Report for exercise 5 from group G

Tasks addressed: 5
Authors: Zhaozhong Wang (03778350)
Anastasiya Damaratskaya (03724932)
Bassel Sharaf (03794576)
Thanh Huan Hoang (03783022)
Celil Burak Bakkal (03712329)
Last compiled: 2025–01–16

The work on tasks was divided in the following way:

Zhaozhong Wang (03778350)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Anastasiya Damaratskaya (03724932)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Bassel Sharaf (03794576)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Thanh Huan Hoang (03783022)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Celil Burak Bakkal (03712329)	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 explored different approaches to function approximation using both linear and nonlinear methods. Our analysis focused on two distinct datasets to demonstrate the capabilities and limitations of each approach.

Part 1: Linear Data Analysis

For our first analysis, we examined a dataset that exhibited strong linear characteristics. Our approach was to apply a linear approximation method using least squares minimization. The results revealed a linear coefficient of 0.75000024, demonstrating that the underlying function closely follows f(x) = 0.75x.

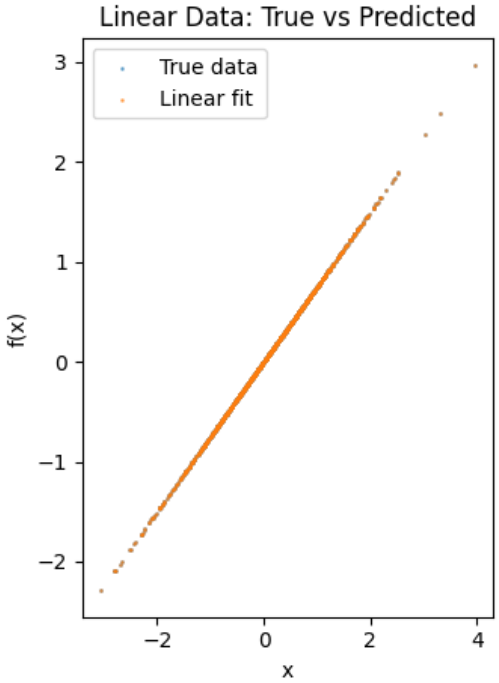


Figure 1: Linear data approximation showing perfect alignment between true data (blue) and linear fit (orange). The overlap is so precise that the two plots are nearly indistinguishable, demonstrating the effectiveness of linear approximation for this dataset.

The precision of our approximation was remarkable, achieving a mean squared error (MSE) of approximately 1.06×10^{-10}. As shown in Figure 1, the linear fit perfectly overlaps with the true data points, making the two plots nearly indistinguishable. This exceptionally low error and perfect visual alignment validated that our choice of a linear approximation method was ideal for this dataset. The plot spans from approximately -2 to 4 on the x-axis and -2 to 3 on the y-axis, showing consistent linear behavior across the entire range of our data.

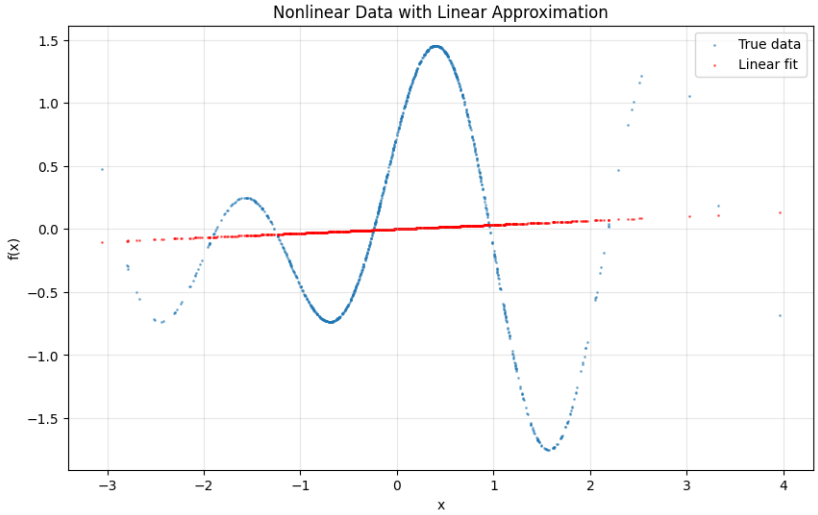
Part 2 & 3: Nonlinear Data Analysis

The second phase of our investigation provided a more challenging scenario with nonlinear data. We approached this in two steps to demonstrate the limitations of linear approximation and the power of nonlinear methods.

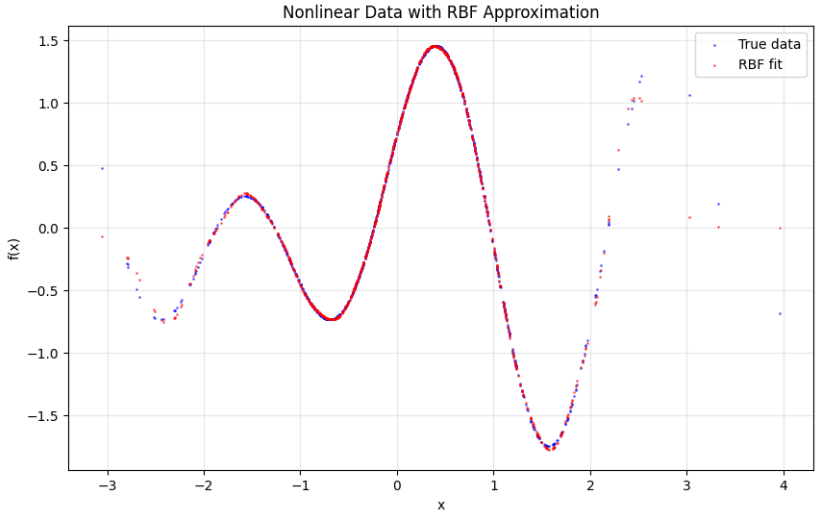
First, we deliberately applied a linear approximation to the nonlinear data. As shown in Figure 2(a), this attempt resulted in poor performance. The linear fit produced a coefficient of 0.03321036 and a high MSE of 0.77489199. The visual representation clearly shows how the linear approximation (red line) fails to capture the wave-like pattern of the true data (blue points).

Second, we implemented a Radial Basis Function (RBF) approximation. After careful consideration, we selected our parameters with 50 basis functions (L = 50) and a bandwidth parameter $\epsilon = 0.5$. The results, illustrated in

Figure 2(b), show that the RBF approximation achieved an MSE of 0.0020501213, which is dramatically better than the linear approximation's MSE of 0.77489199.



(a) Linear approximation of nonlinear data



(b) RBF approximation of nonlinear data

Figure 2: Comparison of approximation methods on nonlinear data. (a) Shows the poor fit of linear approximation, with the red line unable to capture the wave-like pattern. (b) Demonstrates the superior performance of RBF approximation, where the red fit closely follows the true data pattern.

The quantitative improvement was substantial: the RBF approach achieved an MSE of 0.0020501213, representing a 379-fold improvement over the linear approximation. This significant reduction in error demonstrates the effectiveness of the RBF method for handling complex, nonlinear relationships.

Discussion of Parameter Selection

Our choice of RBF parameters was guided by several considerations. The number of basis functions (L = 50) was selected to provide sufficient flexibility to capture the nonlinear patterns while avoiding overfitting. The bandwidth parameter $\epsilon = 0.5$ was chosen to allow each basis function to influence an appropriate local region of the input space.

We deliberately avoided using RBF for the linear dataset despite its power. This decision was based on the

principle of parsimony - when a simpler model (linear approximation) perfectly captures the relationship, using a more complex model (RBF) would only introduce unnecessary computational overhead and the risk of overfitting.

Conclusion

Our analysis clearly demonstrates the importance of selecting appropriate approximation methods based on the underlying structure of the data. For linear relationships, simple linear approximation provides optimal results with computational efficiency. However, when dealing with nonlinear patterns, more sophisticated approaches like RBF become necessary. The dramatic improvement in MSE when switching from linear to RBF approximation for our nonlinear dataset (423x better) quantifiably demonstrates this principle.

The RBF method's success lies in its ability to adapt to local patterns through its Gaussian-like basis functions, allowing it to capture complex behaviors that linear approximations fundamentally cannot represent. This flexibility makes it an invaluable tool for nonlinear function approximation, though it should be applied judiciously when simpler methods would suffice.

