly spaced samples on the domain where we want to realize the approximation. Then, we simply perform linear regression to find the best coefficient matrix C since $\hat{f}$ is a linear combination of radial basis functions.

For epsilon, we will take a non-squared value as it was the case in the task on Diffusion Maps.

Part 1: Approximate the function in dataset (A) with a linear function Here, we are going to approximate function in dataset (A) with linear function. First, we plot a graph of how the data looks like:

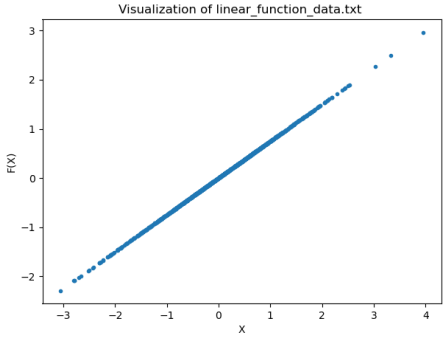

Figure 1: Function F in Dataset (A)

Then we fit a linear model on the data $(X, F(X)$ thus learning an estimator of A. We then transform X, obtaining estimate points $\hat{f}(X) = AX$. All of this is implemented with our function "linear fit transform()" that returns a tuple made of the transformed data and the coefficients matrix A. Here is the result of our approximation:

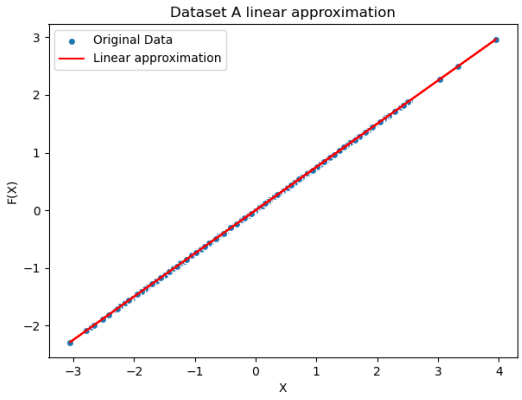

Figure 2: Linear approximation for Dataset (A)

As it can be seen from the graph (*Figure 2*), our linear approximation is accurate and fits the observed data almost perfectly. We find an MSE of 10^{-10}, which is very low. This confirms the intuition one can have looking at *Figure 1* that F is a linear function.

Part 2: Approximate the function in dataset (B) with a linear function Now, we will be performing the exact same approximation with Dataset (B). Let's first take a look at what the data looks like:

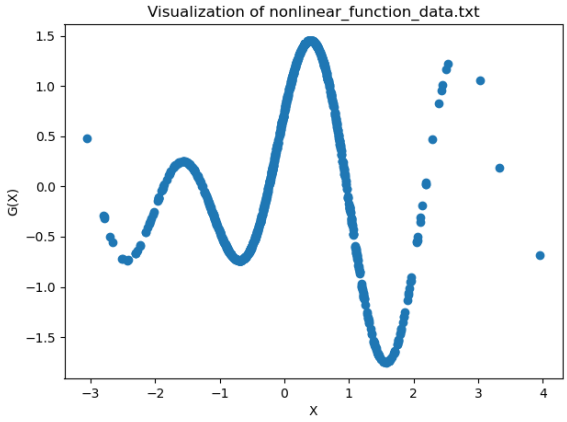

Figure 3: Function G in dataset (B)

The function does not look linear at all. We plot the result of the linear approximation:

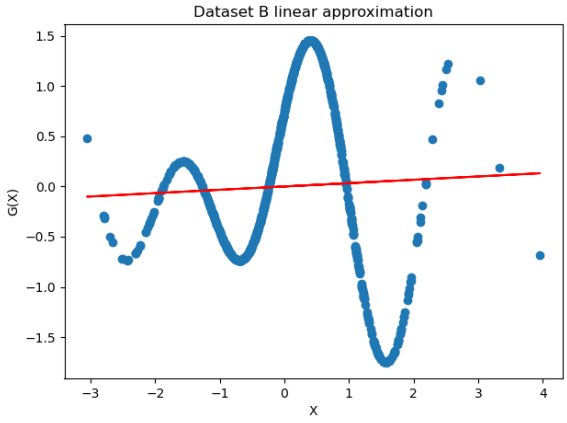

Figure 4: Linear approximation for Dataset (B)

As expected, this renders a very poor approximation of function G, with an MSE of 0.77 10 times higher than linear approximation of F. Linear model is not a good fit for approximating this function, we will see now if our RBF model fares better.

