Report for exercise 5 from group J

Tasks addressed:	5
Authors:	Wenbin Hu (03779096) Yilin Tang (03755346) Mei Sun (03755382) Daniel Bamberger (03712890)
Last compiled:	2023–06–26
Source code:	https://github.com/HUWENBIN2024/TUMCrowdModelingGroupJ/tree/main/ex5

The work on tasks was divided in the following way:

Wenbin Hu (03779096)	Task 1	1/3
	Task 2	1/3
	Task 3	1/3
	Task 4	1/3
	Task 5	1/3
Yilin Tang (03755346)	Task 1	1/3
	Task 2	1/3
	Task 3	1/3
	Task 4	1/3
	Task 5	1/3
Mei Sun (03755382)	Task 1	1/3
	Task 2	1/3
	Task 3	1/3
	Task 4	1/3
	Task 5	1/3
Daniel Bamberger (03712890)	Task 1	0
	Task 2	0
	Task 3	0
	Task 4	0
	Task 5	0

Report on task 1, Approximating functions

In this task, our aim is to fit two different data sets, each containing 1000 one-dimensional points with two columns: the first column represents the x-values, and the second column represents the corresponding f(x) values. Dataset(A), named **linear_function_data.txt**, while Dataset(B), named **nonlinear_function_data.txt**.

When examining the distribution of data points, we notice that both datasets have a higher concentration of points in the central range of x-values, specifically within the interval from -3 to 4. As we move towards the edges of this interval, the density of data points decreases, resulting in a sparser distribution. There are three parts in this task and in all parts we have used a least-squares minimization to obtain the matrices A (linear case) and C (nonlinear case) as described in the following equations:

$$\min_{\hat{f}} e(\hat{f}) = \min_{\hat{f}} \|F - \hat{f}(X)\|^2 = \min_{A} \left\|F - XA^T\right\|^2 \tag{1}$$

$$\min_{\hat{f}} e(\hat{f}) = \min_{\hat{f}} \|F - \hat{f}(X)\|^2 = \min_{C} \left\|F - \phi(X)C^T\right\|^2 \tag{2}$$

Part 1: Approximate the function in Dataset(A) with a linear function

In this part, our goal is to approximate the function in Dataset(A) using a linear function. To achieve this, we need to minimize formula1. To solve this minimization problem, we employ the least-squares minimization algorithm provided by the **scipy** library. By applying this algorithm, we obtain an approximated linear function that fits the dataset quite well. In the figure above, we can observe how closely the approximated function aligns with the dataset. In the case of Dataset(A), we determine the slope of the linear function to be approximately 0.75. The resulting linear function provides a good explanation for the data points in Dataset(A), as shown in Fig.1.

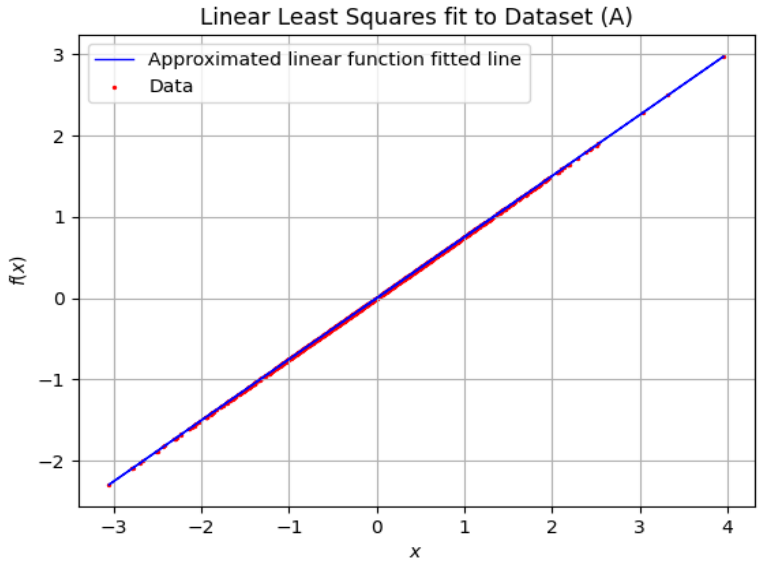


Figure 1: Plot of the linear approximated function over the linear data contained in Dataset(A)

And why it is not a good idea to use radial basis functions for this dataset, since the data in Fig.1 shows a linear pattern, it is clear that using a linear function is the most suitable choice for approximating this linear dataset. Radial basis functions, on the other hand, are better suited for approximating nonlinear datasets.

Part 2: Approximate the function in Dataset(B) with a linear function

In this part, our goal is to approximate the function in Dataset(B) using a linear function. After applying the same procedure, we obtained the slope of the linear function to be approximately 0.03. However, the resulting linear function does not capture the dominant oscillating behavior present in the data as shown in Fig.2.

This is because our dataset is nonlinear, and it is not possible to accurately approximate a nonlinear dataset using a linear function.

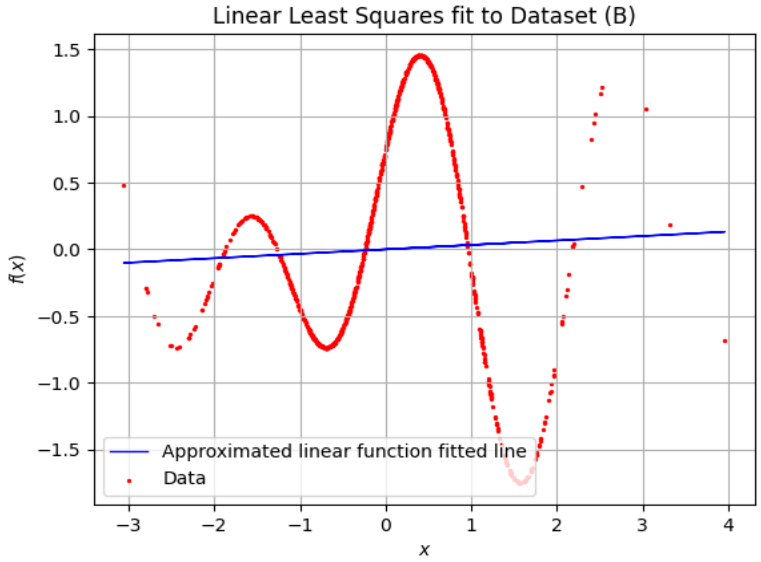


Figure 2: Plot of the linear approximated function over the non-linear data contained in Dataset(B)

Part 3: Approximate the function in Dataset(B) with a combination of radial functions

In this part, we have to approximate the function in Dataset(B) with a combination of radial functions, that turns out to be a nonlinear function. To achieve this, we need to minimize formula2. To improve upon our previous approach we make introduce the radial basis functions:

$$c\phi_l = \exp\left(-\|x_l - x\|^2 / \epsilon^2\right) \tag{3}$$

where the point x_l is the center of the basis function, and the parameter epsilon is linked to the peaked-ness. We use the approach:

$$cf(x) = \sum_{l=1}^{L} c_l \phi_l(x) \tag{4}$$

Since our data is one-dimensional the parameters c_l are in $\mathbb{R}$. The x_l are chosen equally spaced in the range of the minimum and maximum data points of the set.

