Report for exercise 5 from group E

Tasks addressed:	5
Authors:	Çağatay Gültekin (03775999) Vatsal Sharma (03784922) Zhitao Xu (03750803) Gaurav Vaibhav (0366416) Minxuan He (03764584)
Last compiled:	2024–01–18
Source code:	https://gitlab.lrz.de/mlcms-ex-group-e/mlcms-ex-group-e

The work on tasks was divided in the following way:

Çağatay Gültekin (03775999)	Task 1	100%
Minxuan He (03764584) **Project lead**	Task 2	100%
Vatsal Sharma (03784922)	Task 3	100%
Zhitao Xu (03750803)	Task 4	100%
Gaurav Vaibhav (0366416)	Task 5	100%

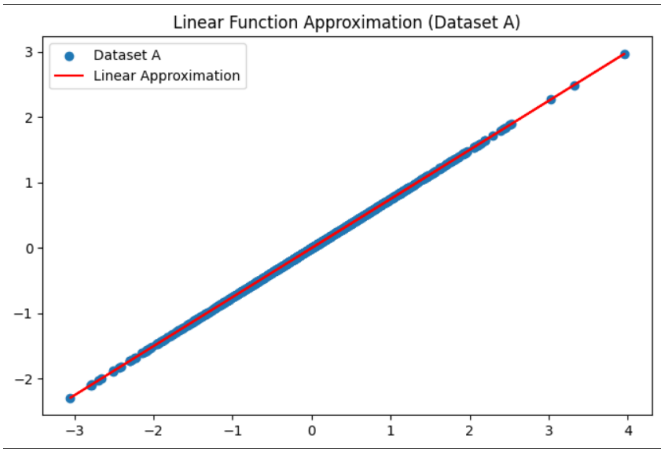


Figure 1: Linear Function Approximation on Linear dataset

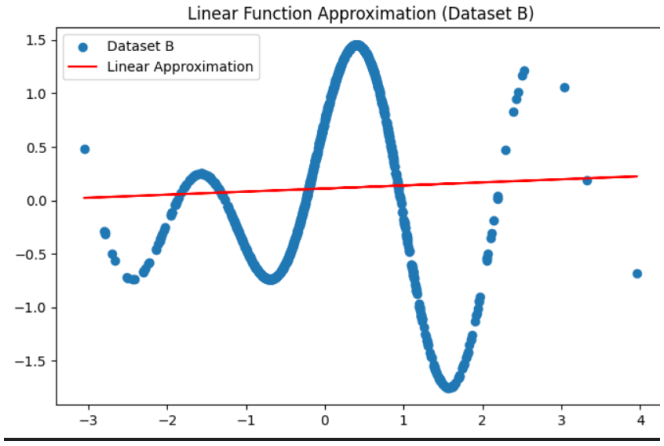


Figure 2: Linear Function Approximation on Non-Linear dataset

Report on task 1, Approximating functions

In this task, we investigated the approximating function for both linear and non-linear data. While doing this, we used linear functions and combinations of radial functions. For the least squares solution, we used scipy.linalg.lstsq as stated in the exercise sheet for all the subtasks. In addition, according to the practice sheet, we used np.linspace for the subtasks.

1.1 Approximate the function in the dataset (A) with a linear function

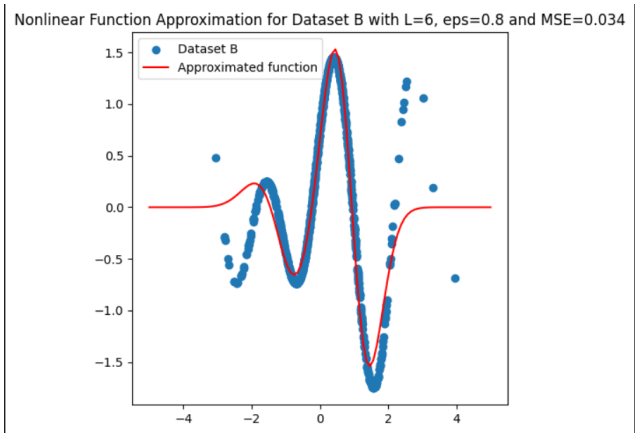
In this subtask, we used linear function to approximate a given linear dataset which is dataset A. We used the cond parameter as None. This is the graph of the actual data and approximated function1.

1.2 Approximate the function in the dataset (B) with a linear function

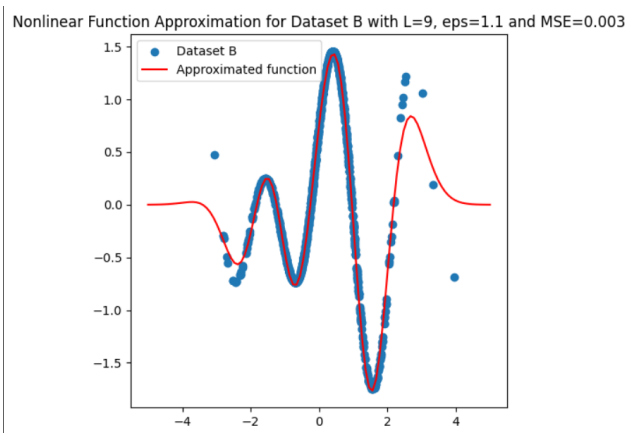
In this subtask, we used linear function to approximate a given non-linear dataset which is dataset B. We used the cond parameter as None. This is the graph of the actual data and approximated function2.

1.3 Approximate the function in the dataset (B) with a combination of radial functions

In this subtask, we use a combination of radial functions to approximate a given non-linear dataset which is dataset B. We choose to use epsilon squared in the denominator for the radial functions. Our main reason is not to have very different numerical data. We choose the cond parameter as 1e-5 to overcome the overfitting issue. We used many L(number of radial functions) and epsilon values to create different function approximations and we saw that less than L=6 and more than L=18 gives a very bad graph and MSE in general. The main reason for that is when we use too little L, it underfits the data and when we use too large L then it starts overfitting. The same case was observed when we used less than 0.2 and more than 2 for the epsilon values. So, our L values are (6, 9, 12, 15, 18) and our epsilon values are (0.2, 0.5, 0.8, 1.1, 1.4, 1.7, 2). We use all the combinations to see how the function approximations change according to parameter couples and we show the MSE values in the graphs because it shows how it is good as a numerical value(lower is better because it is an error function). These are the best couples for every L value: 3, 4, and 5.

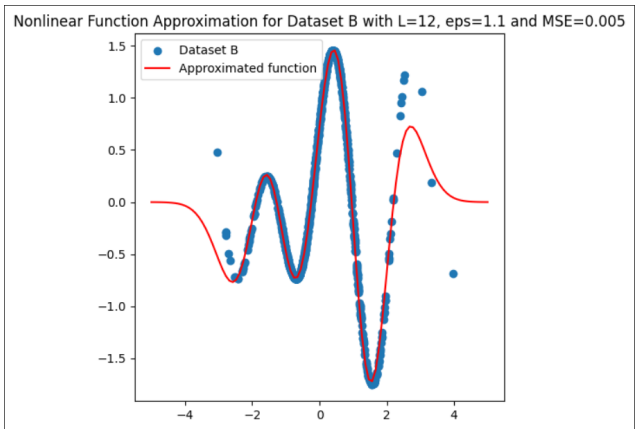


(a) L=6

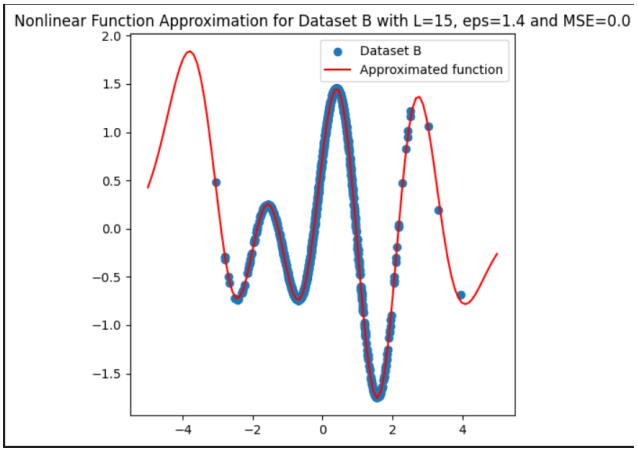


(b) L=9

Figure 3: The best graphs for L=6 and L=9



(a) L=12



(b) L=15

Figure 4: The best graphs for L=12 and L=15

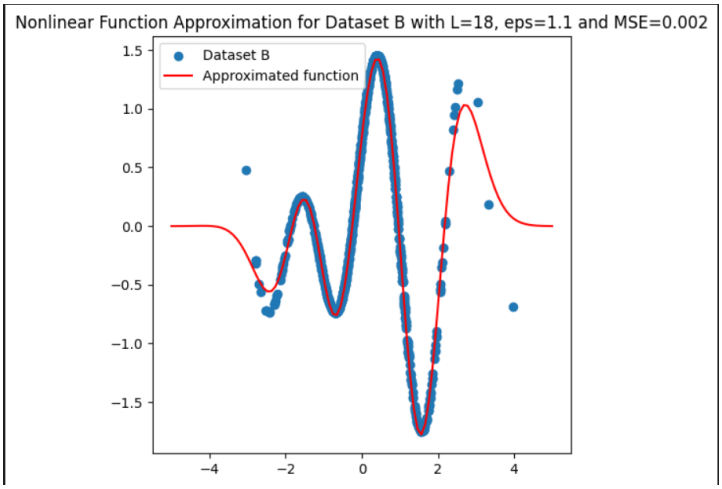
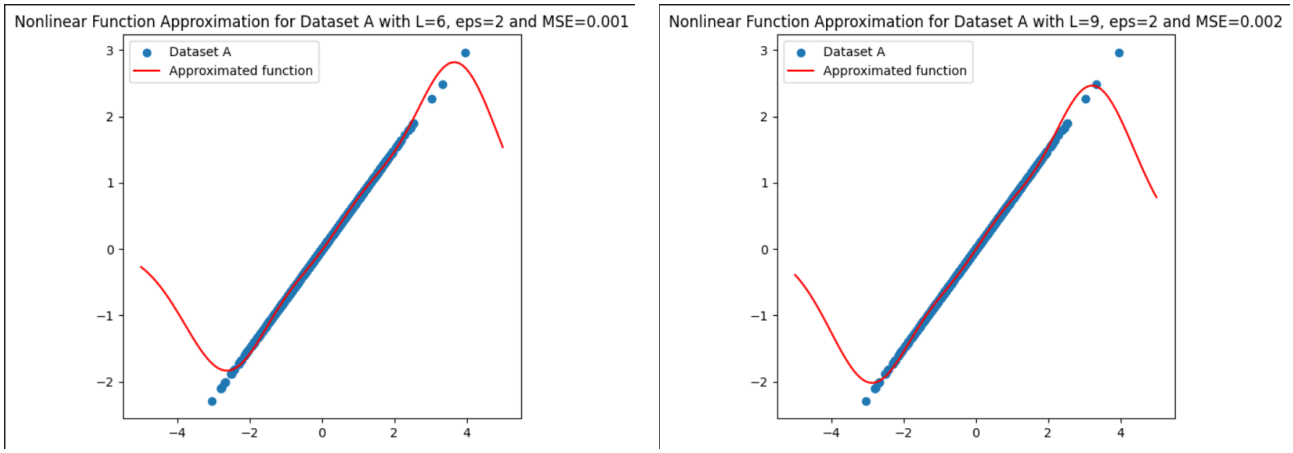