Required Packages and Setup

For reproducibility, our implementation required the following Python packages:

- NumPy for numerical computations
- Matplotlib for visualization
- SciPy for least squares optimization

We implemented several key functions in our utils.py file:

- `linear_solve(x, y, rcond)`: Computes the least-squares solution using numpy's lstsq solver. The rcond parameter was crucial for numerical stability in solving the system $Ax = b$.
- `rbf(x, centers, eps)`: Implements the radial basis function computation using the formula:

$$\phi(x) = \exp(-\|x - c\|^2/\epsilon^2) \tag{1}$$

  where c represents the centers and ϵ is the bandwidth parameter.
- `compute_mse(y_true, y_pred)`: Calculates the mean squared error between true and predicted values using:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{true,i} - y_{pred,i})^2 \tag{2}$$

- `load_function(filename)`: Handles data loading and preprocessing, returning input features X and target values y from our data files.

All computations were performed using standard numerical methods, with careful attention to numerical stability through appropriate regularization in the least squares solver (rcond parameter). For the RBF implementation, we ensured proper handling of array dimensions and efficient computation of pairwise distances through vectorized operations.

Report on task 2: Approximating Linear Vector Fields

In this task, we're given a dataset where there are 1000 pairs of 2D points, and each pair contains the starting point and the end point of a dynamical system, with the time interval between these 2 points being $\Delta t = 0.1$. Throughout this task, our goal is to learn the flow of the dynamical system based on the given data and predict the trajectory of the states given a random starting point.

Part 1: Learning the Weights of the Linear Model

First we want to get the samples of our learning objective, which is the vector representing the flow of each starting point. In this task we use the naive Euler method, obtaining the vectors $\{\hat{v}^{(k)}\}_{k=1,\dots,N}$ via dividing the difference between two subsequent states by the time interval Δt. The formula is shown below:

$$\hat{v}^{(k)} = \frac{x_1^{(k)} - x_0^{(k)}}{\Delta t} \tag{3}$$

For simplicity, now we use a $N \times 2$ matrix $\hat{v}$ to denote the vectors $\{\hat{v}^{(k)}\}_{k=1,\dots,N}$, where each vector $\hat{v}^{(k)}$ is the k-th row of the matrix. Similarly, we use x_0 and x_1, both of shape $N \times 2$, to represent the point sets $\{\hat{x_0}^{(k)}\}_{k=1,\dots,N}$ and $\{\hat{x_1}^{(k)}\}_{k=1,\dots,N}$.

In this task it's indicated that the relationship between a point and its vector is linear, so after getting $\hat{v}$, we treat x_0 as the sample points and $\hat{v}$ as the results, and we assume their relationship is approximately $\hat{v} = Ax_0$. Then, using the famous linear least square formula $A = ({x_0}^T x_0)^{-1} x_0^T \hat{v}$, we get the result of A, which is $\begin{bmatrix} -0.494 & -0.464 \\ 0.232 & -0.957 \end{bmatrix}$

Part 2: Checking the Accuracy

To test the accuracy of the model, we compute the mean squared error (MSE) between predicted $\hat{x_1}$ and the actual x_1. The predictions x_1 is calculated via `solve_ivp` method in `scipy.integrate`, and we set the parameter `t_eval` of this method as `None`, meaning that the time intervals are automatically selected by the solver. Here each single error in MSE is the squared L2 distance between one predicted $\hat{x_1}^{(k)}$ and its actual value $x_1^{(k)}$. The formula is shown below:

$$MSE = \frac{1}{N} \sum_{k=1}^{N} \left\| \hat{x_1}^{(k)} - x_1^{(k)} \right\|^2$$

The MSE result is around 10^{-5}, which means that our model is fairly reliable.

Part 3: Inference

Now we want to use our results so far to draw the whole vector field in the domain $[-10, 10]^2$, and plot the trajectory of the states starting from $(10, 10)$, with $\Delta t = 0.1$ and $T_{end} = 100$. The result is shown in Figure 3

In this figure we can see that the system is a stable system with one fixed point. Theoretically, a linear system $\dot{x} = Ax$ is asymptotically stable if all eigenvalues of A have strictly negative real parts, while asymptotically stable means the trajectories of the system will converge to the equilibrium point (usually the origin) as $t \to \infty$. The eigenvalues of our A is $[-0.725 + 0.232j,\ -0.725 - 0.232j]$, whose real parts are both strictly negative. This is in harmony with our theoretical analysis.

Report on task 3: Approximating Nonlinear Vector Fields

In this task, we use a dataset containing N=2000 rows and four columns for N data points x_0 and x_1 in two dimensions, as demonstrated in Figure 4. The first two columns contain initial points, while the other two have the same points advanced with an unknown evolution operator ψ. The task is to examine the underlying dynamics by comparing the results of linear and radial basic functions approximation of the vector field, as well as identifying the steady states of the system.

Part 1: Vector Field Estimation With a Linear Operator

We start with estimating the vector field similar to the previous task with a linear operator $A \in \mathbb{R}^{2\times 2}$. In order to approximate A, we apply least-squares minimization such that:

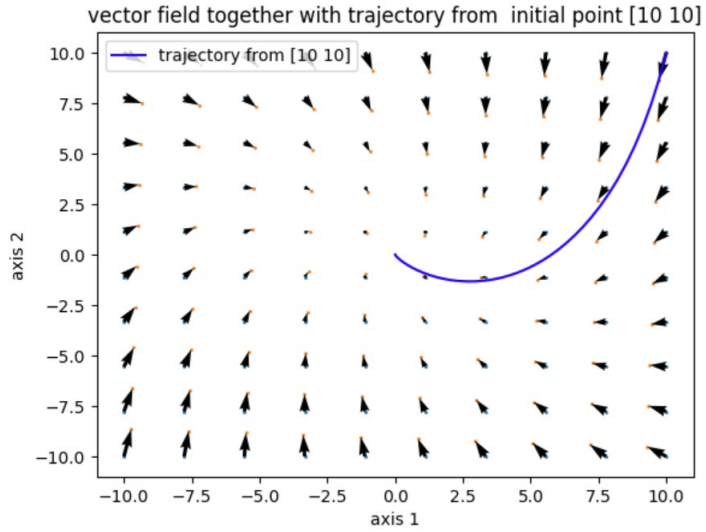


Figure 3: Vector field in $[-10, 10]^2$ and the trajectory starting from state $(10, 10)$

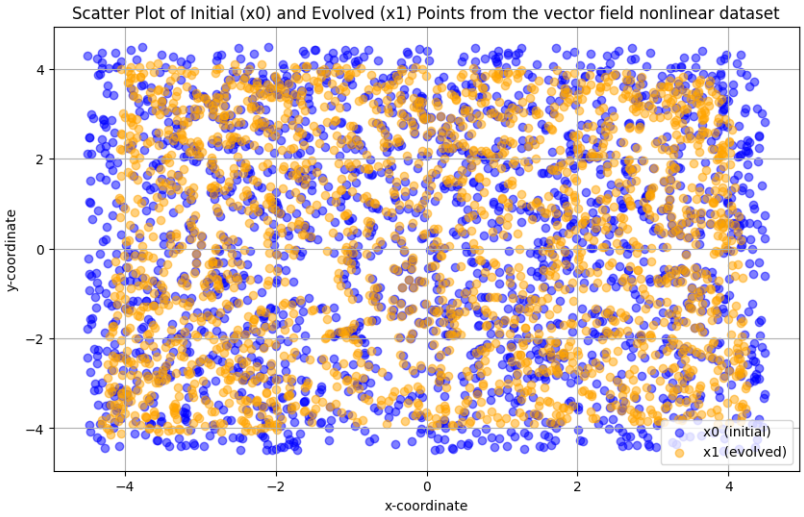


Figure 4: Scatter plot of initial and evolved points over the domain $[-4.5, 4.5]^2$.

$$\frac{d}{ds}\psi(s, x(t))\Big|_{s=0} \approx \hat{f}_{\text{linear}}(x(t)) = Ax(t). \tag{4}$$

We use linear approximator for our system with $\Delta t = 0.01$ and receive $A = \begin{bmatrix} -1.002 & 0.087 \\ -0.025 & -4.327 \end{bmatrix}$. With the returned linear operator A, we can solve the equation above, which implies that the change in $x(t)$ over a small time Δt is directly influenced by A, resulting in evolved end points being computed as

$$\hat{x}_1^{(k)} = x_0^{(k)} + \Delta t \cdot Ax_0^{(k)} \tag{5}$$

for all N initial points after Δt. In order to determine the accuracy, we again compute MSE for known and predicted points, which results in 0.037287. The MSE is relatively small, meaning the linear approximation reasonably fits the dataset. However, it is not very close to 0, indicating that it does not describe the evolution operator ψ very well, which can be a sign of nonlinearity.