To ensure an accurate approximation of peaks at a distance of one, it is important for the grid spacing of Gaussian peaks to be at least on a length scale of 1. If the data peaks do not align with the grid, a higher grid density is needed. In our approach, we used a uniformly spaced sampling over the domain $[x_{min}, x_{max}]$ to define our center points (x_l) for the radial basis functions. We employed the **np.linspace**$(x_{min}, x_{max}, \mathbb{L})$ instruction to determine the center points, where $\mathbb{L}$ represents both the number of center points and radial basis functions created. However, having $\mathbb{L}$ equal to the number of data points would lead to over-fitting and poor generalization, interpolation, and extrapolation. In order to achieve a good approximation, we estimated that a grid with $\mathbb{L} = 16$ and a separation of about 0.5 ($\epsilon = 0.5$) would be suitable. This configuration allows for well-localized peaks while maintaining reasonable smoothness and minimal overlap with neighboring peaks.

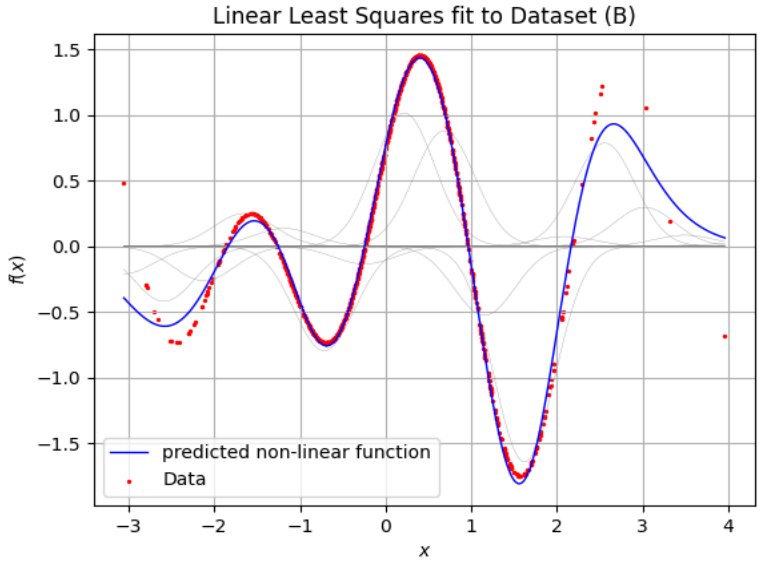


Figure 3: Plot of the non-linear approximated function over the non-linear data contained in Dataset(B)

The resulting fit quality is satisfactory. As shown in Fig.3, the model explains the data almost perfectly, except for some small deviations on the left boundary of the observed range. The grey lines in the figure mark the contributions of the individual basis functions and add up to the blue fit curve.

Regarding the use of Radial Basis Function for linear data, we decided not to use non-linear basis functions like radial basis functions for Dataset(A). Since the data in Dataset(A) shows clear linear characteristics, using radial basis functions would require a very fine grid to accurately model the linear behavior. This is because radial basis functions have localized influence. Furthermore, in the sparser regions at the sides of the data, choosing strongly peaked radial basis functions would lead to over-fitting, resulting in excessive oscillations.

Report on task 2, Approximating linear vector fields

Task description: Implement the approximation for a provided set of 2-dimensional linear vector field data. Given: Two datasets **linear_vectorfield_data_x0.txt** and **linear_vectorfield_data_x1.txt**.EAach contains 1000 data points x_0 and x_1 in 2d.

To accomplish this task, we utilize two Python files: **utils.py** and **task2_approx_linear_vector_fields.ipynb**. The first file contains the necessary functions for solving the linear system, plotting the dataset, and visualizing phase portraits and trajectories. On the other hand, the second file is dedicated to simulation purposes.

Part 1: Estimate the linear vector field that was used to generate the points x_1 from x_0

We need to use the finite-difference formula 5 to estimate the linear vector filed $v^{(k)}$ at all points ${x_0}^{(k)}$ with a timestep $\Delta t = 0.1$. After that, we obtain the linear vector field $\hat{v} \in \mathbb{R}^{1000\times 2}$.

$$\hat{v}^{(k)} = \frac{x_1^{(k)} - x_0^{(k)}}{\Delta t} \tag{5}$$

Since the vector field is linear, we can expect formula6 satisfies for all k.

$$\nu\left(x_0^{(k)}\right) = v^{(k)} = Ax_0^{(k)} \tag{6}$$

To approximate the matrix $A \in \mathbb{R}^{2\times 2}$, we using **np.linalg.lstsq** to minimize $\left\|\hat{v} - x_0 A^T\right\|^2$.

Fig.4 shows the result. The left part is the visualization for the given two datasets, right part is the phase portrait for this Ax linear system.

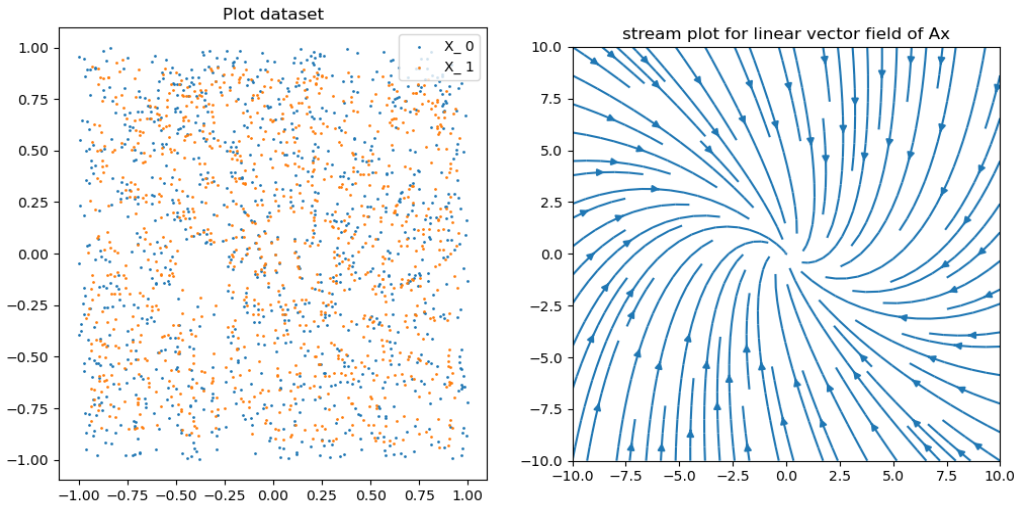


Figure 4: Left: Plot for the given datasets. Right: phase portrait for the linear vector field.