Figure 5: The best graphs for L=18

1.4 Why is it not a good idea to use radial basis functions for dataset (A)?

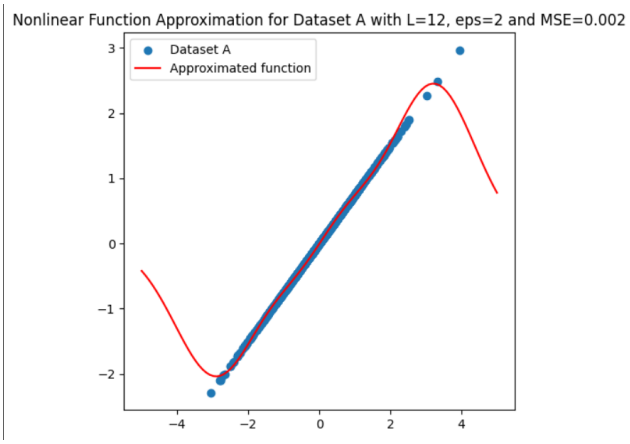
To answer that question we used the system that we used in our non-linear dataset in the linear dataset and we see that it is not a good approximation for that dataset. However, when we choose epsilon so large like a hundred, it is perfectly fitting because linear datasets' true function f is very smooth. On the contrary, it is not a good idea to use radial basis functions for linear datasets because it will make it more complex than the linear solution. When we get an overly complex model with a non-linear function for linear data, it can cause overfitting and it may harm the function approximation process because of the computational complexity.

Our L values are (6, 9, 12, 15, 18) and our epsilon values are (0.2, 0.5, 0.8, 1.1, 1.4, 1.7, 2, 100). We use all the combinations to see how the function approximations change according to parameter couples and we show the MSE values of the graphs because it shows how it is good as a numerical value(lower is better because it is an error function). For epsilon=100, graphs are always the same for every L value so we just put one of them. These are the best couples for every L value and epsilon=100 case: 6, 7, and 8.

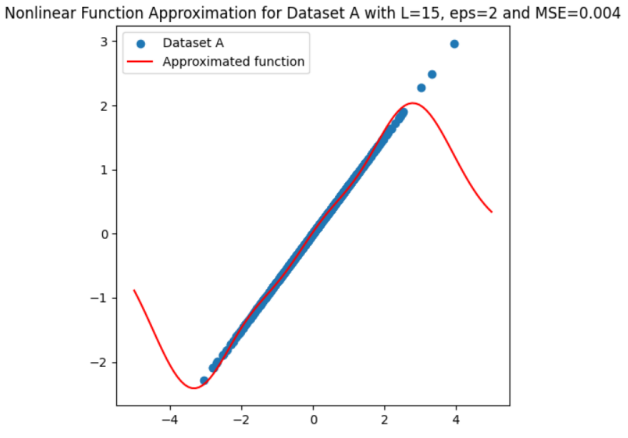


(a) L=6 (b) L=9

Figure 6: The best graphs for L=6 and L=9

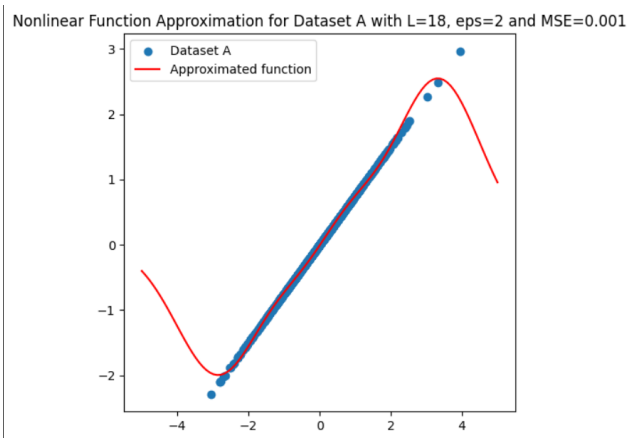


(a) L=12

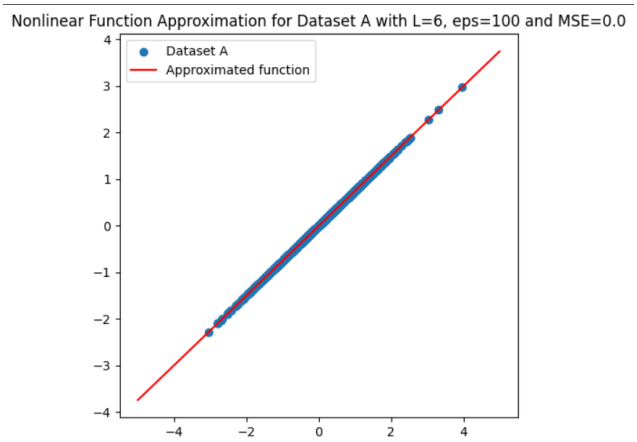


(b) L=15

Figure 7: The best graphs for L=12 and L=15



(a) L=18



(b) epsilon=100

Figure 8: The best graphs for L=18 and the graph for all L values when epsilon value becomes 100

Report on task 2, Approximating linear vector fields

In the second task, we need to download `linear_vectorfield_data_x0.txt` and `linear_vectorfield_data_x1.txt` from Moodle. They each contain 1000 rows and two columns for the 1000 data points x_0 and x_1 in two dimensions. We will use this data next.

2.1 Estimate the linear vector field that was used to generate the points x_1 from the points x_0

`linear_vectorfield_data_x0.txt` and `linear_vectorfield_data_x1.txt` as shown in the figure 9, the data represent the status of the system at different time steps.

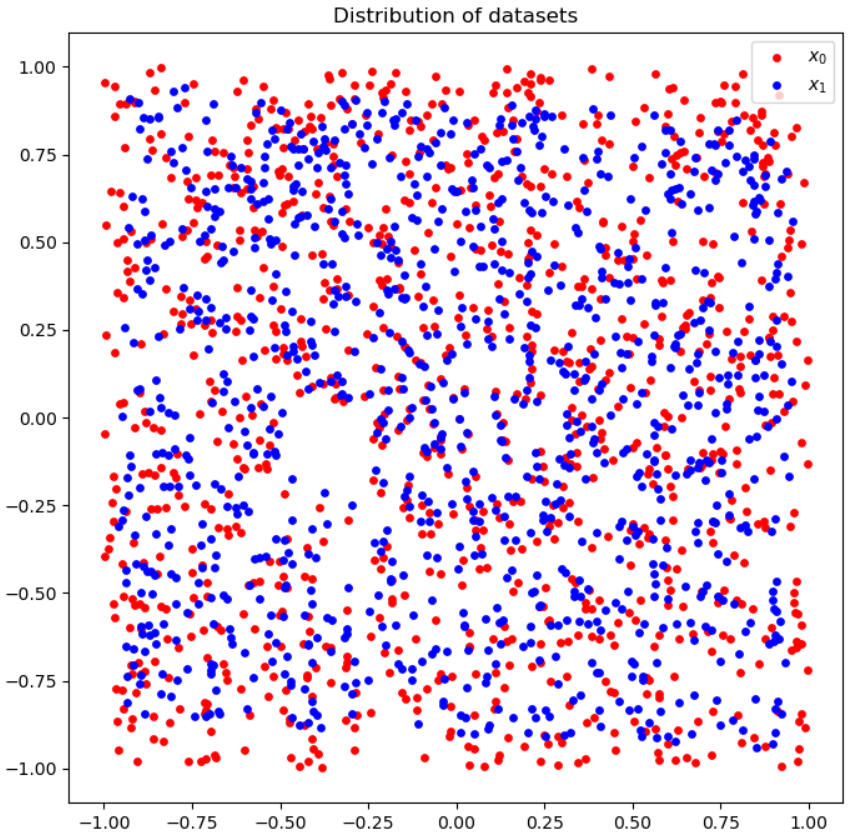


Figure 9: Distribution of datasets

We can use the formula 1 finite difference formula for the vector field $v^{(k)}$ which describes the motion of $x_0^{(k)}$ in time Δt. Note: Δt in the formula 1, because the vector field is a linear system of point $x_0^{(k)}$, Δt does not affect $x_1^{(k)}$ prediction, so we take a value of 0.1.

$$\hat{v}^{(k)} = \frac{\hat{x}_1^{(k)} - \hat{x}_0^{(k)}}{\Delta t} \tag{1}$$

Approximate matrix A using the algorithm of least squares minimization (via `np.linalg.lstsq() function`). Expect that for all k,

$$v(\hat{x}_0^{(k)}) = \hat{v}^{(k)} = A\hat{x}_0^{(k)} \tag{2}$$

The final corresponding vector field is shown in figure 10.