Part 3.1: Approximate the function in dataset (B) with a combination of radial functions After realizing that G is not a linear function, we will approximate it using radial basis functions linear decomposition. To do so we built a function "non linear fit transform()" that returns the transformed data as well as the matrix C of coefficients.

We choose parameter $L = 50$ and $\epsilon = 0.05$:

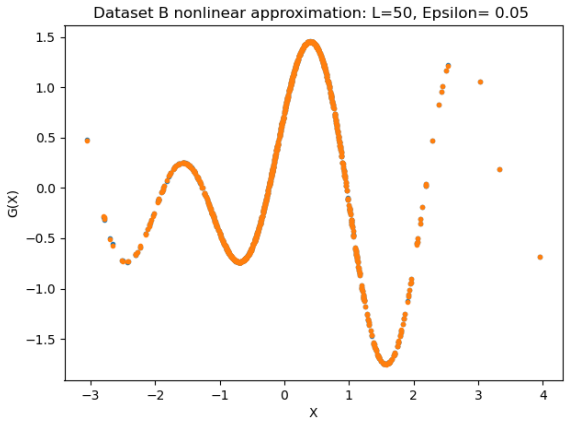

Figure 5: RBF approximation for Dataset (B)

This time, our approximation fits the data way better. We find a MSE of $1.2 * 10^{-6}$, the error is significantly lower than our linear approximation. Since we are humans, we may call our model reality and deem the function G in dataset (B) a nonlinear function that can be sucessfully approximated with a linear decomposition of radial basis functions.

We chose L = 50 because it was the smallest number giving a satisfying fit to the curve. With a smaller value, the approximation would lack the complexity needed to follow the shape of the function, made of 6 local extrema. We have 1000 data points, to approximate those 6 extrema we need enough centers uniformly distributed on the domain to approximate the function around them. With 50 centers, we have roughly 8 centers for each local extremum, which is a sufficient ratio. The number of data points has also to be taken into acccount here, we have 1 center for 20 data points which also seems like a reasonable proportion. Choosing a larger L is simply unnecessary as it will increase the number of basis function to compute and make our model more complex.

After we have our value for L, we can think about epsilon i.e the bandwidth of our basis functions. We have for reference the value chosen in diffusion maps $\epsilon = 0.05$, and it fits well in this case. Larger bandwidth will better approximate smooth 'flat' functions, that is not the case here. Function G hase a lot of turns and change of directions; which is why we need a relatively small bandwidth to capture that behavior.

Part 3.2: Approximating function in dataset (A) with radial basis functions (Why is it not a good idea?) Here we approximated function F in dataset (A) with RBF linear decomposition. First off before even seeing the result, we can say it is a bad idea because it induces unnecessary complexity in our model, in fact after a first glance at the data we can already assume that the function is linear of the form $\hat{f}(x) = ax$ (1 dimensional case) leaving us with a simple least squares problems. Using RBF approximation here is simply a waist of time and resources, since it implies tuning the parameters L and epsilon, then computing the L basis function and then eventually come back to our original least squares problem. This approach is highly unefficient and unneccessary. With that being said, here are the results of our nonlinear approximation of F:

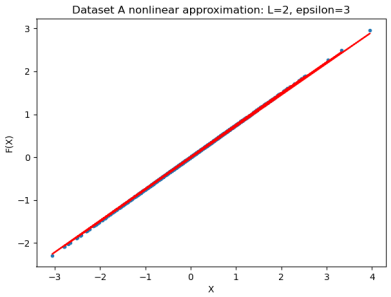

Figure 6: RBF approximation for Dataset (A)

Surprisingly, we are left with a quite precise approximation of linear function F; however we can see that it does not fit the data as good as our first linear model. Moreover, we find an error of $2 * 10^{-5}$ which is 10^5 larger than the linear model error: our linear model is a better approximation of reality than RBF model.