Part 2: Solve the linear system and compute the mean squared error (MSE)

In this part, we solved the linear system $\dot{x} = \hat{A}x$, where $\dot{A} \approx A$. We integrated the system for all initial points $x_0^{(k)}$ up to a time $T_{end} = \Delta t = 0.1$ to obtain estimates for the points $x_1^{(k)}$. To evaluate the accuracy of our approximation, we calculated the mean squared error (MSE) using the formula 7. The resulting **MSE** value was 0.0030599275959897333, indicating a small average integration error. This confirms that our linear approach for approximating the vector field was a good choice, as expected, and that the approximation was good.

$$\frac{1}{N}\sum_{k=1}^{N}\left\|\hat{x}_1^{(k)} - x_1^{(k)}\right\|^2 \tag{7}$$

Part 3: Visualize the trajectory as well as the phase portrait

In this part, we again solve the linear system with $\dot{x} = \hat{A}x$ for $T_{end} = 100$ with an initial point (10,10) which is far outside the initial data. The visualization for the trajectory, as well as the phase portrait in domain $[-10, 10]^2$, is shown in Fig.5

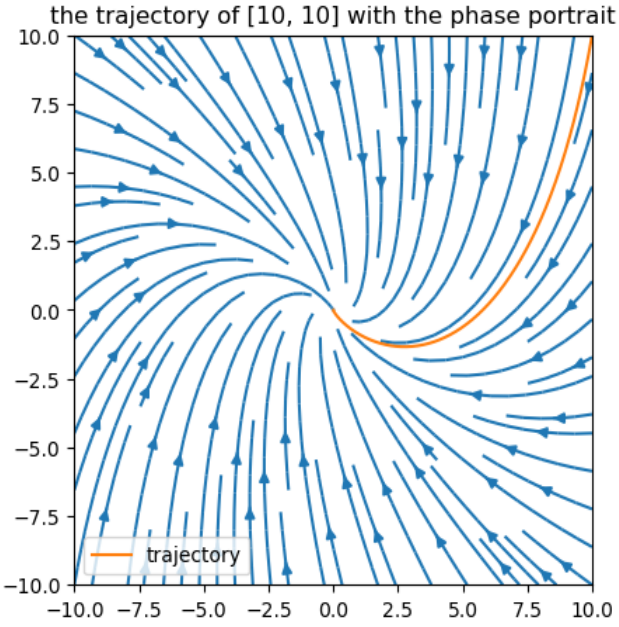


Figure 5: phase portrait with a trajectory of a point at (10,10).

In the phase portrait, it is evident that there exists an attractive steady state located at the origin (0, 0). The trajectory followed by the initial point at (10, 10) is represented by the orange line, illustrating its path toward reaching the steady state.

Report on task 3, Approximating nonlinear vector fields

In this task, we approximate a non-linear vector field dataset. We use linear and non-linear approximation methods separately and compare the 2 methods. The datasets we use are **nonlinear_vectorfield_data_x0.txt** and **nonlinear_vectorfield_data_x1.txt**, where the first one contains initial 2D points and the second one describes positions of x_0 after they move for 0.01s. These points are in the range $[-4.5, 4.5]^2$. Relation between x_1, x_0 is an unknown evolution operator $\psi : T \times R^2 \to R^2$, such that:

$$x_1^{(k)} = \psi(\Delta t, x_0^{(k)}), k = 1, ..., N \tag{8}$$

,where $\Delta t = 0.01$. It is more feasible and reasonable to estimate the derivative of the evolution function, which is:

$$\frac{d}{ds}\psi(s, x(t))|_{s=0} \tag{9}$$

Our implementation can be found in **task3.ipynb**.

Part 1: Linear Approximation

Firstly, we try to use a linear function to approximate the derivative of the evolution function. That is:

$$\frac{d}{ds}\psi(s, x(t))|_{s=0} = Ax(t) \tag{10}$$

We use **numpy.linalg.lstsq** to solve the least square problem $argmin_A\|x_1 - Ax_0\|^2$. The phase portrait of our linear approximation can be visualized in Fig..

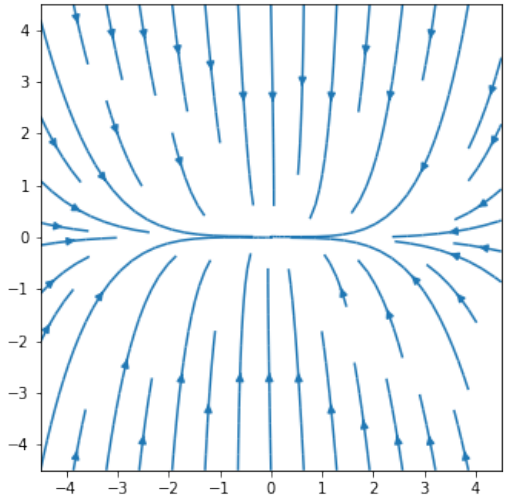
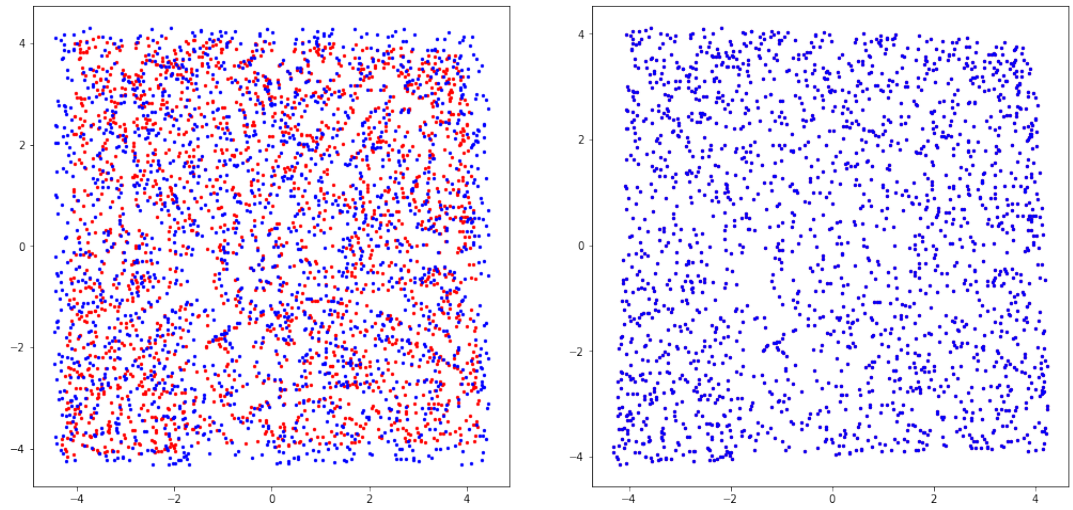


Figure 6: phase portrait of linear approximation for the non-linear vector field dataset.

To evaluate the linear approximation method quantitatively, we calculate the mean square error (MSE) of the predicted points and ground truth points i.e. $\|x_1 - A^*x_0\|^2$, where A^* is the result of $argmin_A\|x_1 - Ax_0\|^2$. The MSE result is **0.03728697**. To qualitatively show the result, we scatter the predicted points and ground truth points in Fig.7(a). As we can see, though the MSE result looks small, the gap between the predicted points and the ground truth points is not neglectable.



(a) (b)

Figure 7: We compare prediction results of linear approximation and non-linear approximation. Red points are ground truth points(x_1) and blue points are predicted points. Apparently, non-linear approximation is better than the linear one.