Part 2: Vector Field Approximation Using Radial Basic Functions

Now, to compare the estimations, we approximate the vector field using radial basic functions (RBF). First, we need to decide the number of centers L and the scaling parameter ϵ in order to construct the basis. Since a low L value may lead to underfitting because it cannot capture all the complexity, while a significant L value may lead to overfitting, we perform a grid search over a range of values between 100 and 1000 in intervals of 10 for L, as suggested in the task. For ϵ, we use `np.logspace` function to generate a values range that spans several orders of magnitude. After defining the ranges, we iterate over the values of L and ϵ and predict the end points. After receiving the evolved points, we compute MSE and store the results in the list for later analysis, which are visualized in Figure 5. Based on finding the minimum MSE, the optimal $L = 940$ and the optimal $\epsilon = 2.154$ with $MSE = 2.966 \times 10^{-15}$.

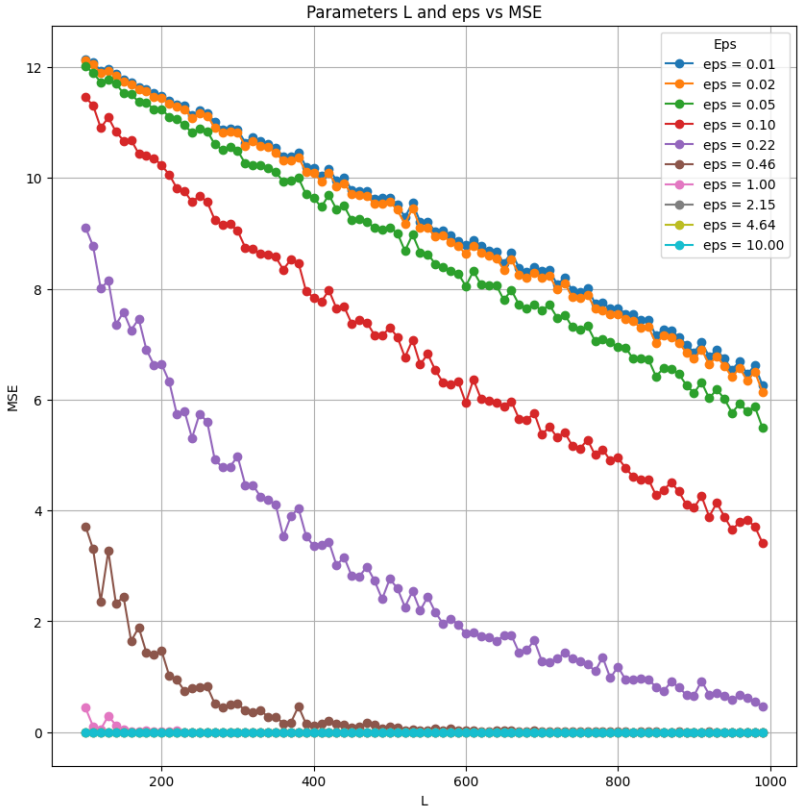


Figure 5: Relations between parameters L and ϵ and the MSE results.

For a more general analysis, we use the RBF approximator to estimate the vector field and set the approximator parameters to the identified optimal values. Then, we split the original dataset into train (80%) and test (20%) sets in order to evaluate the performance of the system independently and detect possible under- or overfitting. We furthermore split x_0 (input data) and $\frac{dx}{dt}$ (output data) for consistency reasons between train and test sets. After fitting the approximator with the train set, we predict the end points using the test set and, once again, compute MSE, which results in $MSE = 4.32 \times 10^{-11}$. Compared to the linear approximation, the MSE of RBF approximation is extremely low and indicates that the RBF approximation successfully models the vector field with near-perfect accuracy. Since linear approximation does not achieve comparable MSE results, the vector field appears to be nonlinear.

Part 3: Steady States of the System

To address part 3 of the task, we will use the approximated vector field associated with ψ resulting from RBF approximation to simulate the system's behavior over a more significant time horizon and determine its steady states. The vector field $v(x)$ is defined in Equation 3. The RBF centers c_j for the approximation are selected

from x_0 using random sampling and, after constructing the basis and computing the approximator's weights, we proceed with the system simulation. We iteratively apply the approximated vector field, starting from x_0, to predict the system's progress over a more significant time horizon, setting the number of timesteps = 1000. The next points in each iteration are computed as follows:

$$x_{next} = x_{current} + v(x_{current}) \cdot \Delta t. \tag{6}$$

At the end of the iteration, we evaluate $v(x)$ at the final points of the simulation and sort out the points close to zero using Euclidean norm:

$$\|v(x_{final})\|_2 < \epsilon_{\text{tolerance}},$$

where $\epsilon_{\text{tolerance}} = 1e-3$ is a small tolerance value. This step helps to identify the steady states of the system since the steady states are points where the vector field $v(x)$ approaches zero, indicating no further evolution. After determining the steady states, we round up the values to three decimals and sort out unique ones. As a result, we receive four unique steady states: [-3.779, -3.283], [-2.805, 3.131], [3, 2], and [3.584, -1.848].

Determining the steady states provides insights into the underlying system's stability and overall structure. Since we detected multiple steady states in the system, we can conclude that our system is complex and nonlinear. The existence of multiple steady states further plays a significant role in identifying whether a nonlinear system can be topologically equivalent to a linear system. A linear system usually has at most one steady state, typically at the origin or another fixed point, while multiple steady states often result from nonlinearity, which cannot be replicated as a linear system. This indicates that, in general, a nonlinear system cannot be topologically equivalent to a linear system, as in our case.

Report on task 4: Time-Delay Embedding

Part 1: Time-Delay Embedding of a Periodic Signal

In this task, we examine the concept of time-delay embedding by analyzing a periodic signal. Time-delay embedding is a technique used to reconstruct the state space of a dynamical system from a single time series, enabling the analysis of its underlying structure and behavior.

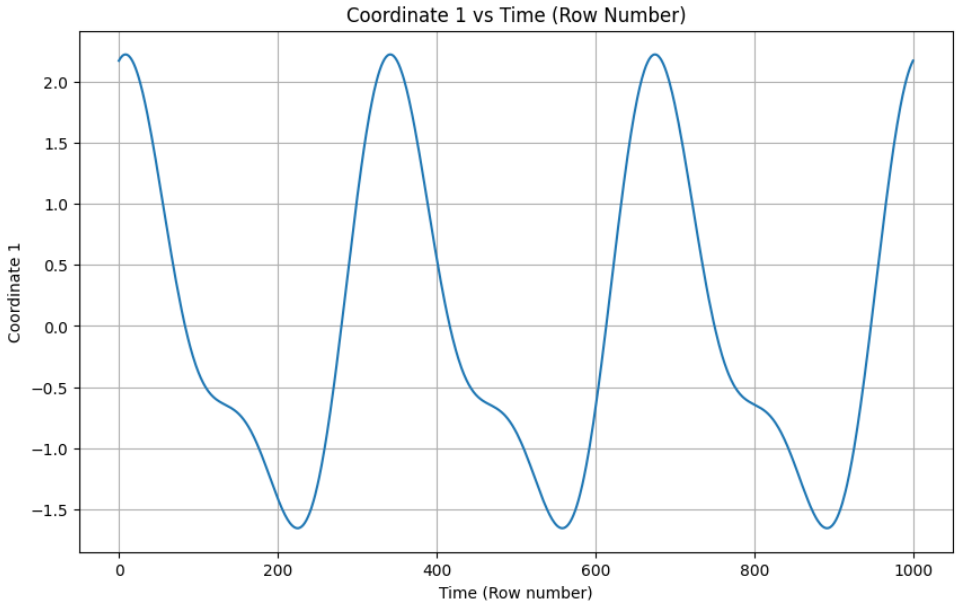
Figure 6 presents two plots illustrating this process:

- Figure 6a: This plot displays the first coordinate of the signal against time, represented by the row number in the dataset. It provides a direct visualization of how the signal progresses over time.
- Figure 6b: Here, the first coordinate is plotted against its delayed version, with a delay ($\Delta n = 10$) of 10 time steps. This delayed plot is crucial for constructing the time-delay embedding, as it captures the system's state at a previous time, allowing for the reconstruction of the phase space.