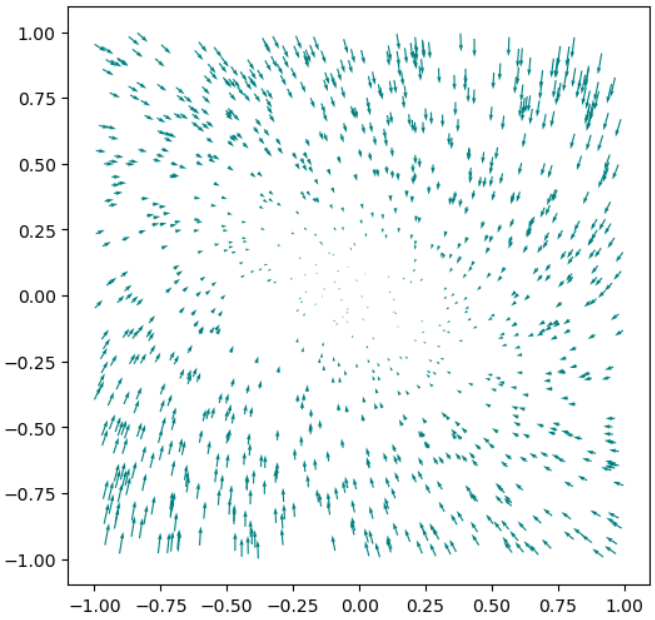


Figure 10: Linear vector-field dataset

2.2 Solve the linear system and compute the mean squared error.

From the first part, we find the approximate matrix $\hat{A}$ as:

$$\begin{bmatrix} -0.49355245 & -0.4638232 \\ 0.23191153 & -0.95737573 \end{bmatrix}$$

From the linear system $\dot{x} = \hat{A}x$ we can get $x_1^{(k)}$. According to the tip in the tutorial, we use a more accurate integration method (`integrate.solve_ivp` function) instead of the Euler method. Then substitute the obtained $\hat{x}_1^{(k)}$ into the formula 3 to get the mean square error (N=1000). The MSE is 0.003059927595989735. It is small and represents a good approximation.

$$\frac{1}{N}\sum_{k=1}^{N} \|\hat{x}_1^{(k)} - x_1^{(k)}\|^2 \tag{3}$$

2.3 Choose the initial point and solve the system, Visualize the trajectory and the phase portrait.

This part of the task selects a point (10,10) far away from the initial data and solves its trajectory, Tend =100. We need to solve the linear system $\dot{x} = \hat{A}x$ again. By passing the initial point and estimated matrix $\hat{A}$ to the `integrate.solve_ivp` function, for the linear system, is solved and the trajectory of the system is obtained(Fig.11). The red represents the initial point, the orange line represents the trajectory, and the blue is the phase portrait. The phase diagram represents the change of the system in the state space over time. We can see that this dynamical system has a single stable attraction locally at the origin. Through visualization, we can observe the changes in the system more easily and intuitively.

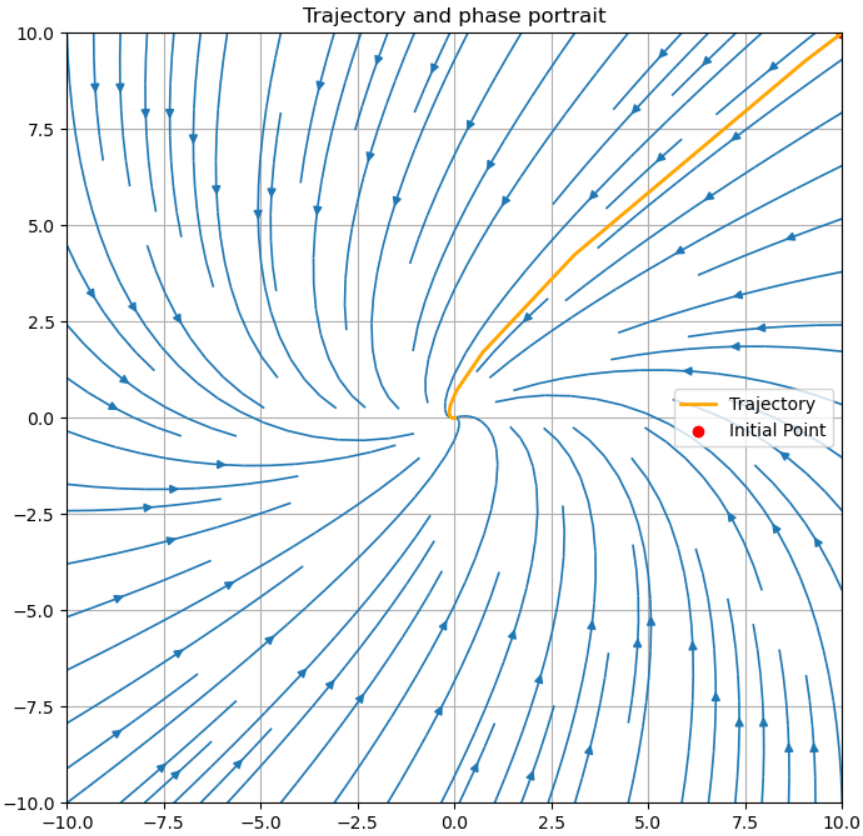


Figure 11: Phase portrait and trajectory

Report on task 3, Approximating nonlinear vector fields

Supervised machine learning framework aims at predicting a target value y after the model has seen enough (x,y) tuples. In this question, we deploy machine learning algorithm to identify and approximate the convolution operator of a particular dynamical system. As in any typical ML problem, given ordered tuples (x,y), we choose a family of functions that we intend to fit and later generalize for the upcoming test data.
Here we are given two txt files **nonlinear_vectorfield_data_x0** and **nonlinear_vectorfield_data_x1**, where the former contains the position coordinates of 1000 particles at time t = 0, and the latter stores the position coordinates of the same 1000 particles at time t = 0.01. Our task is to identify and approximate the hidden function that maps these two position sets. We will be working of two families of functions - linear ones and non-linear ones.

3.1 Linear Convolution

$$\frac{d}{ds}\psi(s, x(t))\bigg|_{s=0} \approx \hat{f}_{\text{linear}}(x(t)) = Ax(t).$$

Figure 12: Update Rule

Our aim for this part will be to approximate A as accurately as we can. Notice we are not including any bias b in our analysis right now. At a high level, every position tuple of the form (x,y) gets multiplied with A and hence changes into a new position coordinate tuple at every instant, see 12. We wish to calculate A now. Again to find this matrix we need a principled way, which we achieve by following the direction of minimising the loss function given to us. In our analysis, we are using mean squared error as the loss function. Luckily in

linear systems, there always exists a closed-form solution for A such that the loss reaches the global minimum. This A can be given as -

$$\hat{A}^T = (X^T X)^{-1} X^T F.$$

Figure 13: Closed form solution for A

In this particular task, we intend to manually perform these matrix multiplications and reach A, in the next subtask, we will use prebuilt least square minimizer libraries. This is intentionally done to learn both hand coding, and also to learn the stable library routines provided by Python. After running the code, we calculate A to be -

$$\begin{bmatrix} 0.95064476 & -0.04638232 \\ 0.02319115 & 0.90426243 \end{bmatrix}$$

After calculating the coefficient matrix, we are supposed to predict positions for those 1000 particles through our model for time t = 0.01 and compare it with the ground truth data given to us. We are also asked to report the differences between both of them through the MSE (mean squared error).

On running our analysis, we find the MSE to be **0.002933405709444968**. See figurs 14 for the phase portrait diagram and 15 for visualization of differences through MSE.

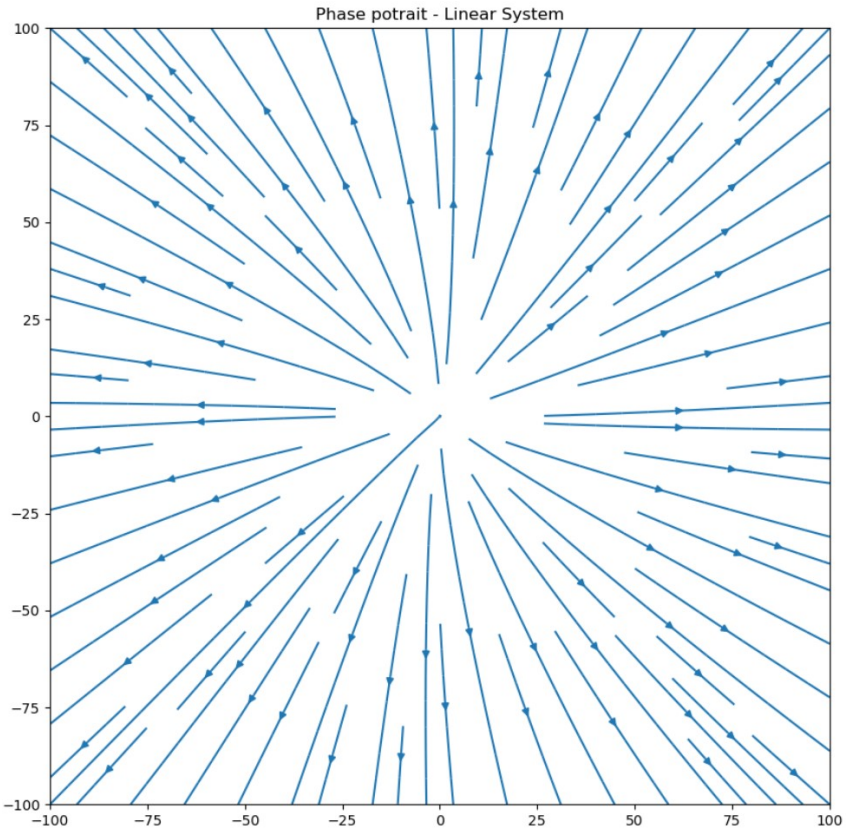


Figure 14: Phase Portrait Diagram

3.2 Non-linear Convolution

Now we proceed to the section where we will use the family of non-linear functions. We use RBFs to approximate the underlying non-linear mapping. This time we use the python library **numpy.linalg.lstsq,**