Part 2: Non-Linear Approximation

Since we can't ideally approximate the non-linear vector field dataset with a simple linear function, we then try to use a non-linear function to approximate it. The non-linearity we use is called **Radial Basis Functions**. Similarly, We use this function to approximate the derivative of the evolution operator. Radial Basis Function does not directly apply data X to a linear transformation and instead replaces it with distances between data points and manually defined basis. The distance function we use is $\phi_l(x) = exp(-\|x_l - x\|^2/\epsilon)$ for x to l-th basis, where x_l is the center point of the basis, and ϵ is the bandwidth. For supervising the function, we use L2 norm minimization:

$$argmin_C \|F - \phi(X)C^T\|^2 \tag{11}$$

, where $C \in R^{d \times L}$, L is the number of basis, and

$$\phi(X) = [\phi_1(X), \phi_2(X), ..., \phi_L(X)]^T \in R^{N \times L} \tag{12}$$

We have conducted a comprehensive experiment to find a good ϵ and the number of the basis L. For simplicity, we let the basis be uniformly distributed in $[-4.5, 4.5]^2$ space. For clarification, the version of ϵ we use doesn't have square power. Firstly, for L, ranging from 100 to 1000, we find **1000** is the best; based on $L = 1000$, we search ϵ from 0.1 to 10, and we find $\epsilon = 2.60$ is the best. The corresponding phase portrait is shown in Fig.. Under this circumstance, MSE achieves a very low amount: **2.84657e-16**, which is significantly lower than the error of the linear approximation. To qualitatively show the result, we scatter the predicted points and ground truth points in Fig.7(b). As we can see in the image, the predicted points are almost the same as the ground truth. With the quantitative result and the qualitative result, we can conclude that the vector field is non-linear.

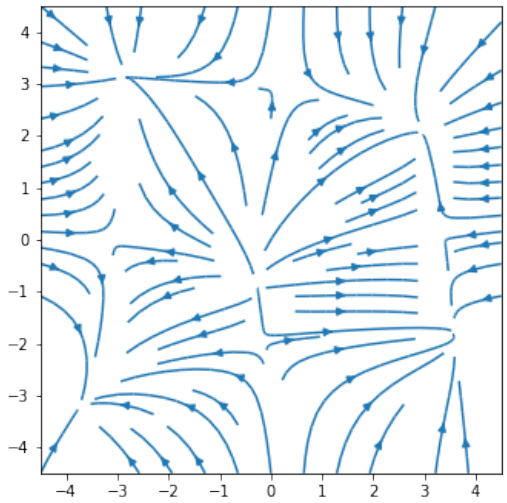


Figure 8: phase portrait of non-linear approximation for the non-linear vector field dataset.

Part 3: Steady States

In this part, we try to analyze steady points of approximation systems. We use approximated vector fields to solve the system for a long time ($\Delta t = 10$). We use **scipy.integrate.solve_ivp** to help us to solve it. The result is shown in Fig.9(a), 9(b). It is easy to get a conclusion that the non-linear approximation system is not topologically equivalent to the linear approximation system, since the number of steady points for the non-linear system is **four**, and it is one for the linear system.

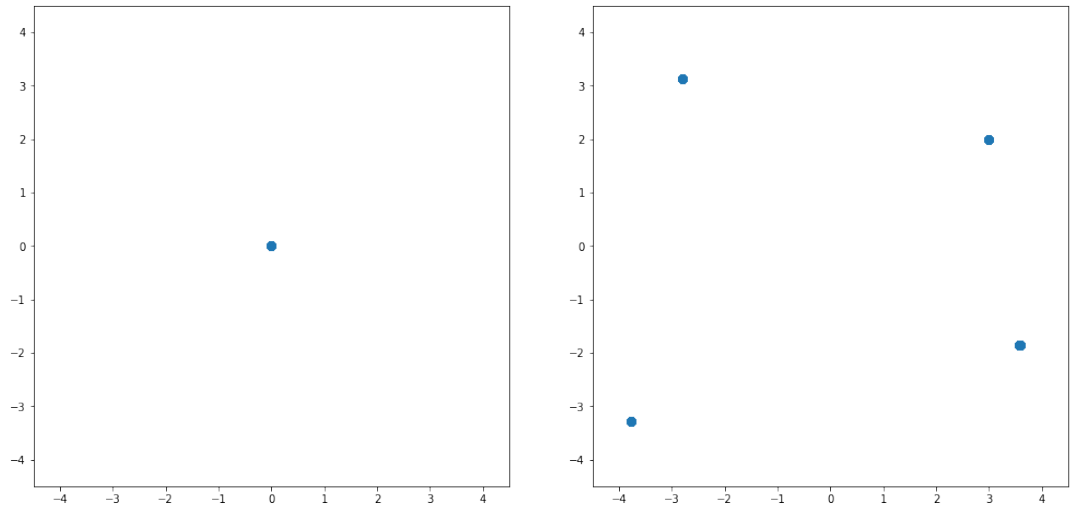


Figure 9: Fig.9(a): Steady points in the linear approximation system; Fig.9(b): Steady points in the non-linear approximation system.

Report on task 4: Time-delay embedding

In this part, we explored Takens theorem and its practical application known as time-delay embedding. Takens theorem is a fundamental concept in nonlinear dynamics that allows us to extract additional information from a time series through time-delay embedding. When we have a measured time series x(1), x(2), ..., x(n) originating from a d-dimensional attractor, we can reconstruct the phase space using delay coordinates. A delay vector takes the form

$$v(t) = (x(t), x(t + \Delta t), x(t + 2 \times \Delta t), ..., x(t + 2d \times \Delta t)) \in \mathbb{R}^{2d+1}. \tag{13}$$

Takens theorem states that for a time series originating from a chaotic attractor in a d-dimensional system, we can construct an embedded phase space of dimension k that is topologically equivalent to the original phase space, where k is greater than or equal to 2d + 1.

Part 1: Embedding periodic signal into a state space

Task description: We need to embedding a periodic signal into a state space where each point carries enough information to advance time.
Given: Dataset **takens_1.txt** with the matrix $X \in \mathbb{R}^{1000\times 2}$.
After obtaining the dataset, we apply time-delay embedding in the first coordinate. We plot the given dataset as shown in Fig.10

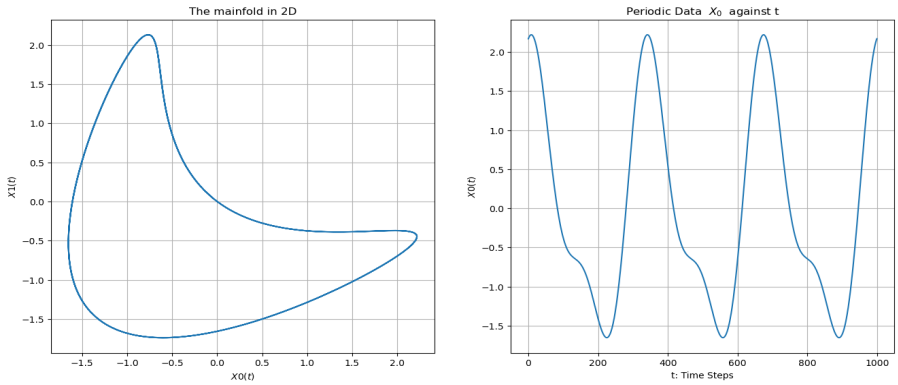


Figure 10: Left: Plot the dataset in 2D space. Right: First coordinate $x_0(t)$ against time.