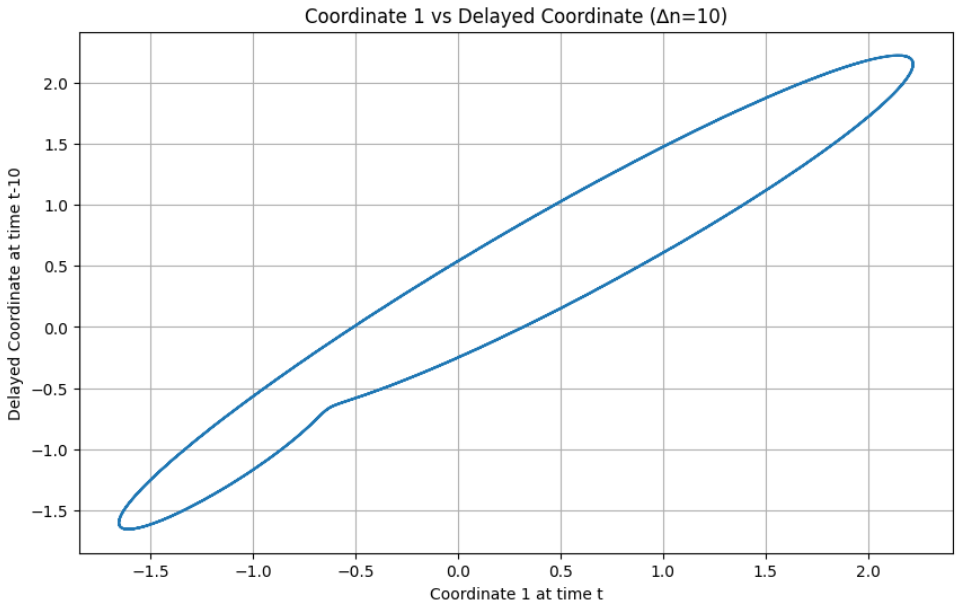
According to Takens theorem, the number of coordinates (or embedding dimension) required to correctly embed a periodic manifold is determined through the formula $2d + 1$, where d is the dimensionality of the attractor (i.e., the manifold's intrinsic dimension). In our case of a periodic manifold, $d = 1$, which results in $2 \times 1 + 1 = 3$ coordinates being needed.

Part 2: Testing Takens Theorem on the Lorenz Attractor

In this task, we explore the application of Takens theorem to chaotic dynamics by approximating the Lorenz attractor from a single time series with the parameters $\sigma = 10, \quad \rho = 28, \quad \beta = \frac{8}{3}$. The plots were generated by solving the Lorenz system of differential equations using the `solve_ivp` function, with a starting point of $(x_0, y_0, z_0) = (10, 10, 10)$ and a time span from $t = 0$ to $t = 100$. Δt, the value for the delay in embedding, was chosen to be 10. The system was evaluated at 10.000 evenly spaced time points. The solution was then used to extract the x, y, and z coordinates over time. Additionally, the attractor was reconstructed from both the delay embeddings of x-coordinates and the delay embeddings of z-coordinates to visualize the system's chaotic behavior and the evolution of its dynamics. Figure 7 shows these plots.



(a) First coordinate against time



(b) First coordinate against its delayed version ($\Delta n = 10$)

Figure 6: Plot of the first coordinate from `periodic.txt` against (a) time (row number), and (b) its delayed version ($\Delta n = 10$)

As we can see, the reconstructed Lorenz attractor from the x-coordinate closely resembles the original, specifically having an 8-shape, while the attractor reconstructed from the z-coordinate shows a completely different shape, resembling a semicircle. The z-coordinate probably fails to capture the system's dynamics as it provides less independent information compared to x. Looking at the original plot, the z-coordinate has a narrower range compared to x and y, limiting its ability to capture the system's complexity. It also does not effectively separate the two lobes of the attractor, resulting in overlapping trajectories and an incomplete representation. Given the system's sensitivity to initial conditions, the z-coordinate lacks the necessary separability of trajectories over time, causing the attractor to appear less distinct.

On the other hand, the x-coordinate might have (partially) succeeded in reconstruction because it exhibits

strong fluctuations and plays a key role in the chaotic behavior. While it may not capture all of the system dynamics, using the x-coordinate alone could still offer a reasonable approximation of the attractor's structure.

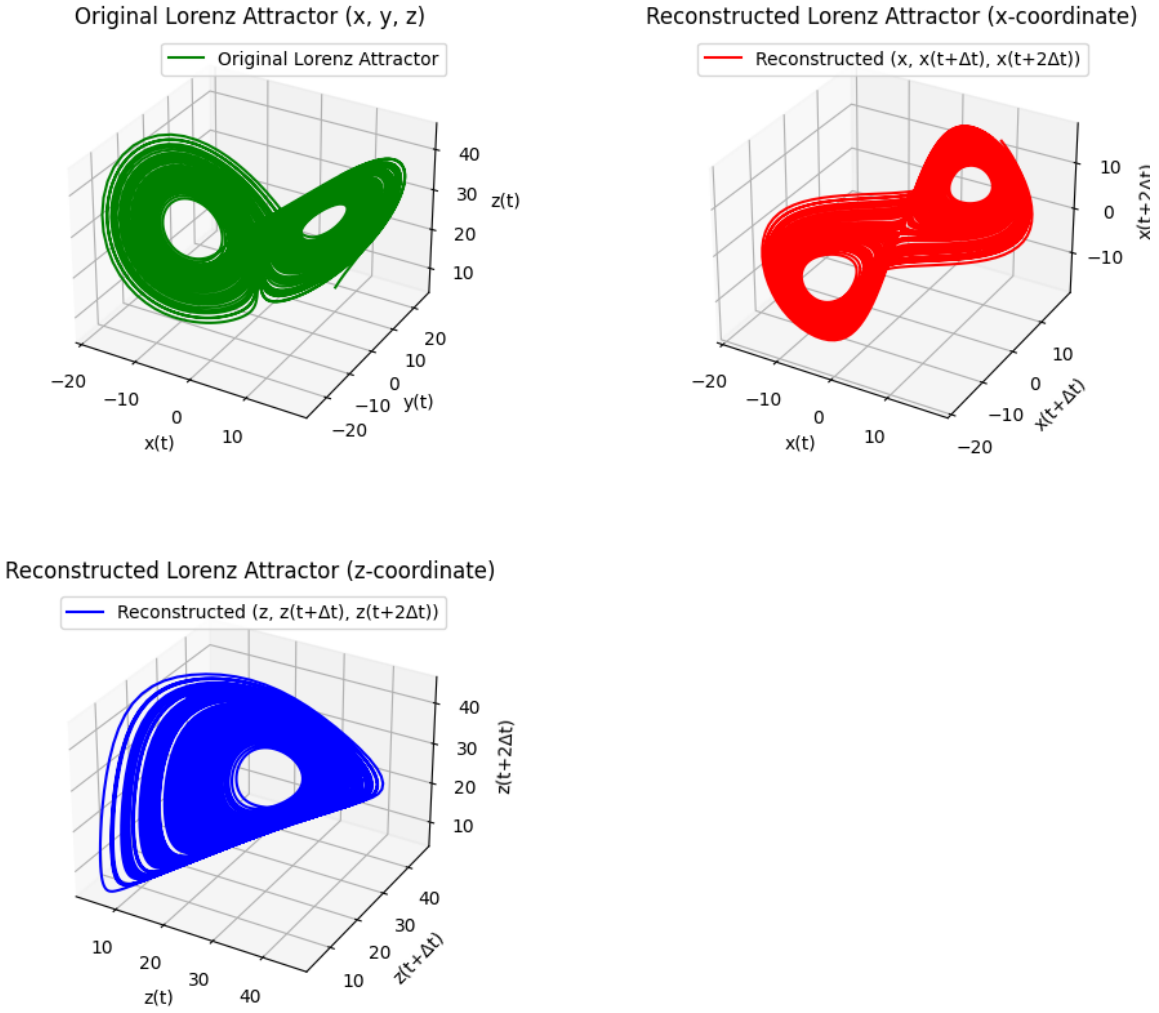


Figure 7: Lorenz attractor (a) with its original values (b) reconstructed from the x-coordinate ($\Delta t = 10$) (c) reconstructed from the z-coordinate ($\Delta t = 10$)

Bonus

The new state space for the Lorenz attractor with the x-coordinate is created by setting the values of x, y, and z as $x_1(t) = x(t), \quad x_2(t) = x(t + \Delta t) = y(t), \quad x_3(t) = x(t + 2\Delta t) = z(t)$., stored in `x_dataset` Δt still has the same value of 10. The state of the system at the next time step is constructed by shifting `x_dataset` by one time step forward, with the first row initialized to zero., stored in `x_dataset_next`.