Report on task 2, Approximating linear vector fields

Part 1: Estimation of the vector field In this first part, we estimated the vector field v that generated the points x_1 from the points x_0. We used the finite-diffence formula to compute the vector field:

with $\Delta_t = 0.1$

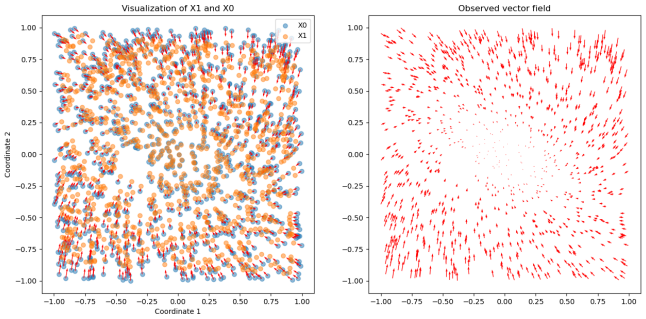

Figure 7: Visualization of linear vector field

Once with have our vector v, we fit a linear model on it and transform x_0, thus getting $\hat{v} = Ax_0$ a linear approximation of vector field v. Here are the plots comparing the true vector field and our approximation:

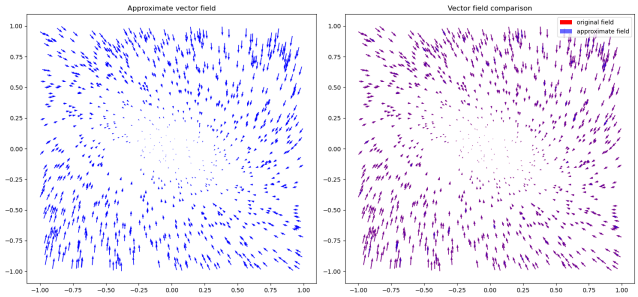

Figure 8: Linear vector field approximation

We found an MSE of 10^{-14} which is extremely low, we can thus say that our linear model fits the vector field perfectly.

Part 2: Solve the linear system and compute the mean squared error Now that we have our estimate of the vector field, which is nothing but the time derivative of the evolution operator of the system:

We used the method solve_ivp to integrate the vector field over $\Delta_t = 0.1$ thus getting an estimate of the points x_1.

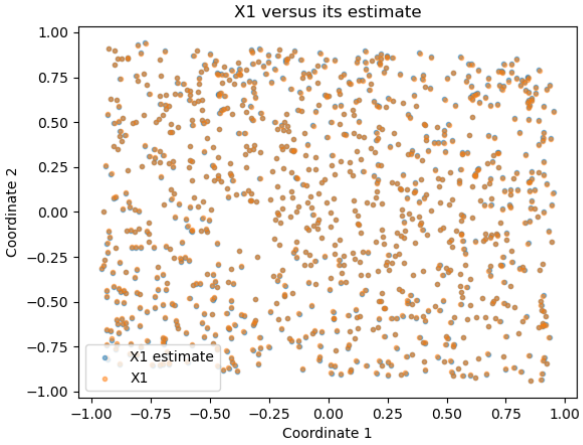

Figure 9: Estimation of x_1 points

As we can see from the graph, we realized an accurate estimation of points in x_1 by integrating our linear equation over time. We found an MSE of 10^{-5}.

Part 3: Visualize trajectory for $x_0 = (10, 10)$ as well as phase portrait

Finally, we replicate what we did for part 2: we use solve_ivp to integrate the linear equation over time, however this time we choose a unique initial point $x_0 = (10, 10)$ instead of a whole vector of initial points, and we integrate until $T_{end} = 100$. Notice that here x_0 lies far outside the training data whose domain is $[-1, 1]^2$ which will be a good indicator to test if our model can generalize properly and has not been overfitting.

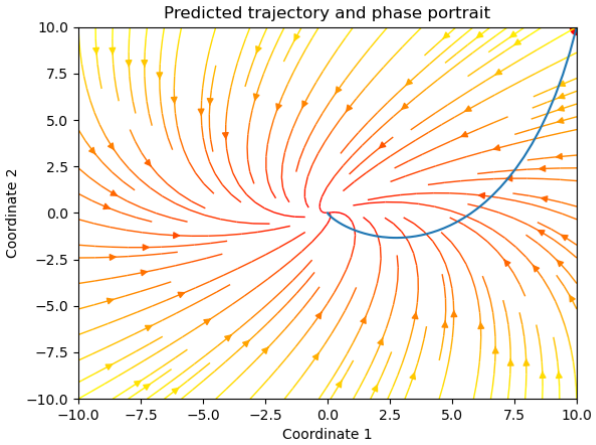

Figure 10: Trajectory and phase portrait for linear vector field approximation

Our trajectory and phase portrait are coherent with each other as well as what could be anticipated from training data; trajectory leeds the point towards the center of the plane, following the vectors displayed by the phase portrait, this could also bee seen in part 1 with vector field v pointing towards the center. With a linear field model, it is not surprising to find that our system has one unique steady state at the origin.