According to the Takens theorem, the periodic manifold in our system is one-dimensional. To accurately embed the periodic manifold, we follow the theorem's requirement of using $(2 \times 1) + 1 = 3$ coordinates. Then, we select a time delay of $\Delta t = 31$ to visualize the embedding. The plot for the first coordinate against its delayed version illustrates in Fig.11.

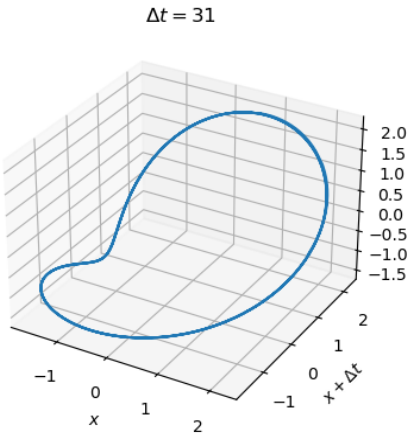


Figure 11: the first coordinate $x_0(t)$ against its delayed version.

In addition, we also plot the figures with different choices of $\Delta t \in [1, 120]$ as the first Fig.12 shown. We could find that when Δt is too small, we observed that the differences between$x(t)$, $x(t + \Delta t)$, and$x(t + 2\Delta t)$ become extremely small, almost forming a straight line. In such cases, the time-delay embedding does not provide significant additional information, as the data points are too closely clustered. However, with an appropriately chosen Δt, we found that the resulting embedding becomes topologically almost equivalent to the original data plot. ($\Delta t = 31$ and $\Delta t = 41$)

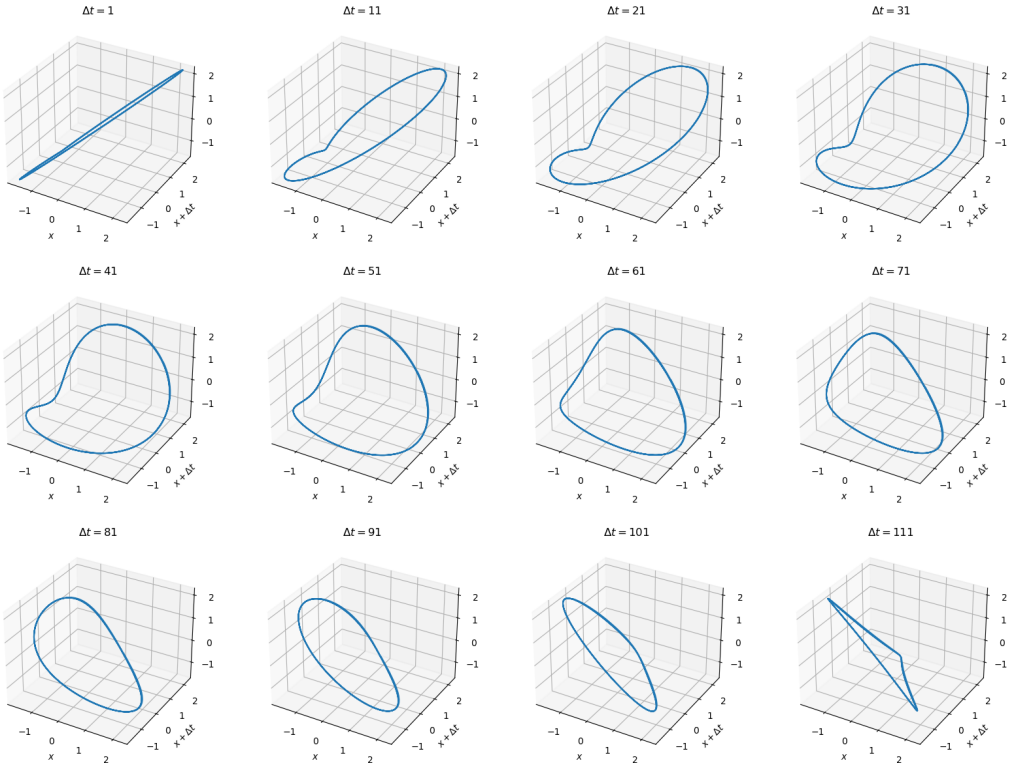


Figure 12: the first coordinate $x_0(t)$ against its delayed version with different Δt.

Part 2: Approximating chaotic dynamics from a single time series

Task description: We need to test Takens theorem on the Lorenz system.
Given: A Lorenz system with $\sigma = 10, \rho = 28, \beta = 8/3$ and initial point (10,10,10) We selected $\Delta t = 100$ and $T_{end} = 100$ to plot the x,y,z against their delayed version separately.

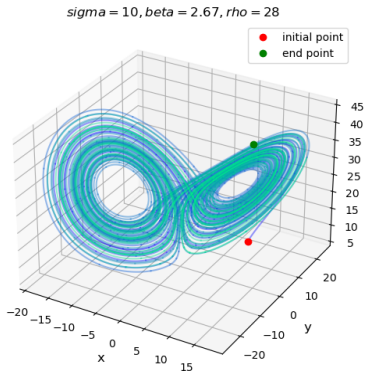
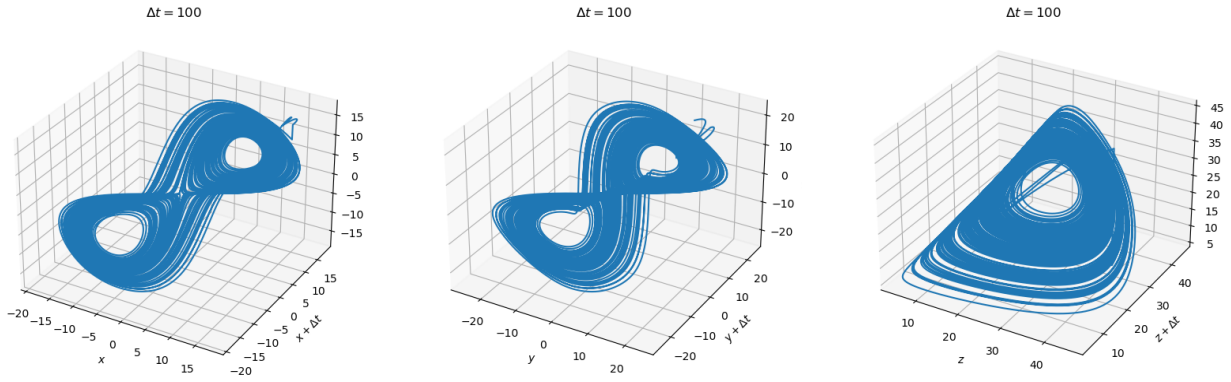


Figure 13: Plot for trajectory in Lorenz attractor with initial point(10,10,10).