The core objective is to approximate the vector field $\hat{\nu}(x_1, x_2, x_3)$, which represents the system's dynamics, using Radial Basis Functions (RBF). To achieve this, we use the state data in `x_dataset` and the corresponding difference between consecutive states, denoted as `diff = x_dataset_next - x_dataset`. This difference represents the change in the system's state between time steps and is used to train the RBF model using the provided `fit()` method, learning the relationship between the state variables `x_dataset` and the vector field `diff`, capturing the dynamics of the system.

The values for L and ϵ are 15 and 9.5, respectively. L controls the smoothness of the RBF kernel, the spread of influence of each data point, and represents the number of centers used in the RBF network. A value of $L = 15$ ensures a balance between capturing sufficient detail and avoiding overfitting the chaotic behavior of the Lorenz

attractor. This moderate choice reflects the scale of the Lorenz system's dynamics. The kernel's influence should be large enough to capture the non-linear relationships between state variables without over-smoothing. As for ϵ, its value influences the smoothness of the approximation. A small ϵ makes the RBF more localized and sensitive, potentially leading to overfitting, while a large ϵ makes the RBF more global, smoothing out the vector field but possibly missing finer details. An ϵ value of 9.5 was chosen to balance between avoiding overfitting and providing a good approximation of the vector field.

Figure 8 shows the reconstruction of the time-delayed x-coordinates from the Lorenz attractor. Although it does not exactly match and appears less "dense" than the original, the 8-shape formed is still recognizable, which is also present in the original graph.

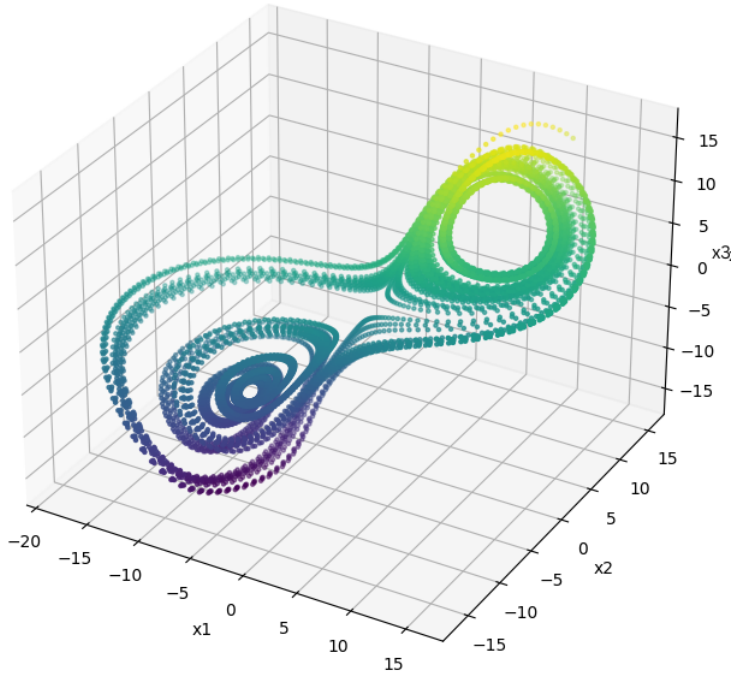


Figure 8: Reconstruction of the x-coordinate time-delay embedding of the Lorenz attractor after approximating its vector field using radial basis functions.

Report on task 5: Learning Crowd Dynamics

All the code and figures mentioned and used in this task can be found in the file `task5.ipynb`.

Part 1: Time-Delay Embedding and PCA

In this part, we use time-delay embedding to create a reasonable state space of the given system.

First we load the file `MI_timesteps.txt` and check its shape and see that it has a shape of (15001, 10), where first column corresponds to timesteps and the rest are the 9 measurements mentioned in the task description.

After that, we create time delay embeddings with 350 delays of the first three measurement areas, which correspond to the columns 2,3,4 in the file `MI_timesteps.txt`. With this we get "windows" of 351×3 coordinates and then we turn these into vectors of length $351 \times 3 = 1053$ dimensions. For this we use the class `TimeDelayEmbedding` in the file `time_delay.py` which we initialize with the parameter `time_delay` $= 350$. After that we pass the data, which is created by filtering out the columns 2,3,4 of `MI_timesteps.txt`, to the method `transform` of `TimeDelayEmbedding` class. After this, we receive the 1053 dimensional vectors that are created with time delay embeddings.

Furthermore, we apply PCA on that data set of 1053 dimensional vectors and use as many principal components as necessary according to Takens. According to Takens theorem, the number of embedding dimensions required to correctly embed a periodic manifold is determined through the formula $2d + 1$, where d is the dimensionality of the manifold (i.e., the manifold's intrinsic dimension). The data has 9 measurement areas so it has actually 9 dimensions but as it is said in the task description, system is periodic and has no parametric dependence, so it probably will be a one-dimensional, closed loop. So in this case of a periodic manifold we have $d = 1$, which results in $2 \times 1 + 1 = 3$ dimensions being needed. When we plot with these 3 PCA dimensions, we get the 3D embedding representation of data in Figure 9.

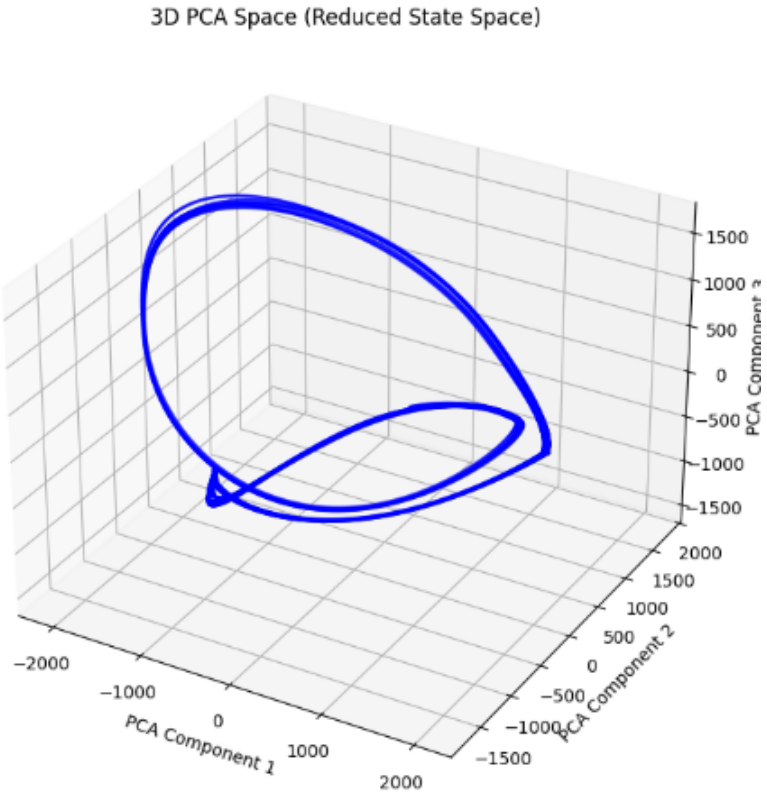


Figure 9: 3D PCA Embedding Space of MI data

Part 2: Coloring of Points by All Measurements

In the 3D embedding space we created in the first part, there are now many different points. In this part, we create 9 plots where these points are colored by the 9 measurement columns in the data of `MI_timesteps.txt`. Note that points are in the same position, only the coloring changes. You can see this in Figure 10. When we inspect these plots, we see that all the 9 observations are functions over our new 3D representation even though in the first part we only used 3 of the observations to create the embedding space. So other 6 observations are also functions of the first 3.