Report on task 3, Approximating nonlinear vector field

Part 1: Estimate the vector field with a linear operator and compute mean squared error to the solution after Δ_t For this first part, we reproduced the process we went through during part 1&2 of task 2. After computing the vector field v with finite-difference formula, we approximated this vector field as a linear function before integrating the vector field estimate over time up to $\Delta_t = 0.01$ giving us an estimate of the points in x_1 vector.

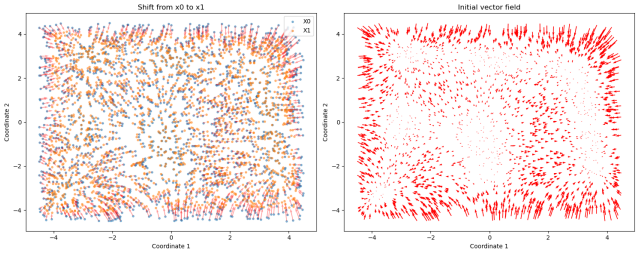

Figure 11: Visualization of nonlinear vector field

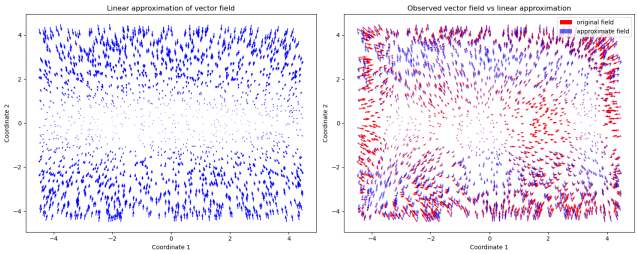

Figure 12: Linear approximation of nonlinear vector field

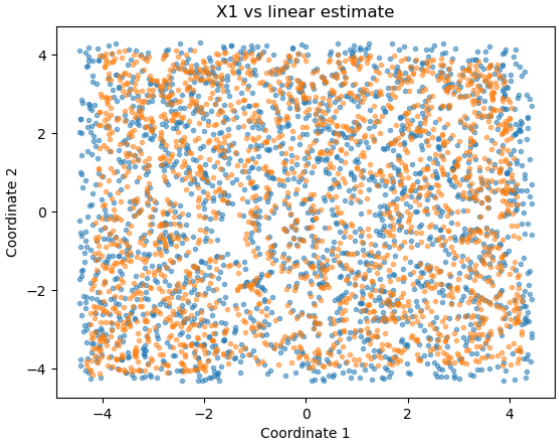

Figure 13: Estimation of x_1 points, linear approximation

As expected, our estimation of x_1 is not satisfying: our estimates do not overlap with the original points (*Figure 13*) and we computed an MSE of $3.7 * 10^{-2}$ which is quite high. Both visually and numerically we found that this linear approximation was not accurate and was not a good representation of the underlying dynamics of the system.

Part 2: Approximate the vector field with RBF and compute mean squared error to the solution after Δ_t After finding that the linear approximation was not a good fit, we implemented the RBF approximation for the vector field v. Same steps as before: use v as our target for nonlinear regression with our *nonlinear_fit_transform()* method, giving us a nonlinear estimate $\hat{v}$. Then we solved the differential system by integrating over time up until Δ_t.

We used $L = 100$ and $\epsilon = 0.1$: higher values of L did not show a significant improvement in the MSE. For task 1, we chose 50 centers for 1000 points; now with 2000 points to approximate we kept the same ratio and chose 100 centers. About ϵ, we had 0.05 as a reference for task 1. However, this value gave an MSE of 10^{-2} not significantly better than linear approximation. We noticed that by increasing the bandwidth, we had a smaller

error, thus settling wiht 0.1. The vector field being a smooth function, it makes sense to have a higher value for ϵ.