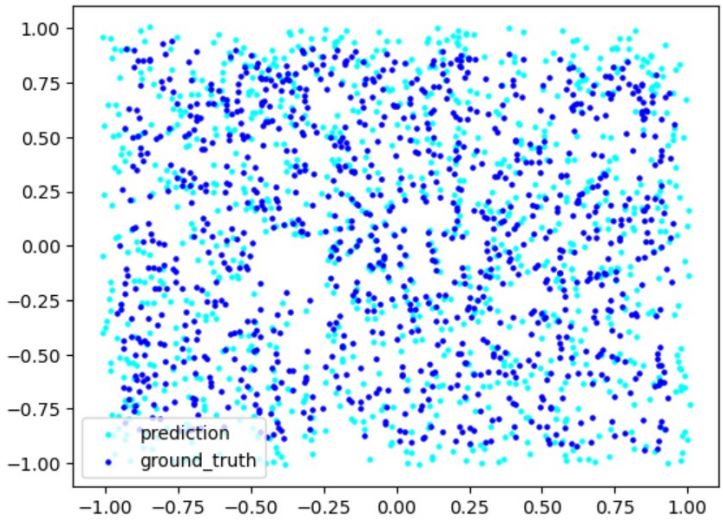


Figure 15: Scatter plot between predicted and actual values

$$\frac{d}{ds}\psi(s, x(t))\Big|_{s=0} \approx \hat{f}_{\text{rbf}}(x(t)) = C\phi(x(t)).$$

Figure 16: Update Rule-2

scipy.linalg.lstsq to estimate our target matrix. Also to reach the right number of clusters and the right value of width, we iteratively choose from a list of L and epsilons.
L = [100, 150, ... 1000]
and eps = [0.3, 0.5, 0.7, 1.0, 5.0, 10.0, 20.0]

After running our analysis, we achieve the final results -

Optimal cluster number - 858
Optimal width (epsilon) - 0.3
MSE value - 2.58e-06

We can see a significant drop in the error upon using the family of non-linear functions as compared to the linear function. Hence we can safely conclude that **the hidden vector field is definitely non-linear**. Also in fig18 we can see a negligible difference between the predicted values and the ground truth values, representing how good our non-linear representation of vector field is.

```
Best configuration: eps = 0.3, n_bases = 858, and dt = 0.10606060606060606 gives MSE = 2.5801512610162232e-06
```

Figure 17: Result

3.3 Steady State of the system

We try and solve for a larger t = 50. We find multiple states of the system in case of non-linear vector fields. See fig19. Going by the rationale in previous assignment, two systems having different number of stable points **can not be topographically equivalent to each other.**

Report on task 4, Time-delay embedding

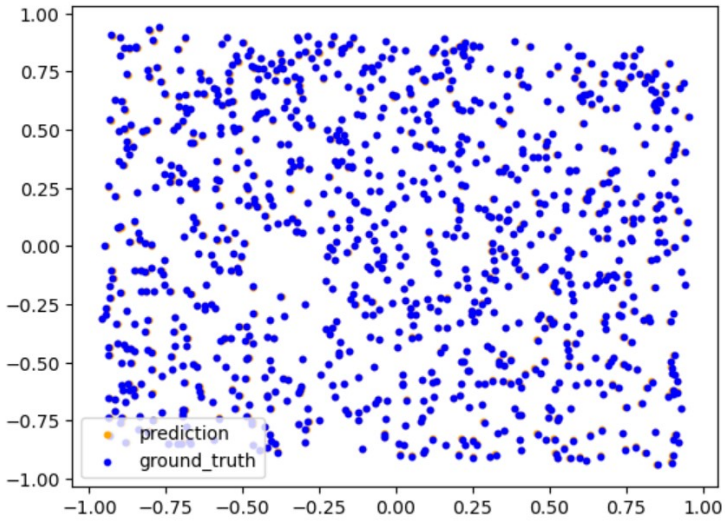


Figure 18: Scatter plot between predicted and actual values (non-linear case)

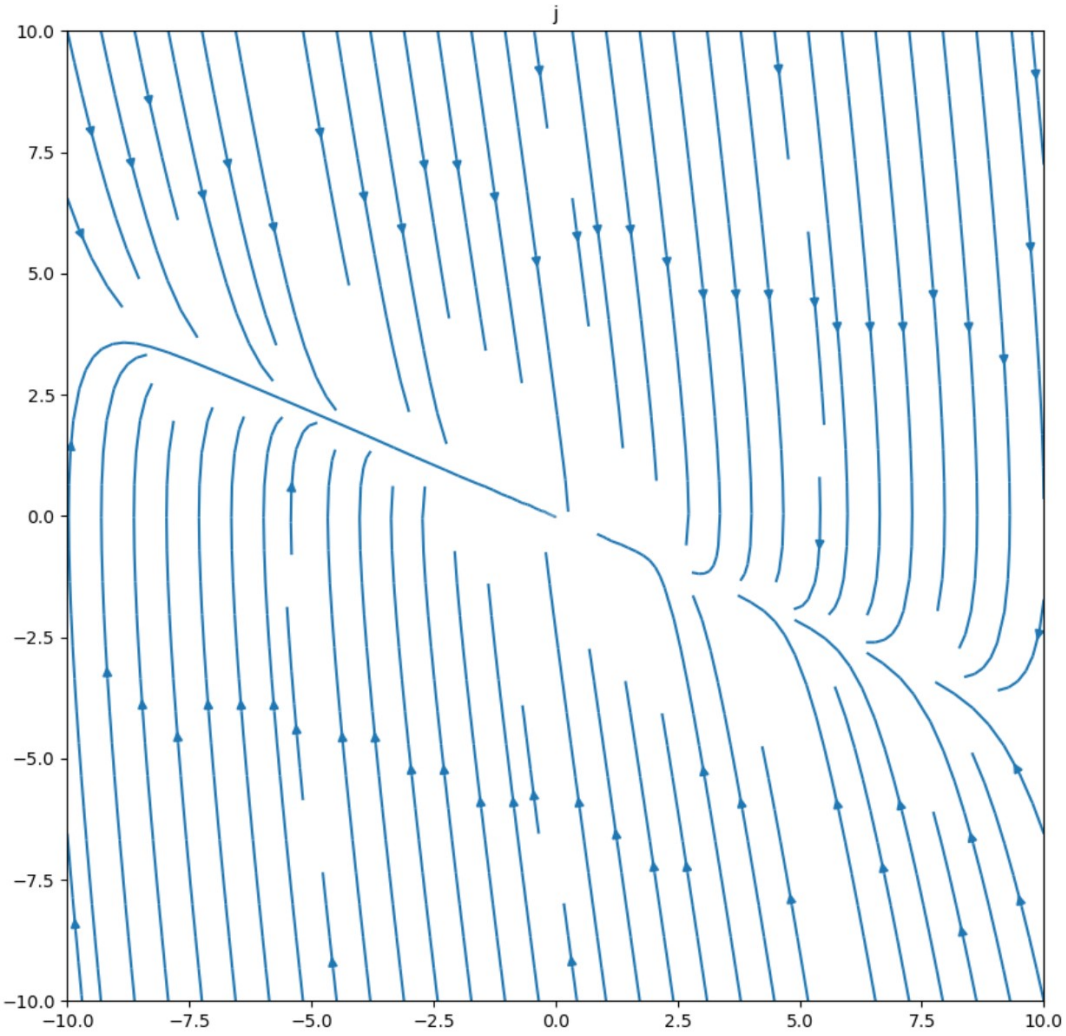


Figure 19: Phase portrait diagram (non-linear case)

As we know, in real world, we can never have the full state of a system. The only thing that we can use to learn the system is the observation. More specifically, the observation means the measurement from different kinds of sensors, such as cameras, temperature sensors, pressure sensors, etc. Time-delay embedding provides us a technique for reconstructing the dynamics of a system from time series observed data. This method is based on Takens theorem, which posits that even time series data from a single observable variable can sufficiently reconstruct the dynamics of a system. This reconstruction is particularly useful for understanding and analyzing complex systems, such as chaotic systems, weather patterns, or stock market dynamics.

Generally, to create the time-delay embedding for some time series data, we need to perform the following steps:

1. **Selecting embedding dimension d and time delay Δt**: The embedding dimension d refers to the dimensionality of the reconstructed space. The time delay Δt is the interval between each data point in the construction of embedding vectors;
2. **Constructing embedding vectors**: This involves creating a series of embedding vectors using the original time series data. At the time step t, we choose the first data point. The time delay Δt tells us after how many time steps do we choose the next data point, and the embedding dimension d tells us how many data points do we choose in total. Then we put all of these data points together, and concatenate them in order to form the time-delay embedding for time step t. For example, let's suppose we have a 1-D time series data x. If $d = 3$ and $\Delta t = 2$, then the time-delay embedding vector at time step t would be $[x(t), x(t + 2), x(t + 4)]$;
3. **Reconstructing the phase space**: All these embedding vectors together form the reconstructed phase space. This space captures the dynamical features of the original system.

In this task, we will create the time-delay embeddings based on the given dataset `takens_1` and the trajectory of the Lorenz attractor we discussed in exercise 4.

4.1 Part 1: Time-delay embedding of dataset `takens_1`

In this part, we will work with the dataset in `takens_1.txt`. We first load the dataset from the file as an array, and we found that the array has the shape $[1000, 2]$, which means that this dataset contains 1000 points in 2-D space. We plan to first plot the raw dataset in 2-D space to get an overall feeling of how the data actually looks like. Then we ignore the second coordinate, assuming that we only have the first coordinate, and try to obtain the time-delay embedding and plot it in the phase space.

In order to visualize the raw data, we write a function `plot_curve`, which can receive a 2-D array of the shape $[n, 2]$ or $[n, 3]$ as an argument. This function can visualize the data in 2-D or 3-D space, decided by the dimension of the input data.