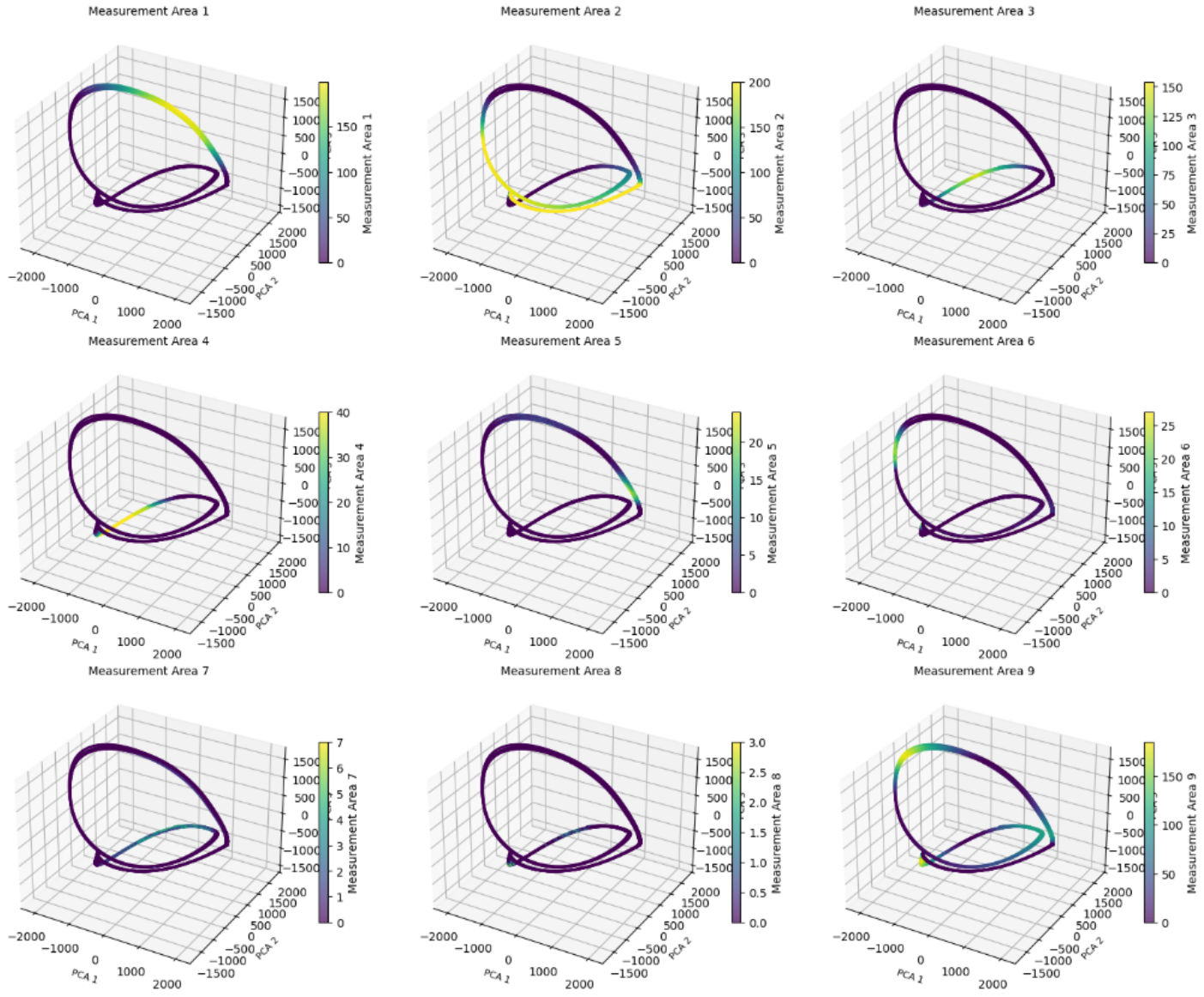


Figure 10: Embedding Coloring All Together

Part 3: Learning the Dynamics

In this part, we aim to learn the dynamics on the periodic curve we embedded in the principal components. For this reason, we first compute the arclength of the curve in the PCA space and then approximate the change of arclength over time.

For computing the arclengths we first compute the differences between successive points in 3D PCA embedding space along each of the 3 dimensions and then compute Euclidean distances between successive points based on these values, which represents the arclength between succesive points. Finally, we compute the cumulative sum of these arclengths until each point, which gives us the traversed cumulative arclength at each point.

For computing the velocity (change of arclength over time) we follow 2 different ways. First way is directly computing gradients on the previously described arclengths, which we plot in Figure 11.

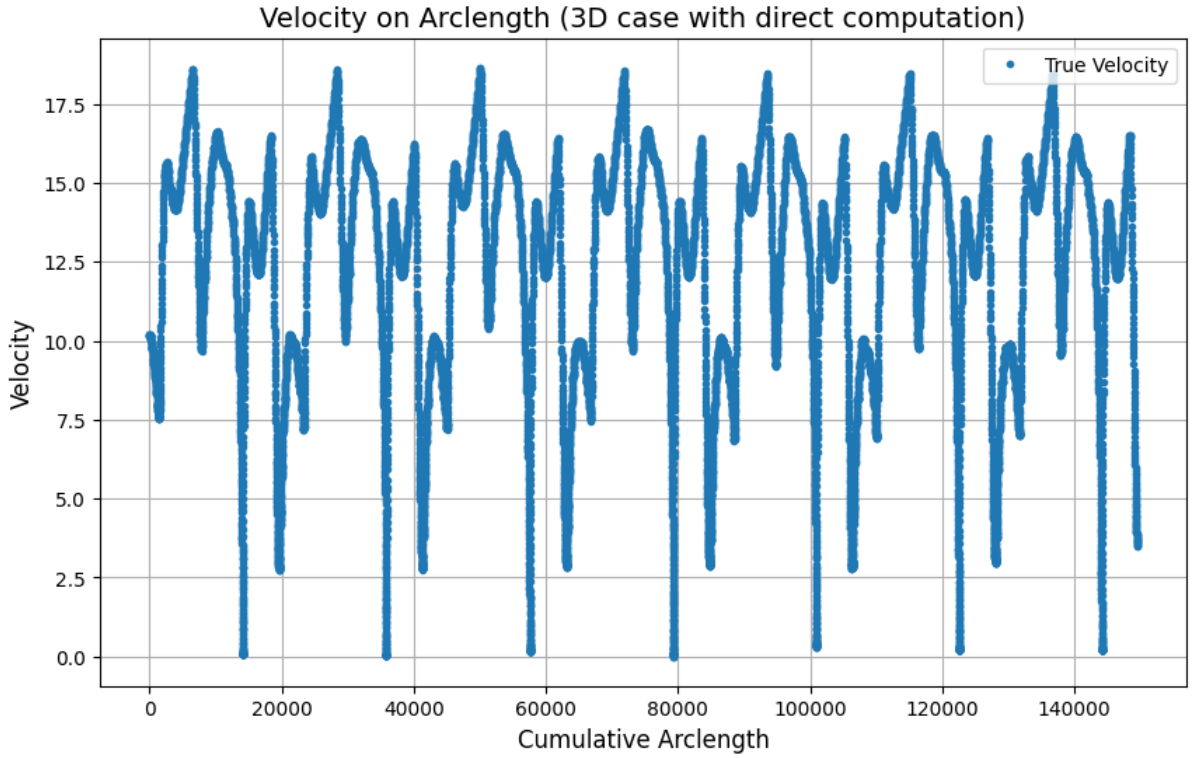


Figure 11: Direct Velocity with Cumulative Arclength

Second way is dimension-wise velocity computation. In this way, we first take the differences of successive points along each of 3 dimensions and then compute the gradients for each of the 3 dimensions of differences, which gives us the velocities along each of the 3 dimensions. The reason why we do this is that if we directly use the direct arclength velocity described in the first way for learning the first measurement area (utilization of the MI building), it can happen that the same velocity values correspond to different measurement values. Therefore, learning results are better when we use the dimension-wise computed velocities described in the second way. In order the show the numerical equality of the 2 ways, we apply the Euclidean formula with these 3-dimensional velocity values and draw a plot of them, which is demonstrated in Figure 12. After that we compute the mean squared error (MSE) between the directly computed velocities and dimension-wise computed velocities (after application of Euclidean formula) which results in the value $7.091801279131613 \times 10^{-8}$. The nearly identical plots in Figure 11 and Figure 12 and the very low value of MSE shows us the numerical equivalence of the 2 approaches.