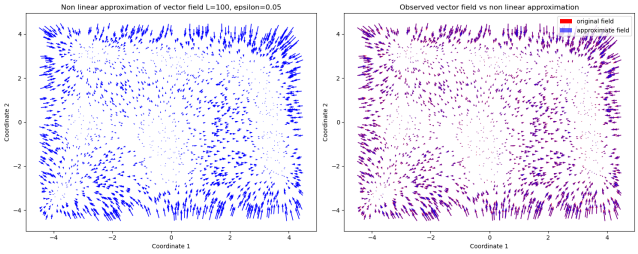

Figure 14: RBF approximation of nonlinear vector field

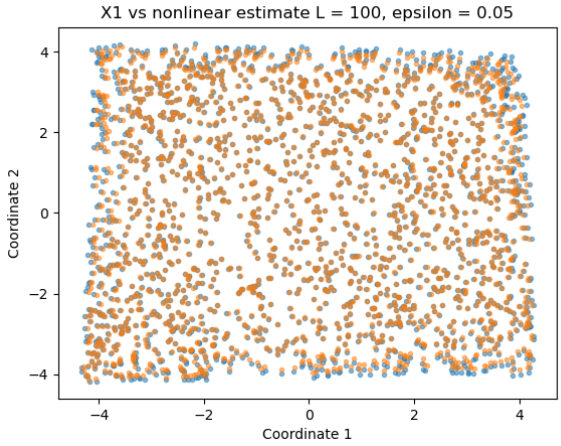

Figure 15: Estimation of x_1 points, RBF approximation

After plotting our new estimate of x_1 points as well as computing the error, we can conclude that our RBF approximation is better than the linear approximation. We get an MSE of $1.3 * 10^{-3}$ more than 10 times smaller than for linear model.

Part 3: Analysis of the system To conclude this task, we performed an analysis of the dynamical system by solving the differential equation for a larger time. We chose $T_{end} = 10$ i.e $1000 * \Delta_T$, naturally keeping L and ϵ unchanged.

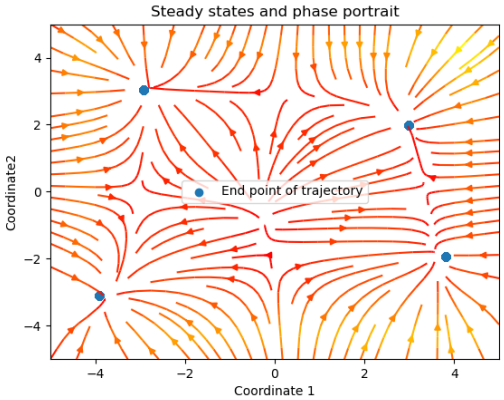

Figure 16: Steady states and phase portrait

We can identify 4 steady states, roughly situated near each corner of the 2D space. This system is not topologically equivalent to a linear system, since a linear system has either 1 or an infinity of steady states.

Report on task 4, Time-delay embedding

Part 1: Embedding of a periodic signal Here, we appplied Takens theorem to a 1 dimensional periodic manifold embedded in a 2D space.

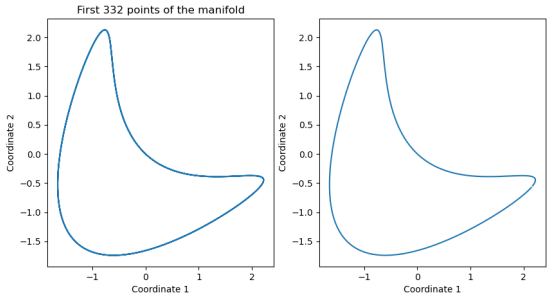

Figure 17: Visualization of 1D periodic manifold

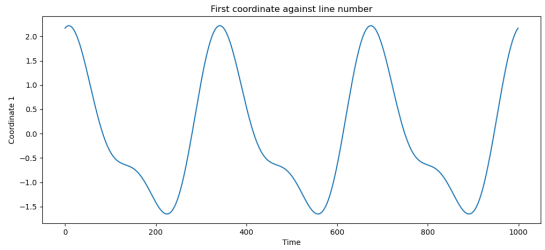

Figure 18: First coordinate over time

We can clearly see from the 2 graphs above the periodicity of our manifold (with a period of 333 timesteps), and the fact that it is a one dimensional curve. According to Takens Theorem, we need to plot $2d + 1 = 3$ (since $d = 1$) delayed coordinates to make sure that the manifold is embedded properly. We chose a delay of $\Delta_n = 100$ rows, and plotted the first coordinate against its delayed version:

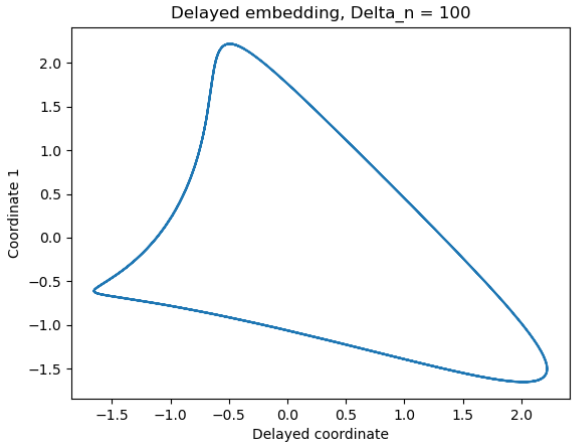

Figure 19: 2-dimensional time-delay embedding

Even though we used $2 < 3$ coordinates, we can observe that our embedding offers a faithful representation of the original manifold, preserving the essential topological properties: 1-dimensional, periodic and smooth. Takens in fact proposes an upper bound, it is possible to find lower dimensional emebeddings, however it is not a general property.

Part 2: Approximating chaotic dynamics from a single time series For this second part, we applied Takens Theorem to a famous example: the Lorenz attractor. To begin with, we created data to generate the Lorenz attractor and visualize it, using code from the previous exercise:

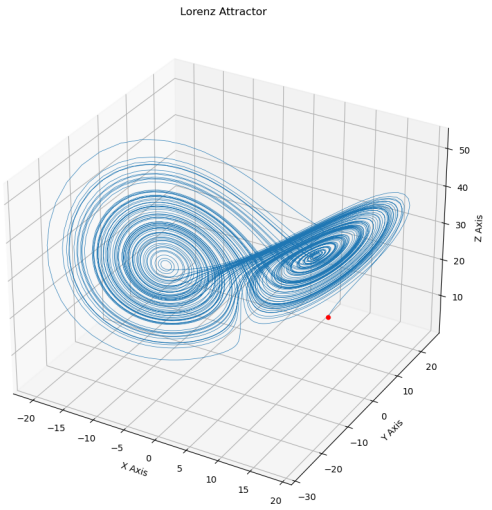

Figure 20: Lorenz Attractor $\sigma = 10, \rho = 28, \beta = 8/3$

The dimension of the Lorenz attractor is a fractal dimension, comprised between 2 and 3. Taking the upper bound 3, we would need 7 dimensions to embed it according to Takens Theorem. However, since visualization would be impossible, we implement a 3-dimensional time-delay embedding. First, we used the coordinate x to perform the embedding:

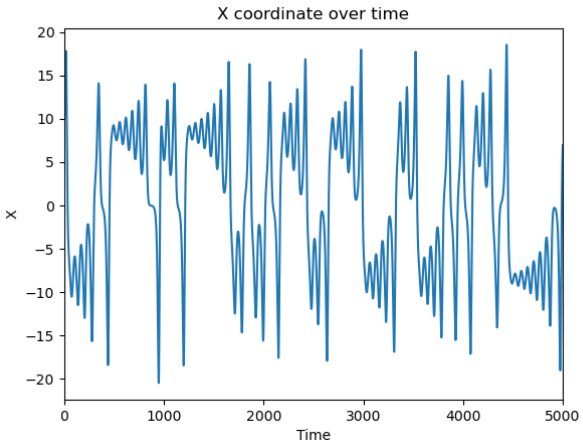

Figure 21: Coordinate X of Lorenz attractor over time

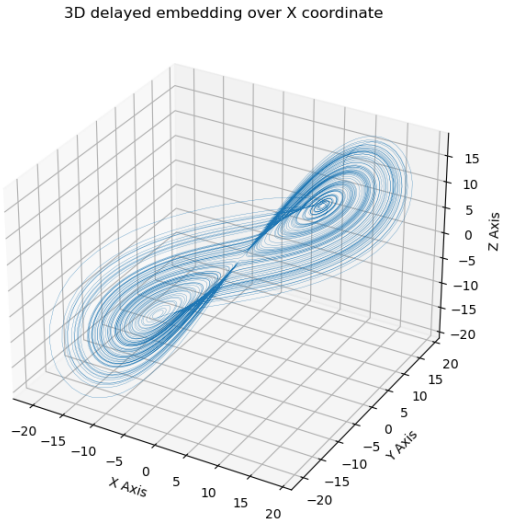

Figure 22: 3-D time-delayed embedding over X coordinate for Lorenz attractor, $\Delta_t = 5$

We have a new proof here that Takens Theorem offers a sufficient but not neccessray condition: this time-delay embedding using X coordinate is an embedding of the Lorenz attractor, even though we used a lower dimensional embedding than recommened by Takens. We can see indeed that the topological properties of the attractor are conserved, such as the characteristic butterfly structure.

We then performed a time-delayed embedding over the z coordinate, and compared the results.

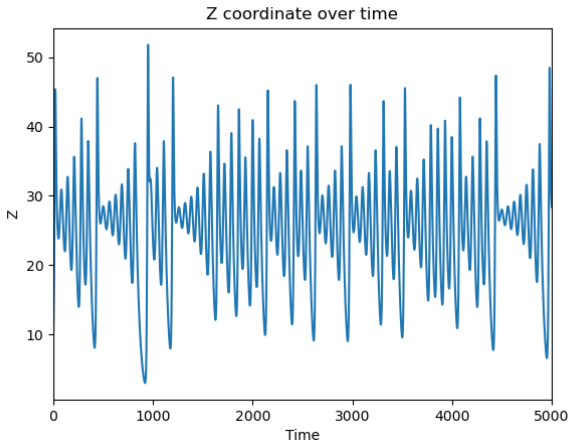

Figure 23: Coordinate Z of Lorenz attracttor over time

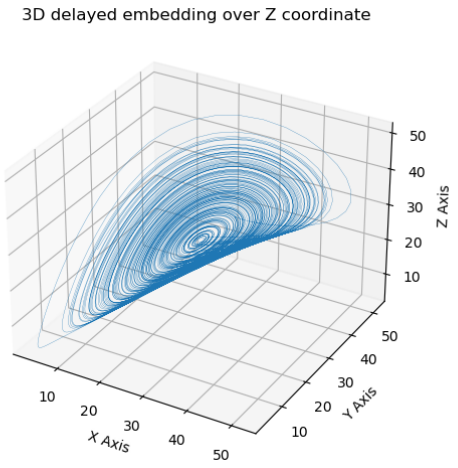

Figure 24: 3-D time-delayed embedding over X coordinate for Lorenz attractor, $\Delta_t = 5$

Thee mbedding with z coordinates does not work, indeed we are left with a single center circular structure instead of the characteristic butterfly shape of the original Lorenz attractor. Due to the limited variability and slower dynamics of the z-coordinate, the embedded plot might not capture the full complexity of the attractor, leading to a less accurate reconstruction compared to embedding using the x or y coordinates.

Report on task 5, Learning crowd dynamics

Part 1: Create a reasonable state space and perform PCA The state space can be seen as a manifold embedded by 9 coordinates (x1, x2, ... x9) each coordinate being a utilization measure of one particular measurement area; the first column of the dataset being the timesteps. Since it is most probably a one dimensional periodic manifold (like we saw in task 4), according to Takens theorem we can use 3 dimensions to embed it.

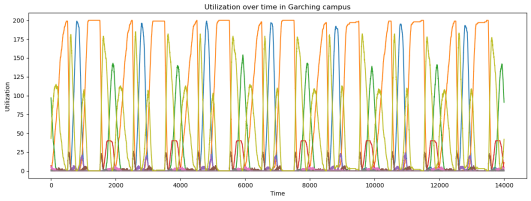

Figure 25: Utilization values of the 9 measurements areas over 7 days in Garching campus

We created a delay embedding with a delay of 350, by creating sliding windows of 351 from the top to the bottom of the file over the 3 first columns (excluding the timesteps of course) rendering us a datsaset of around 13500 rows by 1053 columns. We then performed a PCA over those 1053 columns, keeping the first 3 compoments as stated by Takens Theorem.