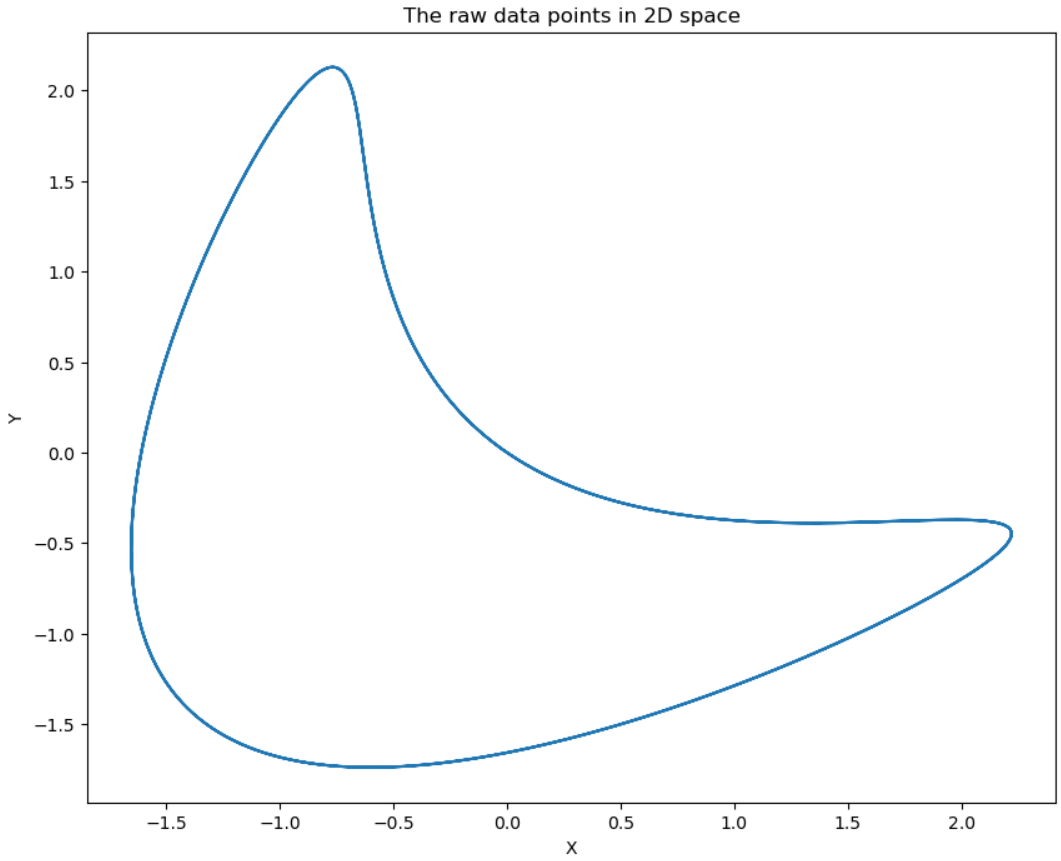


Figure 20: Visualization of raw data in `takens_1.txt`.

The raw data from `takens_1.txt` is shown in Fig. 20. We can see that, the data points form a closed loop, which means if these two coordinates represent e.g. two observed sensor values of a system, then there exists periodicity in this system. Furthermore, this closed loop can be viewed as a one-dimensional manifold. Now we ignore the second coordinate, and imagine that, the first coordinate is some sensor measurement at each time step. We need to study the dynamics of the system according to the first coordinate. We call the function `plot_curve` to visualize the first coordinate against time steps, and the result is shown in Fig. 21.

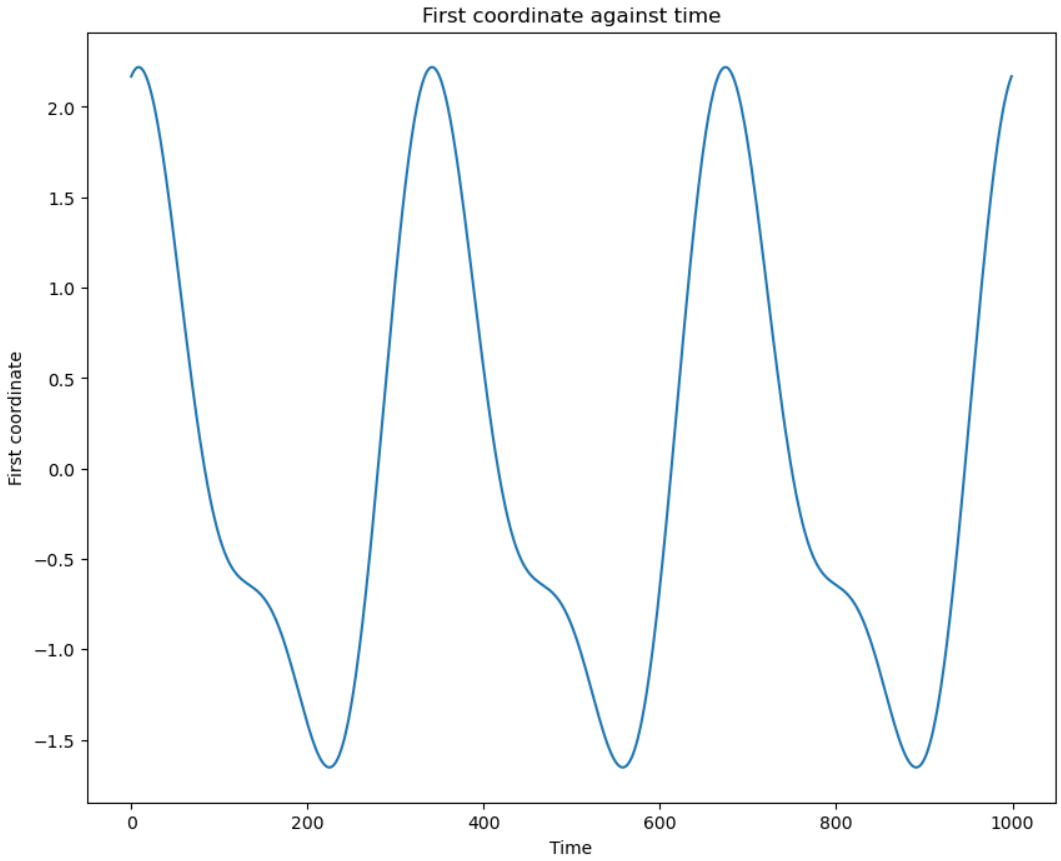


Figure 21: First coordinate against time.

Admittedly, from Fig. 21, we can observe the periodicity in the first coordinate along the time axis. However, we are not sure about if the combination of both coordinates will have the periodicity, since we don't know how the second coordinate will vary. Besides, from Fig. 21, we cannot obtain the exact position of the point in 2-D Euclidean space at each time step, and we even have no idea about in which direction it is moving. Fortunately, we have time-delay embedding, with which we can construct the phase space, and according to Takens theorem, this phase space can capture dynamical features of the system.

We define a class named as `TimeDelayEmbedding`, which has two methods, `time_delay` and `plot_delay`. The method `plot_delay` is just for visualization, while the method `time_delay` plays the key role. The `time_delay` accepts three arguments. The first one `col` represents which column of the raw data are we operating. Here, we have assumed that we only have the first coordinate, which means we are operating the column 0 of the data. The second argument `n` specifies the delay time steps Δn, or in other words, after how many time steps do we select the next value when we construct the embedding. The last argument `target_dim` denotes the dimension of the final time-delay embedding. For example, if we want to construct the embedding with the original coordinate and the delayed version Δn, then `target_dim` should be 2. If we want to construct with the original and the delayed version Δn and $2\Delta n$, then `target_dim` should be 3. In order to visualize the results, here we let the method only accept `target_dim` equal 2 or 3. So we have two lines of code at the beginning of this method to check if the shape of the input data is correct. If not, it will raise a `ValueError`. This function returns the time-delay embedding vectors as an array of the expected shape. This function obtains the time-delay embedding by shifting the array, and combining the common part of the original coordinate and the shifted one(s). The common part is found through appropriate slicing to the arrays. To illustrate, let's suppose we have sensor values at 5 time steps, and we set Δn to 1 and `target_dim` to 3. Then the function will return the part indicated by the red square in Fig. 22.

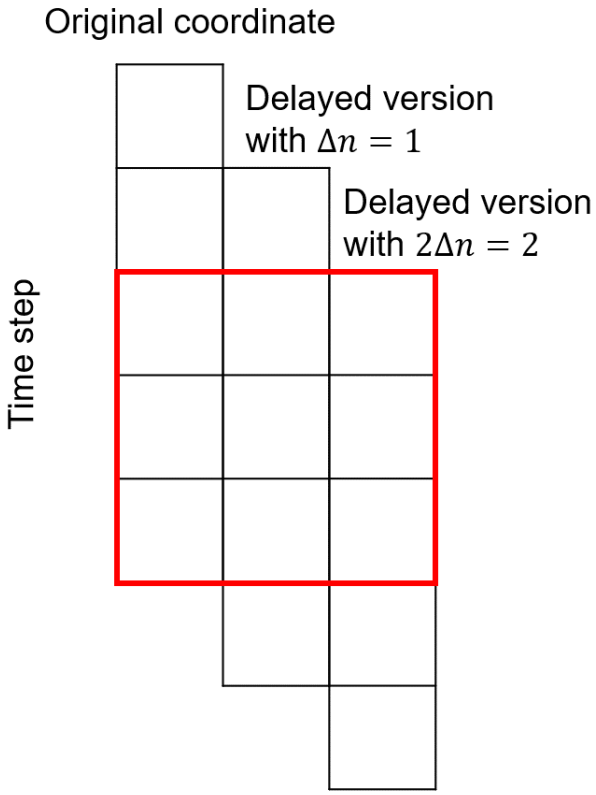


Figure 22: Illustration of how the method `time_delay` works.

We create an instance based on the class `TimeDelayEmbedding`, and pass the raw data. Then we call the method `time_delay` to get the time-delay embeddings with both 1 delayed version and 2 delayed versions. Here we choose $\Delta n = 5$. Finally, we plot them, shown in Fig. 23.