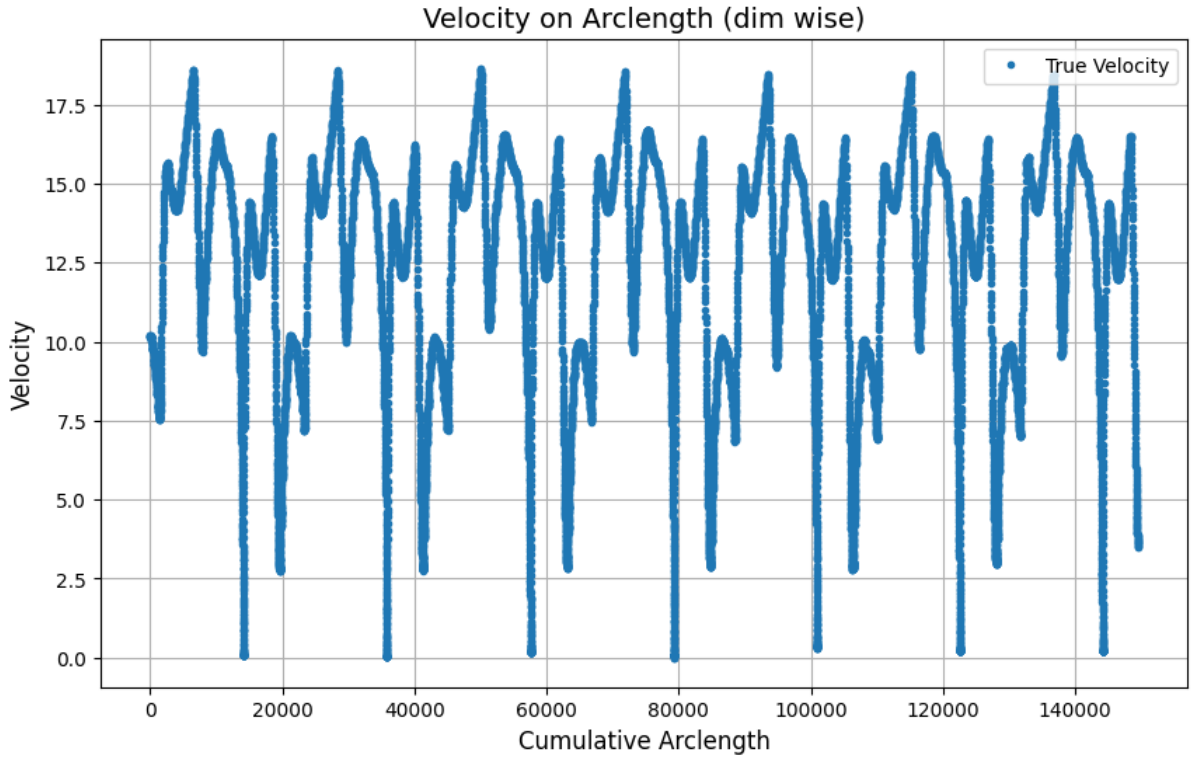


Figure 12: Dimension-wise Velocity with Cumulative Arclength

You can see the plot of dimension-wise computed velocities in Figure 12, where horizontal axis represent cumulative arclength and vertical axis represents velocity at the corresponding arclength values. Note that in Figure 11 and Figure 12 periodicity of the embedding is not taken into account and arclength is computed cumulatively. So, for example if 20000 is the endpoint of closed loop and if we do one more tour on the loop and visit this position again it is represented as 40000, not 20000 again. Also it is not mapped to range $[0, 2\pi]$ like in the Figure 4 of exercise sheet. We will do these later on.

When we inspect Figure 11 and Figure 12, we can see the periodicity of our embedding being reflected with velocity. There is a repeating pattern every 20000 units of arclength, so from this we can infer that one cycle of our closed loop embedding in Figure 9 takes 20000 units in arclength. In Figure 11 and Figure 12 we recognize that the same pattern is repeated 7 times, which also suits the statement in exercise sheet that `MI_timesteps.txt` file contains data of 7 weekdays.

Part 4: Predict the utilization of the MI building

In this part, our goal is to predict the utilization of the MI building, which correspond to the first column in the file `MI_timesteps.txt` after the time steps, for the next 14 days. To have an overview, we first plot the values of this first measurement area (utilization values) over time, which can be seen in Figure 13. Here, vertical axis represents the values of the first measurement area (utilization) and the horizontal axis represent their corresponding time-step values. Here we see the periodicity of data and the 7 seven peaks corresponding to 7 days. When we compute the time difference between 2 peaks to find the length of a period, we find out that there are 2000 time-steps between 2 peaks, which means that the length of 1 period in the data is 2000 time-steps.

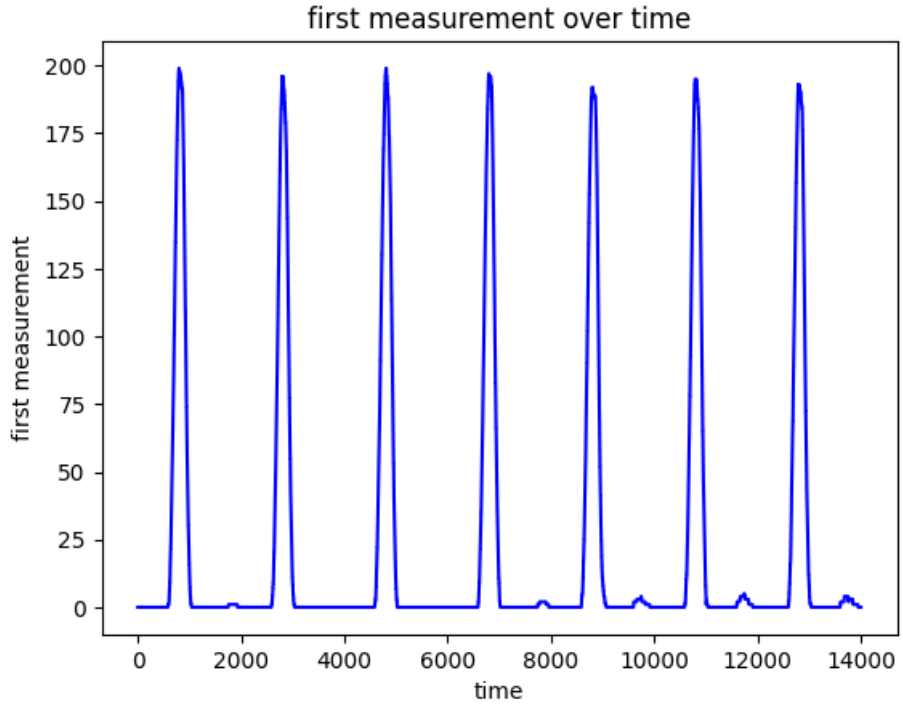


Figure 13: First Measurement Area (Utilization) Values of Given Data

In order to predict the utilization values (i.e. first measurement area) we use radial basis function approximation with the class `RBFApproximator` in the file `approximator.py`.

We first create a `RBFApproximator` with the parameters L=20 and ϵ=1 for approximation from evenly spaced 2000 (corresponds to period length described previously) numbers, over the interval $[0, 2\pi]$ to the first period (i.e. first 2000) of dimension-wise velocities. The numbers in interval $[0, 2\pi]$ represent the mapped arclength values. Note that the interval $[0, 2\pi]$ is also used in Figure 4 of exercise sheet and aligns with the periodicity and circular behaviour of our data.