First, we have a look at the original trajectory in Lorenz attractor. Fig.13. Then corresponding Time-delay embedding in different coordinates show in Fig.14



(a) (b) (c)

Figure 14: a): Time-delay embedding in X; b):Time-delay embedding in Y; c): Time-delay embedding in Z.

From the analysis of the figures presented in Figure 14, several observations can be made regarding the time-delay embedding of the variables x, y, and z.
Dynamics of x and y: The evolution of the x and y variables demonstrates a remarkable similarity, as their trajectories resemble two butterfly wings. This resemblance indicates that the time-delay embedding successfully captures the intricate dynamics of these variables, representing the original trajectory effectively.

The behavior of z: In contrast to x and y, the behavior of the z variable differs significantly. It remains non-negative throughout the evolution. This behavior suggests that the z variable does not convey information about the signs of x and y. Consequently, when using z alone for time-delay embedding, the resulting representation fails to capture the essential sign-dependent dynamics of the system.

In addition, since the Lorenz system exhibits a symmetry property where if (x(t),y(t),z(t)) represents a valid solution, then (-x(t), -y(t), z(t)) also represents a valid solution. However, this symmetry property is not preserved in the time-delay embedding of the z coordinate. As a consequence, the two distinct wings of the original trajectory collapse into a single representation in the embedded space, resulting in a loss of important information.

Therefore, the time-delay embedding of the x and y variables yields reasonable results, accurately representing the original trajectory. However, the z coordinate fails to capture the sign-dependent dynamics and symmetry properties of the Lorenz system, leading to a distorted embedding where the two wings merge.

Bonus: Approximate the vector field $\hat{v}$ on it using radial functions

Task description: We need to approximate the vector field $/hatv$ on it using radial basis functions ans solve the differential equation 14 with **solve_ivp**.

$$\frac{d}{ds}\psi\left(s, x_1(t), x_2(t), x_3(t)\right)\Big|_{s=0} = \hat{\nu}\left(x_1(t), x_2(t), x_3(t)\right) \tag{14}$$

We firstly create two trajectories $x_0(t) = (x((4), x(t+\Delta t), x(t+2\Delta t))$, and $x_1(t) = (x((t+1), x(t+1+\Delta t), x(t+1+2\Delta t))$, and then create a vector v by equation 15

$$v = \frac{x\_1 - x\_0}{time\_step}. \tag{15}$$

In this case, we choose L = 20 as the number of basis functions for approximation. A larger L generally leads to

better approximation quality, but it comes at the cost of increased computation time. Since the computational resources are limited, we select L = 20 to strike a balance between accuracy and computation time.(But still took me almost one hour to run the code)
To define the bandwidth parameter ϵ, we set it to 6, which has been found to perform well with the chosen value of L. The ϵ parameter controls the spread or width of the radial basis functions and influences the approximation quality. The reconstructed vector field result shows in Figure 15.

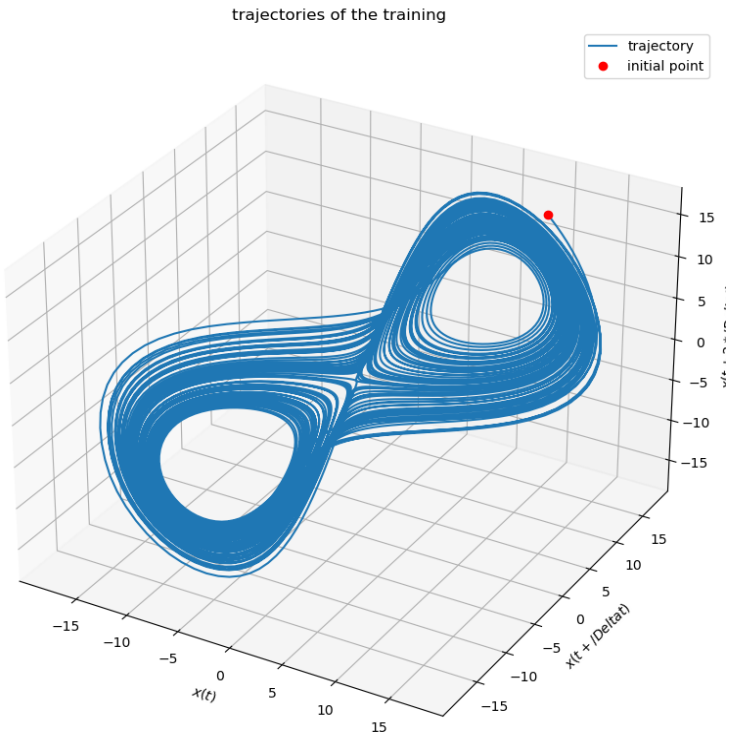


Figure 15: Plot for the trajectory of the approximating the vector field $/hatv$.

Upon comparing our approximate trajectory with the second trajectory in Figure 14 *a*), we observe a remarkable similarity in terms of their outlines.

Report on task 5, Learning crowd dynamics

This part presents a dataset that provides information about the number of people at various locations within TUM Garching, including the MI Building and the 2 mensas. The dataset spans a period of 7 days and consists of a time index (nature number starting from 1) in the first column, followed by nine columns representing the number of people in each area at specific times. Our goal is to analyze and understand the utilization of these locations. Fig.16 shows the utilization of nine different areas, such as the two mensas and the MI building, over the course of seven days.

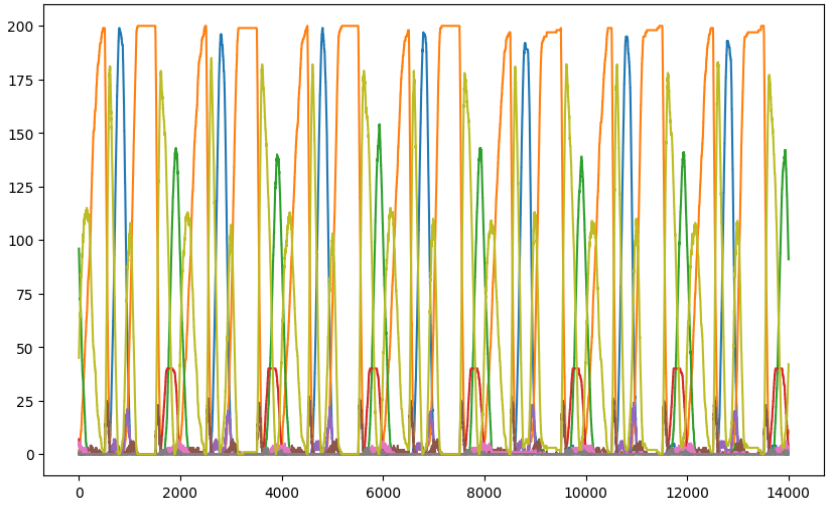


Figure 16: Dataset in the timestep file. The number of people at each place during seven days

Part 1: Create a reasonable state space

The dataset we have is periodic and consists of nine dimensions forming a closed loop. To simplify the analysis, we can reduce it to a one-dimensional representation. According to Takens theorem, we can create a three-dimensional embedding that captures the essential characteristics of the system. In order to select the most appropriate parameter for Principal Component Analysis (PCA), we examine the cumulative explained variance for different numbers of components. From the analysis, we find that only three principal components are needed to effectively represent the system's state with the main energy. To apply PCA, we perform a delay embedding on the first three columns (2, 3, 4) of the dataset, excluding the first 1000 rows. After reducing the nine measurement zones to three with the results shown in Fig.17. We create "windows" of data with 351 rows and 3 coordinates. Then, we use PCA to reduce these windows to three coordinates, which correspond to the principal components. The resulting plot in Fig.18 shows the distribution of the data in a 3-dimensional space, with each dimension representing one principal component.