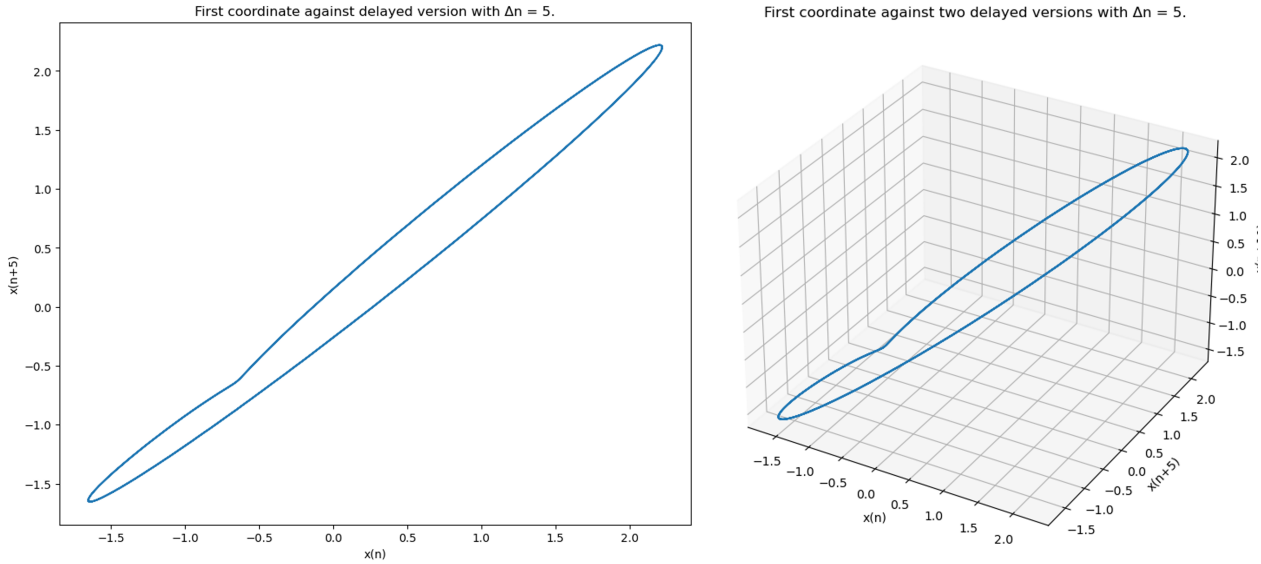


Figure 23: First coordinate against its delayed version(s). Left: First coordinate against delayed version with $\Delta n = 5$. Right: First coordinate against delayed versions with $\Delta n = 5$ and $2\Delta n = 10$.

According to Takens theorem, since the dimension of the manifold is 1, we need $2 \times 1 + 1 = 3$ coordinates to embed it. From Fig. 23, however, we can observe that the results of the first coordinate against one delayed version and two delayed versions are quite similar, and the left figure tells us 2 coordinates are sufficient to correctly embed this manifold.

4.2 Part 2: Time-delay embedding of trajectory of Lorenz attractor

We have already discussed the Lorenz attractor and plotted the trajectories under different settings of parameters in exercise 4. In this part, we will work on the trajectory created from exercise 4, and obtain the time-delay embedding based on a single time series. We copy the `lorenz.py` from last time into the current repository. Likewise, we use the same parameters setting with $\sigma = 10, \beta = 8/3, \rho = 28$, and the starting point is $(10, 10, 10)$. We use the function `lorenz_simulate` to calculate the trajectory array. Note here we pass the argument `t` (specifies the time points at which the solution should be reported) as an array from 0 to 1000 with 100000 time points for a smoother trajectory. The trajectory is shown in Fig. 24.

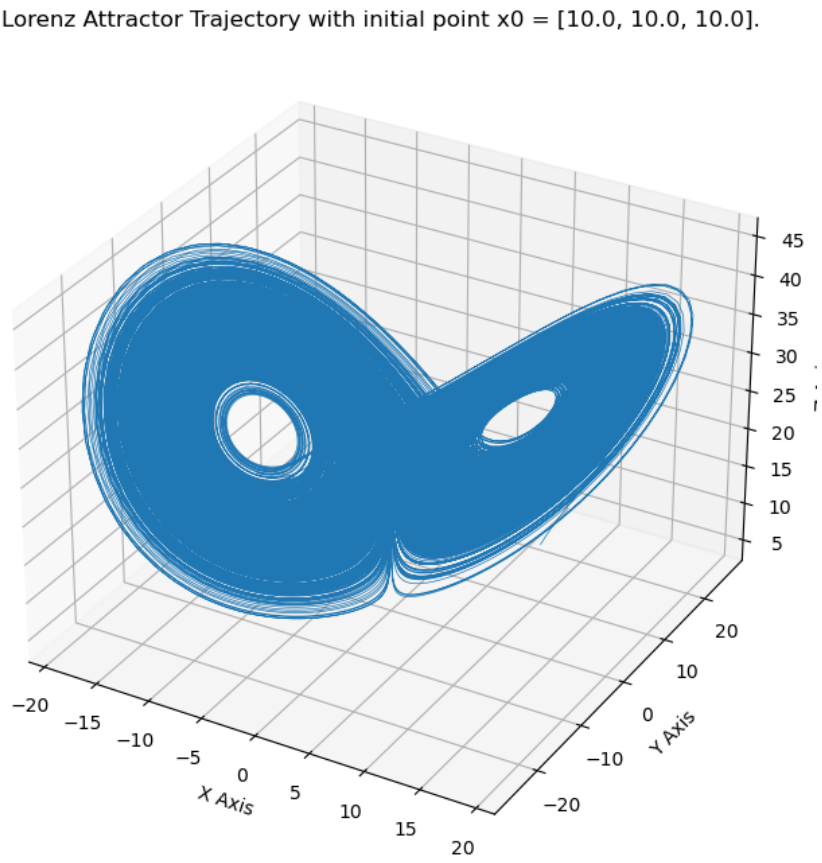


Figure 24: Lorenz attractor trajectory.

By calling the function `lorenz_simulate`, we simultaneously get the trajectory array of shape $[100000, 3]$. At each time step, there are 3 coordinates, and we call them x, y, z, respectively. Similar to part 1, we ignore the y, z, and assume that we only have x as some sensor values at each time step. We call the method `timedelay` to get an array consists of coordinate x and its delayed versions. We choose the delay $\Delta t = 10$ and we set the `target_dim` to 3, which means we are going to plot the coordinate $x(t)$ against its two delayed versions $x(t + \Delta t)$ and $x(t + 2\Delta t)$. The result is shown in Fig. 25. We can see that, the embedding is though not exactly the same as the original trajectory, they still have the same shape of "8". Therefore, this time-delay embedding based on the coordinate x successfully capture the key feature of the original trajectory. This suggests that the coordinate x contains enough information to reconstruct the dynamics of the full three-dimensional system, maintaining its topological features even in the reconstructed space.

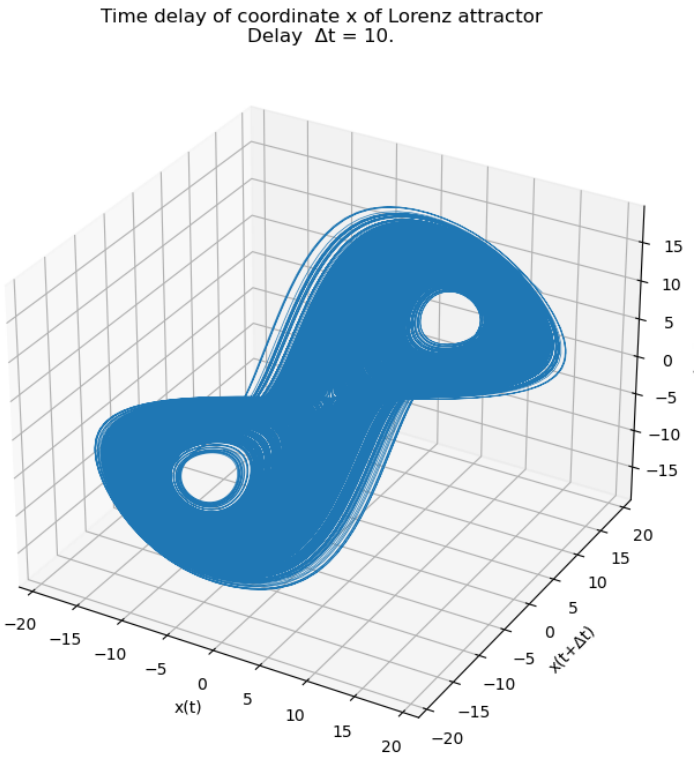


Figure 25: Time-delay embedding of Lorenz attractor: $x(t)$ against $x(t + \Delta t)$ and $x(t + 2\Delta t)$.

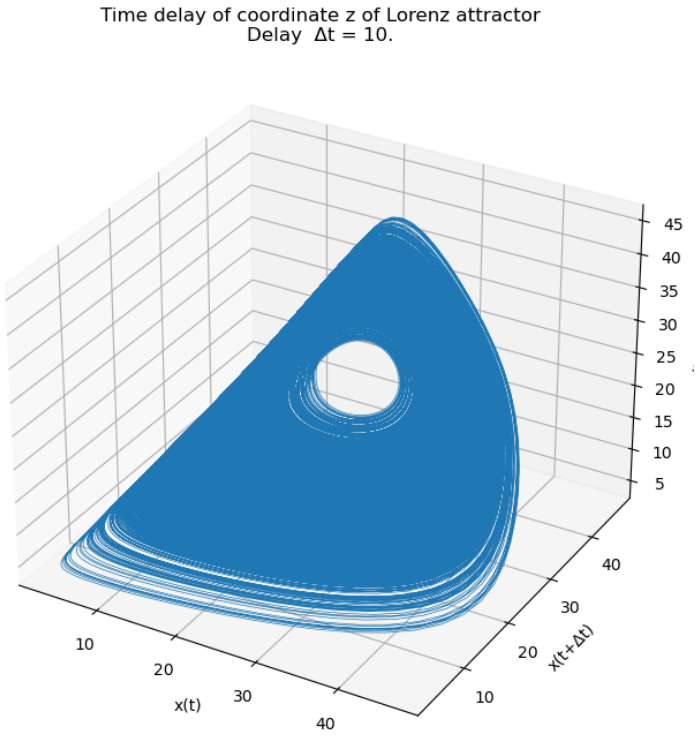


Figure 26: Time-delay embedding of Lorenz attractor: $z(t)$ against $z(t + \Delta t)$ and $z(t + 2\Delta t)$.