Subsequently, with our `RBFApproximator`, we predict the velocity values for evenly spaced 10000 numbers over the interval $[0, 2\pi]$. Here the high number 10000 is chosen for fine-sampling. The plot of the predicted dimension-wise velocities (i.e. v1, v2, v3) and their comparison with true dimension-wise velocities is demonstrated in Figure 17. In these plots blue dots represent the true velocity values and orange lines represent the predicted velocity values.

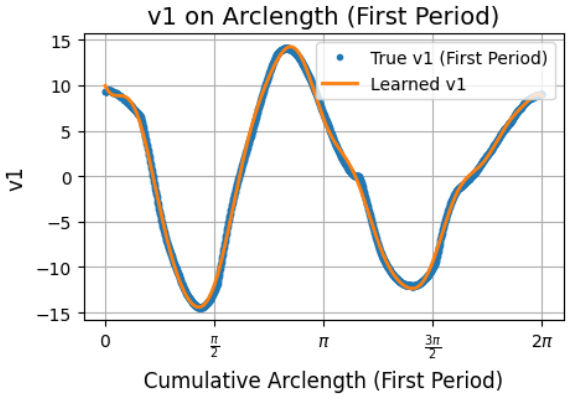


Figure 14: Plot 1

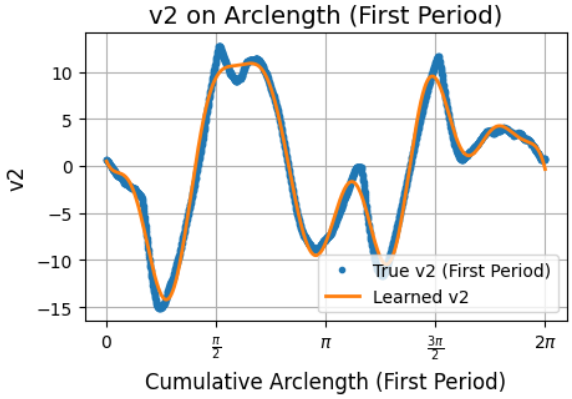


Figure 15: Plot 2

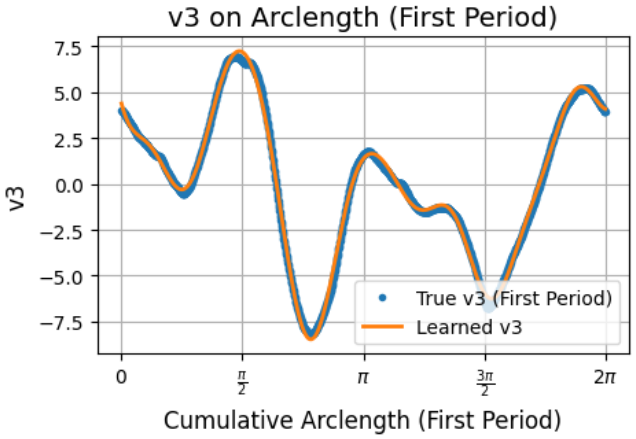


Figure 16: Plot 3

Figure 17: Comparisons of Predicted and True Dimension-wise Velocities

After creating `RBFApproximator` for predicting velocity values, we create another `RBFApproximator` with the values L=500 and ϵ=1 for approximation from arclength velocity values to the first measurement area (i.e. utilization) values.

With both of the `RBFApproximators` we created, for choosing the parameters L and ϵ we manually tried various different values, inspected how it effected the plots of predictions and adjusted the values according to these results until we have satisfying plot results.

The parameter L represents the number of RBF centers to use for constructing the basis and controls the smoothness of the RBF kernel, the spread of influence of each data point. We chose the parameter L such that the model has enough capacity to learn the underlying complex relationship between input and target values but not capture unnecessary noise and avoid overfitting.

Furthermore, the parameter ϵ represents the scaling parameter that controls the width of the RBFs and influences the smoothness of the approximation. Smaller values create narrower RBFs, leading to more localized influence. The value of ϵ parameter is chosen such that it does not capture very small unnecessary variations but also does not miss important important information and have a proper local influence.

After creating this `RBFApproximator`, we predict the utilization values of the first period (first day) with it and compare the predictions with the true utilization values of the first period in the given data of the file `MI_timesteps.txt`, which is demonstrated in Figure 18. Here blue dots represent the true utilization values in the given data and orange lines represent the predicted utilization values.

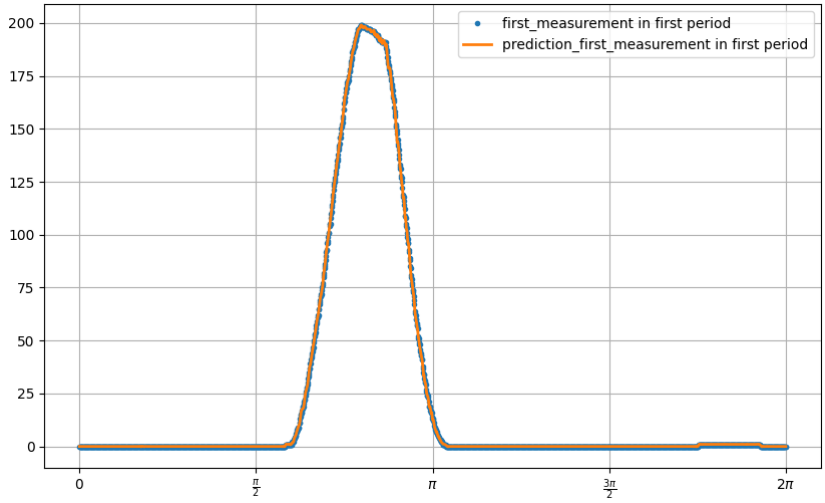


Figure 18: Comparison of Predicted and True First Measurement Area Values in the First Period

Finally, with this `RBFApproximator`, we predict the utilization values for the next 14 days, which is shown in Figure 19. We do that by first predicting velocity values for 14 days with the previous first `RBFApproximator` we created and pass these velocity values to the current `RBFApproximator` and predict the utilization values of 14 days. Here the horizontal axis represents the arclength values and vertical axis represents the predicted utilization values. Note that 1 period (day) is represented with length 2π and here in horizontal axis we have the range $[0, 28\pi]$ for 14 periods (days). The 14 peaks corresponding to the 14 days and the periodic behaviour can be clearly seen here.

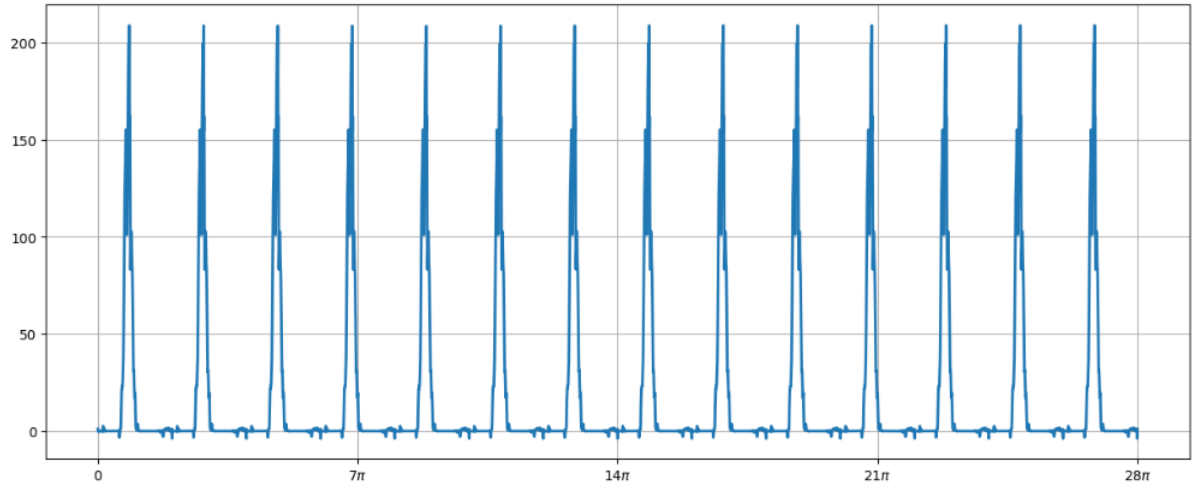


Figure 19: Predicted Utilization (First Measurement Area) Values of MI building for 14 Days