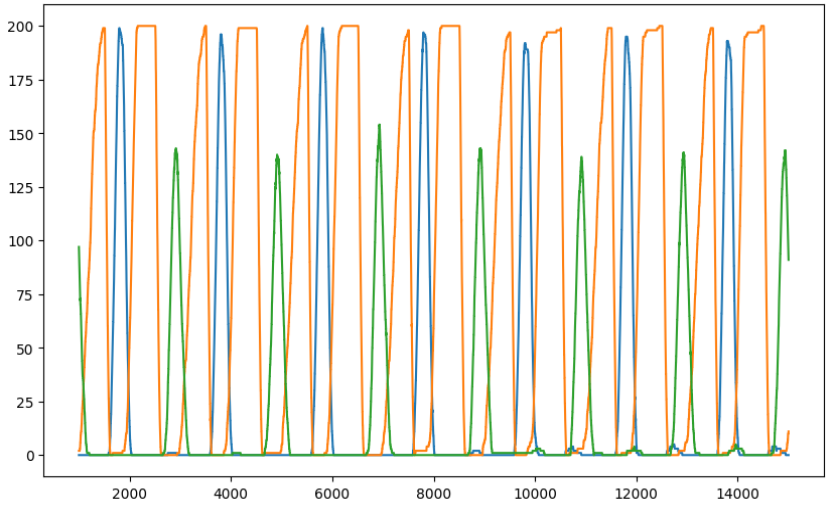


Figure 17: The number of people at each place during seven days(only 3 measurements)

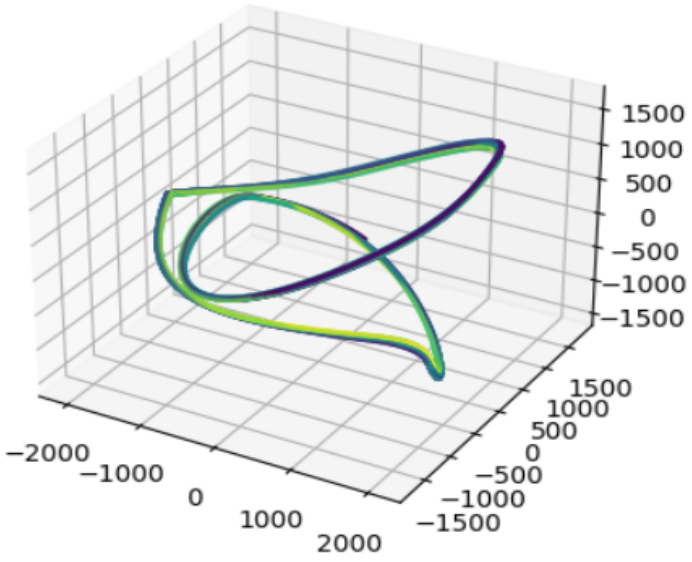


Figure 18: The PCA embedded result with 3 principle components in 3D space

Part 2: Plot the embedded points

In this part, we created 9 plots of the PCA embedding where each point is colored based on the measurement area it belongs to in the original data. To achieve this, we used the scatter function of **matplotlib** and passed the different columns as the color parameter. The resulting plots are shown in Fig.19.