After that, we do the same thing to the coordinate z, and we show the result in Fig. 26. In Fig. 25, the

embedding retains the shape "8", while in Fig. 26, it exhibits a different shape (a "O" shape). This difference in shape compared to the original attractor and the coordinate x embedding could be due to the fact that the coordinate z might not encapsulate the same level of dynamical information as the coordinate x. In the Lorenz system, the z variable is involved in a different way in the equations, specifically it is coupled non-linearly with x and y and damped by the β parameter. The reconstruction from the coordinate z does not preserve the original system's topological features in the same way as the coordinate x does.

Report on task 5, Learning crowd dynamics

In this task, the dataset MI timesteps.txt has been provided which contains utilization data for several measurement areas on the campus, over the course of seven weekdays. As mentioned, a burn-in period of 1000 time steps at the beginning of the file is ignored.

5.1 State space representation of the given system

The given dataset is periodic and has no parametric dependence. So, locally, it is a one-dimensional closed loop. According to the Takens' theorem of Generic delay embeddings, dimension 2*intrinsic dimension(here 1) + 1 = 3 will be sufficient to embed the manifold space spanned by this dataset. Time-delay embedding using PCA was performed on prepared dataset of M(=12000)*1053 dimensions (created using 351 delays of first three measurements to create a vector of length 351*3=1053 dimensions repeated for consecutive M data rows). The state space representation using 3 principal component axes was plotted in fig. 27. One can confirm that is a closed loop and hence, 1 dimensional.

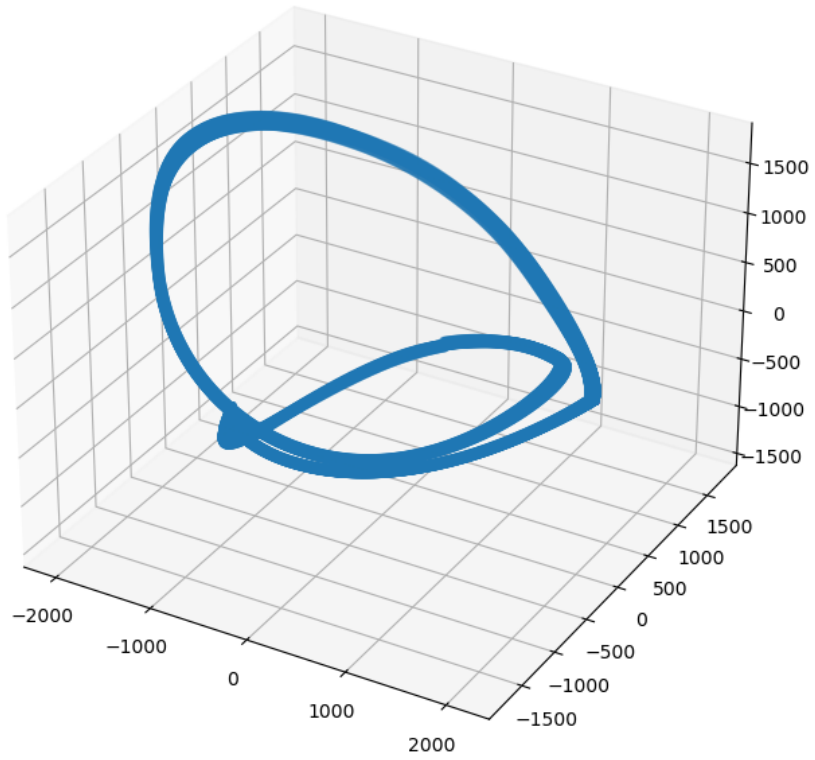


Figure 27: State space representation(using 3 principal axes)

5.2 Measurement areas plots in 3D embedding space

Nine measurement plots on 3D PCA embedding space corresponding to nine measurement areas (where all the points will be in the same position with only change in color) have been shown in fig. 28. When we colour the PCAs by each of the 9 areas, we see the change in densities for each of the 9 regions, shown in yellow. We see that the most of the density is contributed by the first 3 areas. Thus making our choice of picking the first 3 coordinates to approximate the manifold, to be a suitable choice.

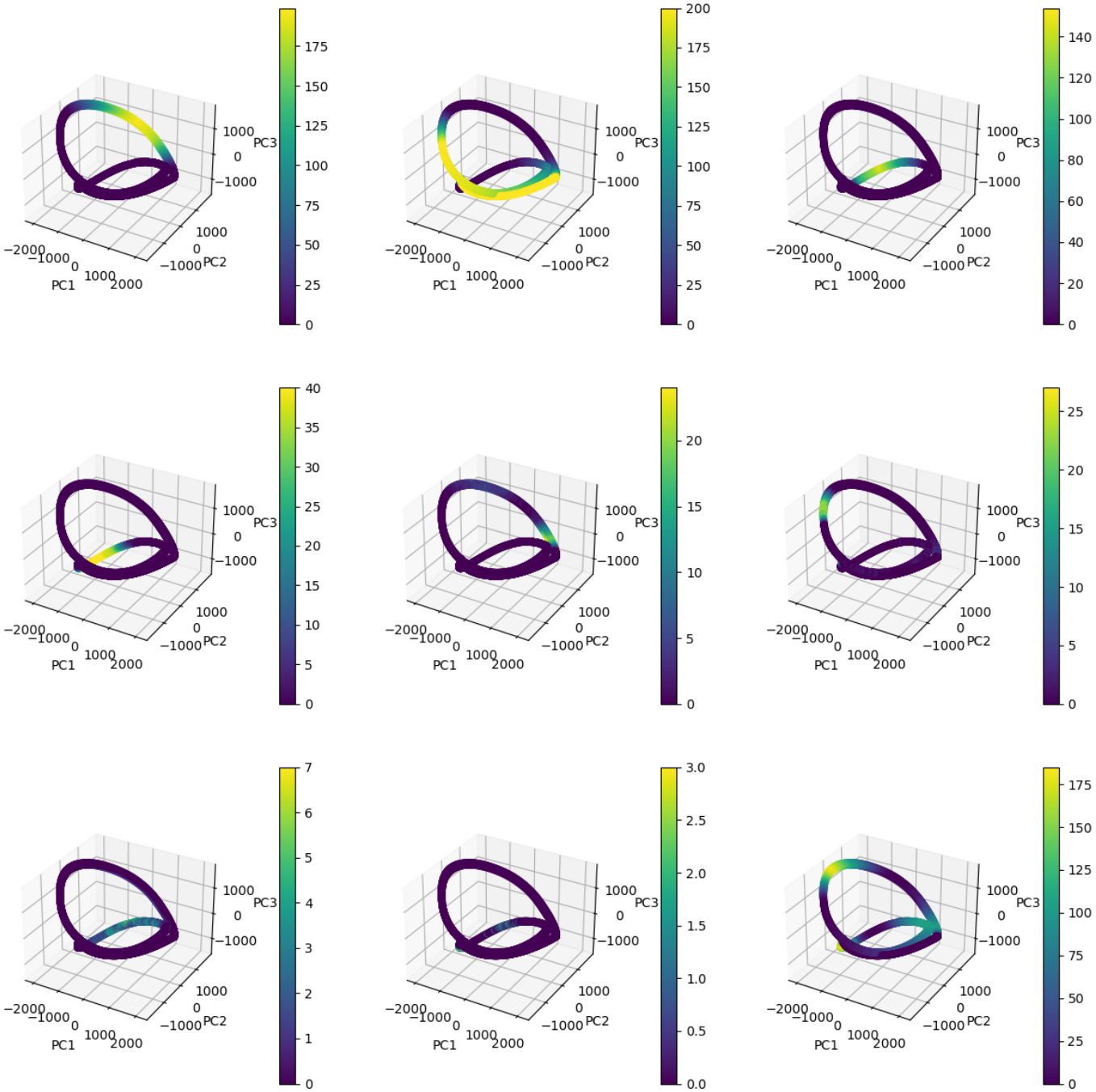


Figure 28: 9 measurement plots on PCA embedding space

5.3 Learning the dynamics

Learning the dynamics on periodic curve obtained from the PCA embedding space is carried out by determining how fast the system advances over the PCA space at every point in the space. In order to measure this, the velocity vector field is calculated by approximating the change of arclength over dt time interval. The velocity and cumulative arclength for one period have been shown in 29.

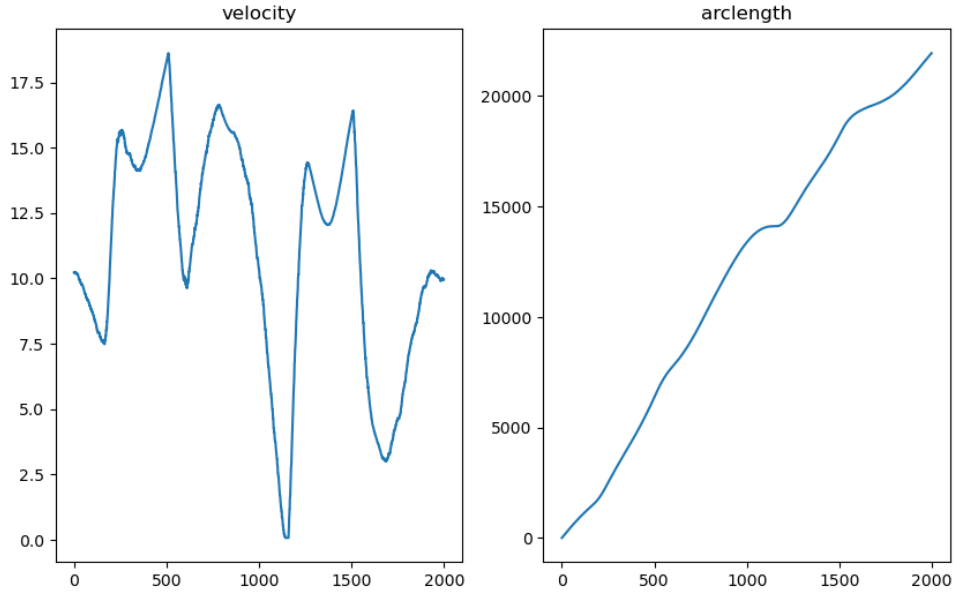


Figure 29: Velocity and arclength for 1 period

5.4 Predicting the utilization of the MI area

In this final subtask, the utilization of the MI area over the next two weeks was to be estimated.