Part 2: Color the points by all measurements taken at the first time point of the delays, for all nine measurement areas After obtaining our new representation of the state space through the first 3 principal components from the PCA, we can produce 9 plots representing the new state space as a periodic curved, colored by the utilization value of the 9 measurements areas.

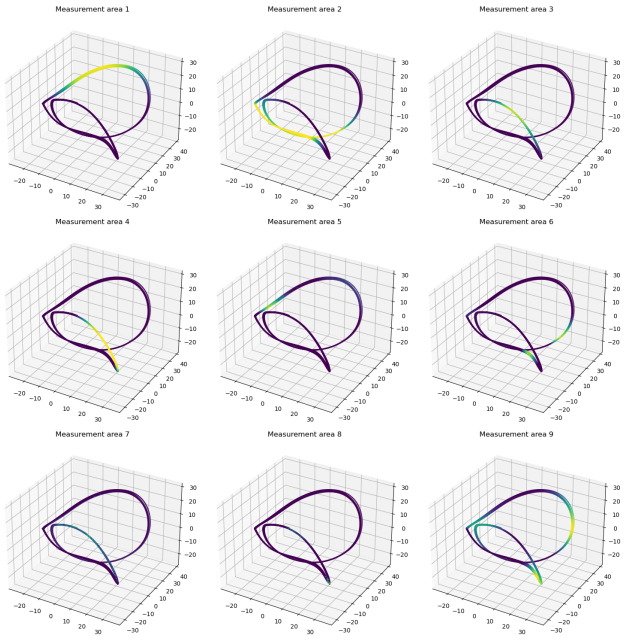

Figure 26: PCA state space colored by measurements values of all 9 areas

As we can see from the plots, our newly defined state space is periodic and 1 dimensional. We can see what parts of the curve are associated to which measurement area: for example, measurement area 1 takes up the upper part of the curve, whereas measurement 2 lies in the downer part. Measurement areas 5 to 8 occupy smaller portions of the curve than the others.

Part 3: Learn the dynamics on the periodic curve you embedded in the principal components.

Here we computed the arclength of the curve in the PCA space, then the velocity over the arclength before approximating the arclength velocity with RBF method.

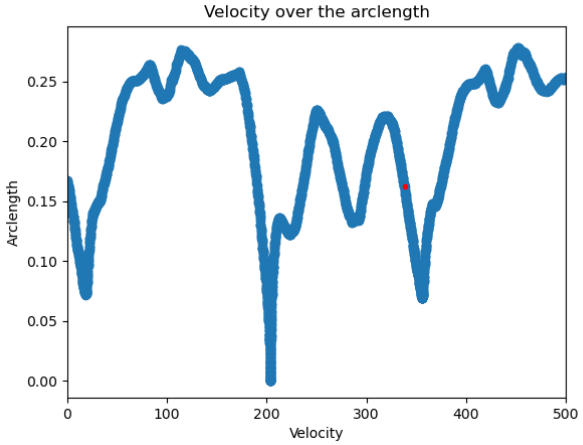

Figure 27: Arclength velocity over PCA space

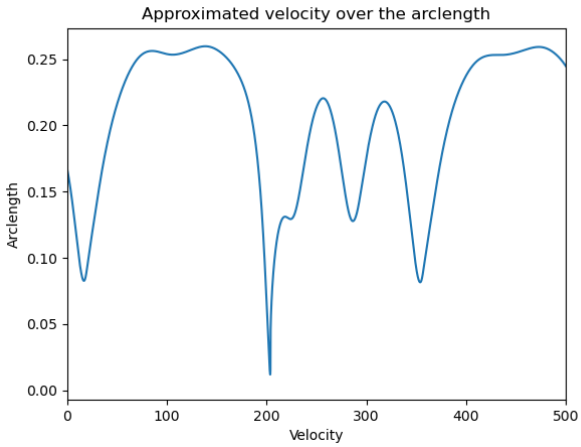

Figure 28: RBF approximation of the arclength velocity

Part 4:Predict the utilization of the MI building for the next 14 days with your system, and plot the results over time.

Finally, we used our approximation of the arclength velocity to integrate it over the 1-d curve, get the arclength values over times and finally approximate with RBF the utlization of MI-building over time.