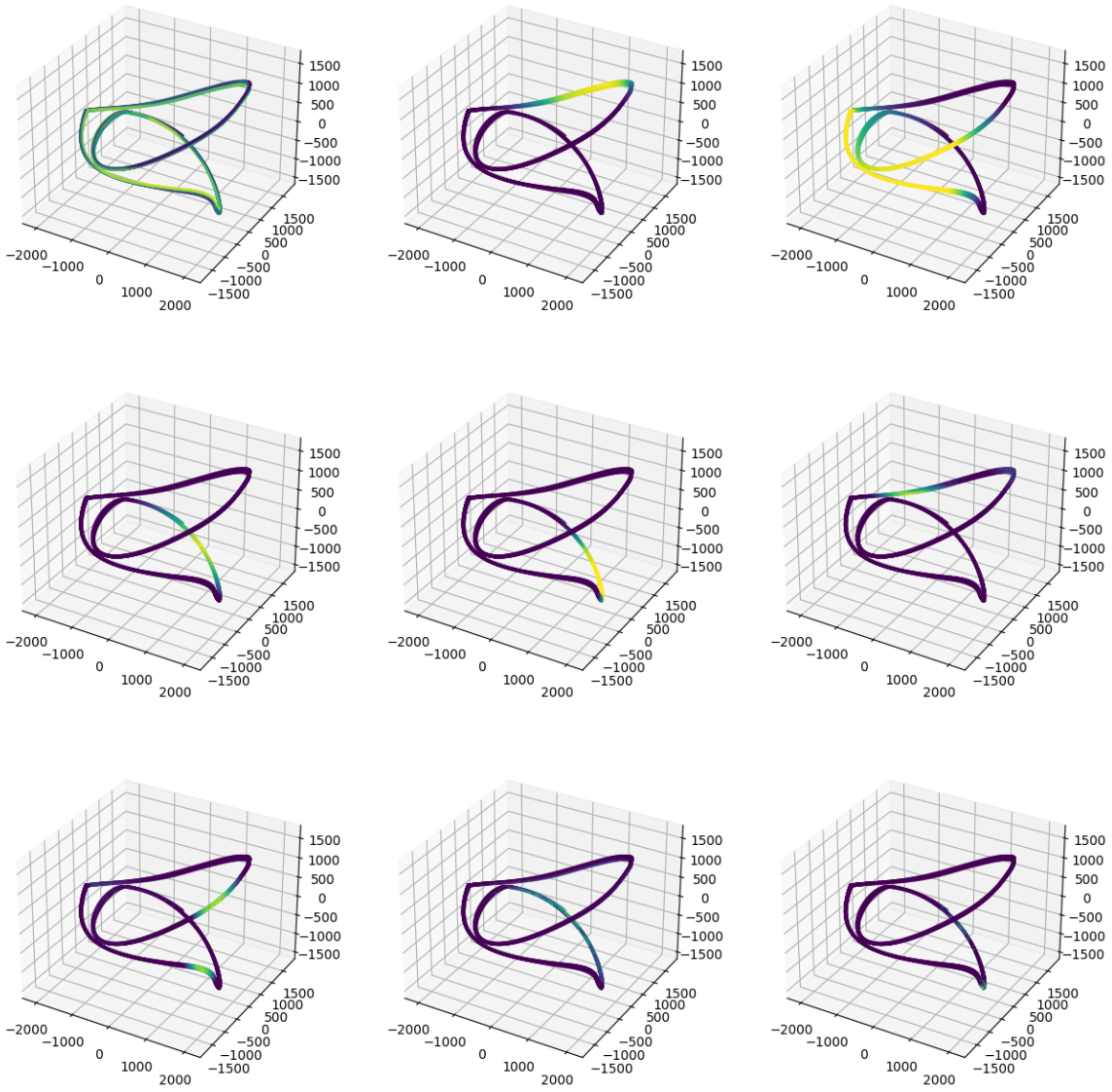


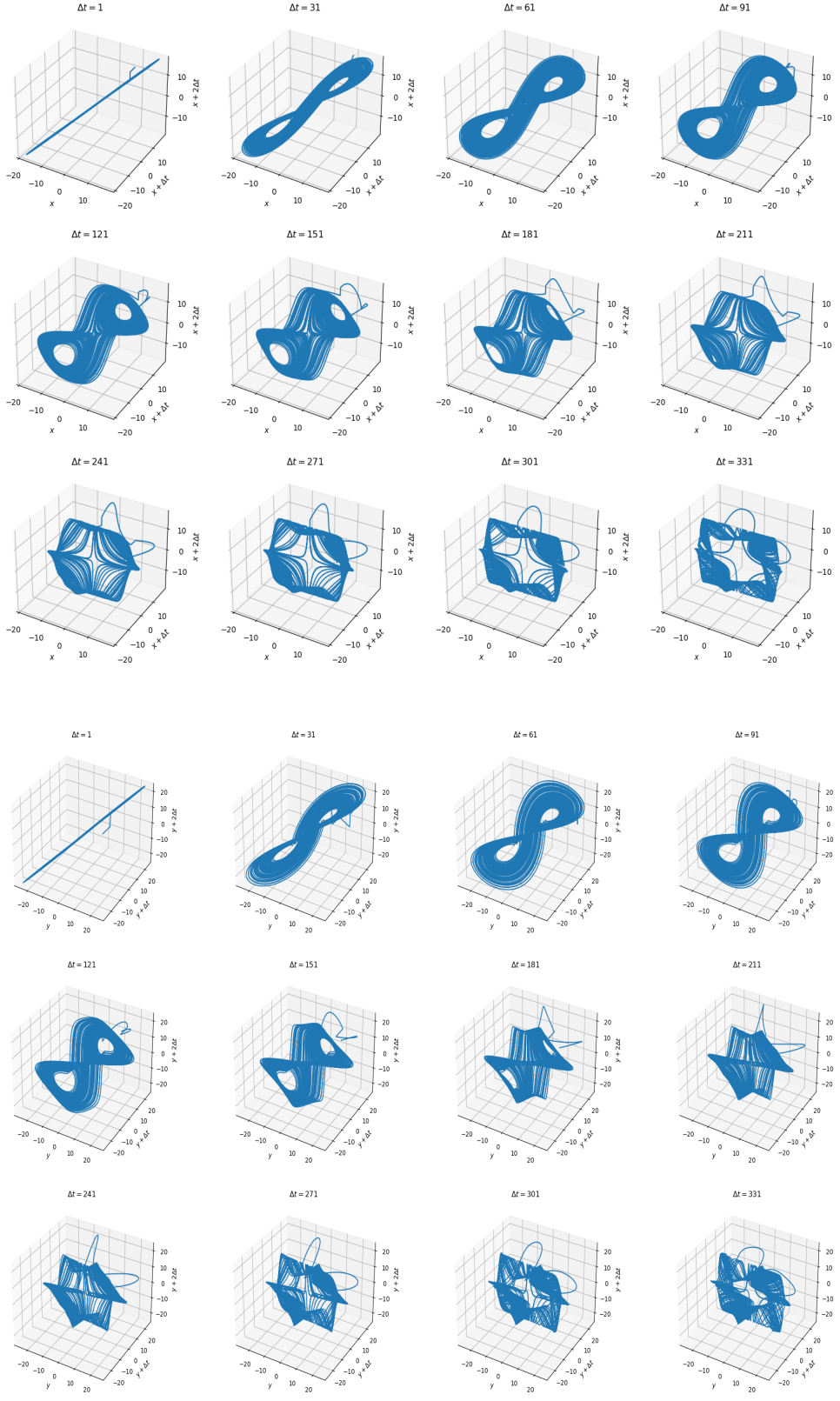
Figure 19: 9 different measurements coloring on PCA embedding

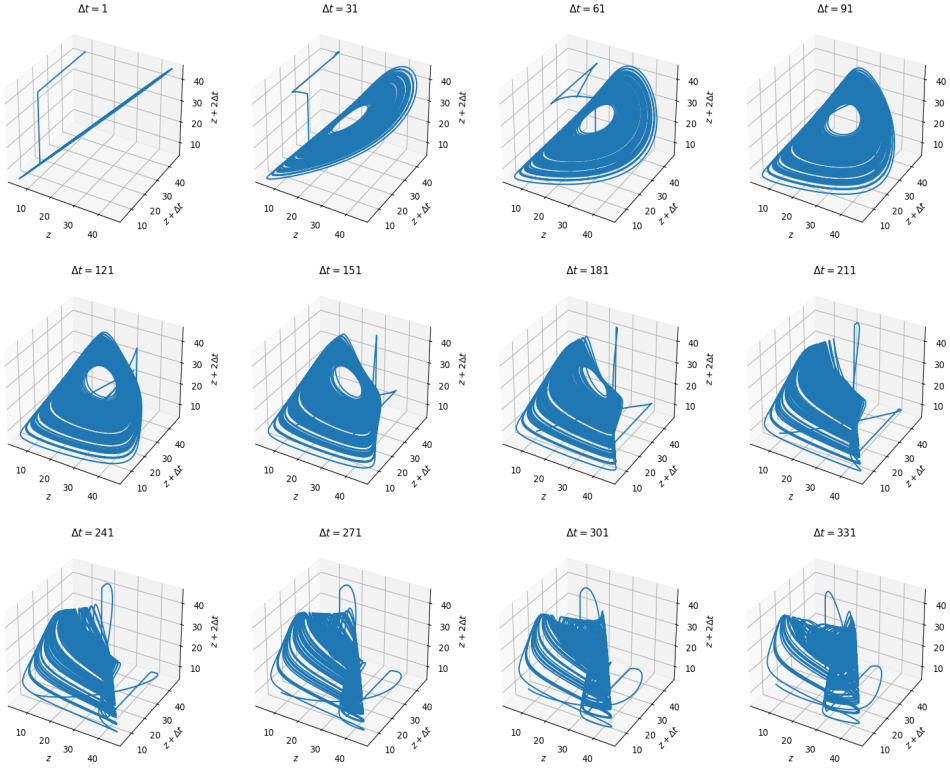
Part 3: Learn the dynamics of the periodic curve

References

Appendix

Task 4 Part 2: Time-delay embedding in X,Y,Z with different choices of Δt




Δt = 1
Δt = 31
Δt = 61
Δt = 91
Δt = 121
Δt = 151
Δt = 181
Δt = 211
Δt = 241
Δt = 271
Δt = 301
Δt = 331
z
z + Δt
z + 2Δt