Choice of parameters L and ϵ

We choose the value of ϵ to be 100. We choose this value because the accuracy of the prediction increases with smaller values of $1/\epsilon$. However, setting $1/\epsilon = 0$ does not produce the least error. The minimum error is often found at some "small" value of $1/\epsilon$, as is the case in this example. While this is typical, results change based on f, the basis functions chosen, and the data set used to construct the approximation. The value of L has to be appropriately chosen so that the kernel produces a positive semi-definite matrix. Choosing the value of L to be very large will slow down computation and will lead to overfitting whereas choosing a value too small may lead to underfitting. For our example we have used L=1000. This is because we have over 14000 datapoints. And the trajectories for each day of the 7 days is defined by 2000 points. Hence, we have decided to use half the number of points i.e 1000 as the number of radial basis functions.

Firstly, Non-linear function approximation of velocity curve for 1 period w.r.t arc length using radial basis function was carried out. This is shown in fig. 30

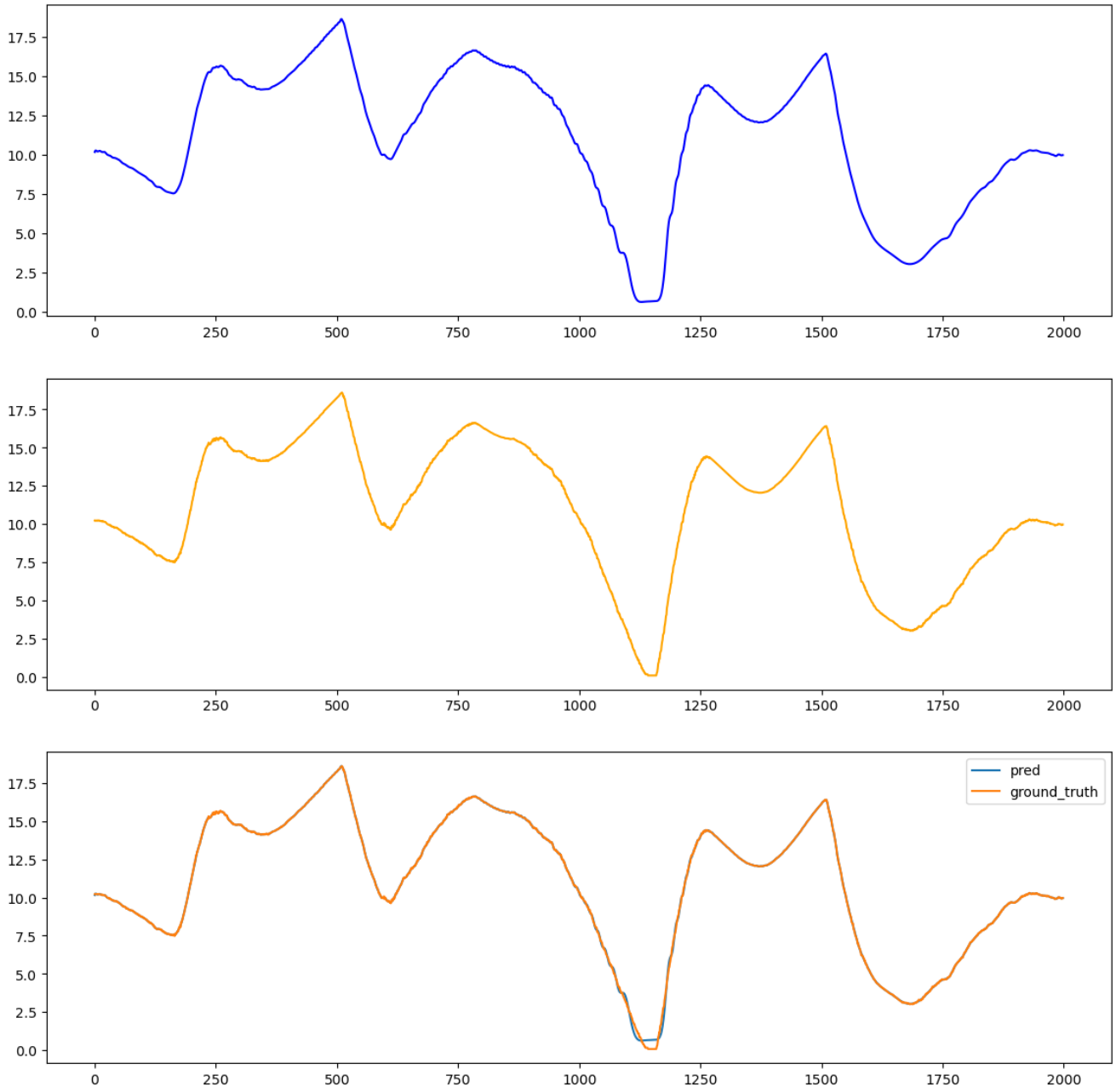


Figure 30: Predicted velocity curve for 1 period

Secondly, the arclength prediction for over 14 days could be done by passing arguments- computed non-linear velocity function as ODE, initial arclength, 2 weeks tspan to the ODE solver. The same has been plotted in fig. 31.

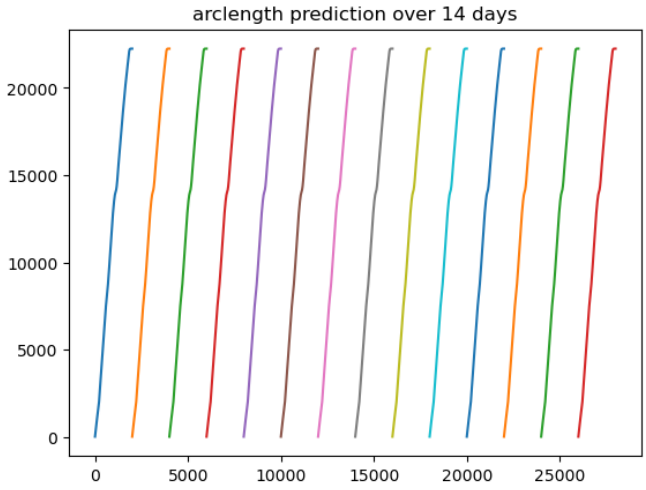


Figure 31: Arclength prediction for over 14 days

Finally, non linearly approximation of function mapping each arclength to a first area measurement was done to compute the linear coefficients. Then, the estimation of MI building utilization (changing arclength in time - 14 days) was done using non-linear approximated function calculated over next two weeks. This has been shown in fig. 32.

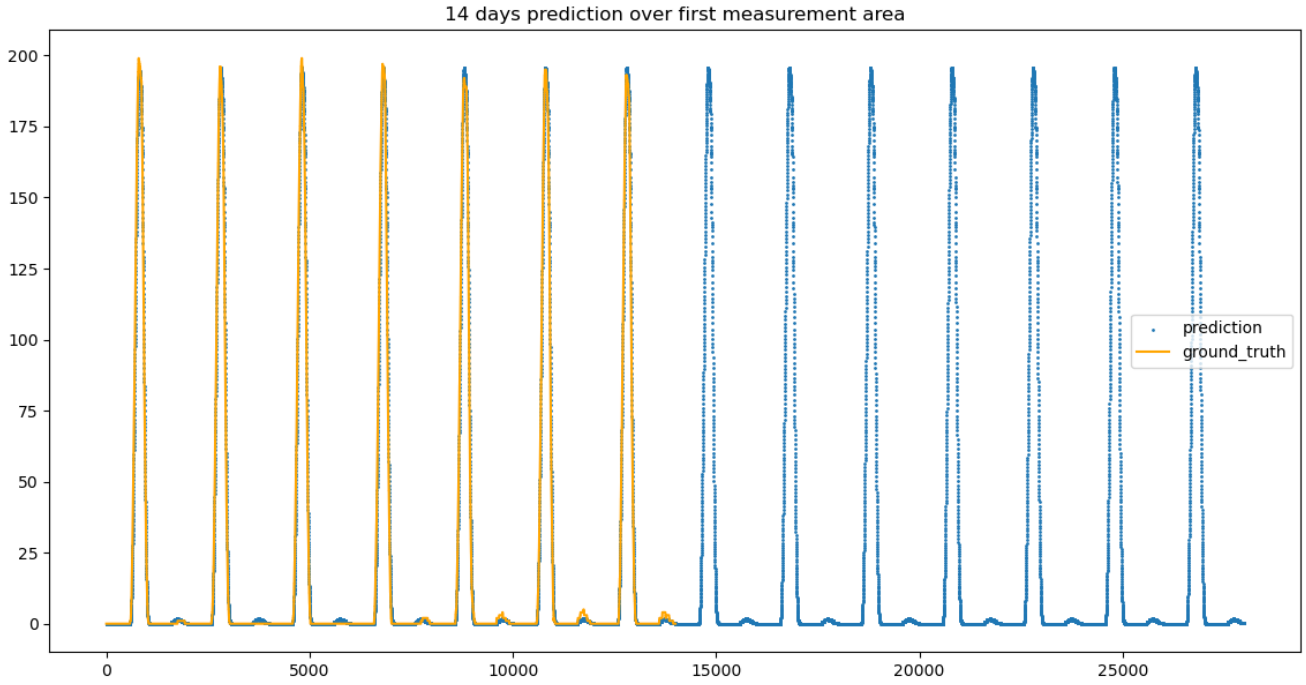


Figure 32: MI utilization prediction for over 14 days