Report for exercise 5 from group F

Tasks addressed:	5
Authors:	Jianzhe Liu (03751196) Hao Chen (03764817) Yang Cheng (03765398) Pemba Sherpa (03760783)
Last compiled:	2023–06–26
Source code:	https://github.com/Chuck00027/MLCMS-GroupF/tree/main/Exercise5

The work on tasks was divided in the following way:

Jianzhe Liu (03751196)	Task 1	25%
	Task 2	0%
	Task 3	0%
	Task 4	0%
	Task 5	100%
Hao Chen (03764817)	Task 1	25%
	Task 2	100%
	Task 3	0%
	Task 4	0%
	Task 5	0%
Yang Cheng (03765398)	Task 1	25%
	Task 2	0%
	Task 3	100%
	Task 4	0%
	Task 5	0%
Pemba Sherpa (03760783)	Task 1	25%
	Task 2	0%
	Task 3	0%
	Task 4	100%
	Task 5	0%

Report on task TASK 1, Approximating functions

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

Approximate the function in dataset (A) with a linear function The objective of this task was to approximate the function in dataset (A) using a linear function.To approximate the linear function, we utilized the least-squares minimization method . The main idea was to find a matrix A that minimizes the sum of squared errors between the predicted output values and the true output values.

First, we loaded the dataset and extracted the input values and true output values. We then constructed a design matrix by stacking the input values with a column of ones to account for the intercept term.Next, we solved for the matrix A using the closed-form solution provided by the least-squares minimization method. The solution, $\mathbf{A}^T = (\mathbf{X}^T X)^{-1}\mathbf{X}^T F$, allowed us to estimate the coefficients of the linear function.

From the estimated matrix A, we could extracted the slope and intercept, which represent the coefficient of the input variable and the constant term, respectively. These values defined the linear function approximation.The corresponding result is shown in Figure 1 and the approximating function is f(x) = 0.7500002294942191 * x + 2.6434719522205095e-07.

Why is it not a good idea to use radial basis functions for dataset (A)? As the result shown in Figure 1, the distribution of the data is linear.Therefore it is reasonable to approximate the dataset with a linear function.On the other hand, radial basis functions are beneficial for approximating nonlinear functions.If radial basis functions are applied here to approximate dataset (A), it might result in overfitting and unnecessary computational complexity.

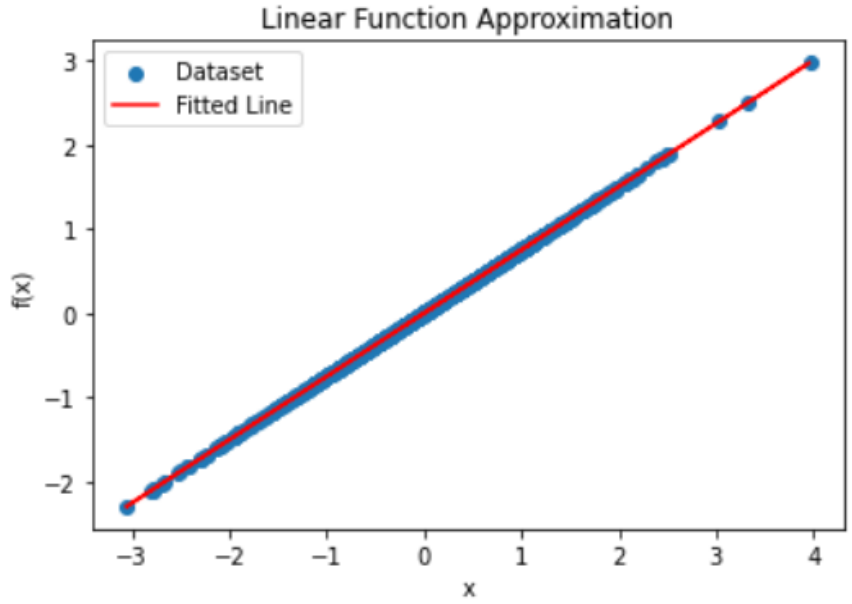


Figure 1: Linear Function Approximation

Part 2

The second part wants to perform a linear approximation on the data, which is contained in `nonlinear-function_data.txt`. The procedure is the same as the one we used on part 1, we simply read the given data file and apply the approximated function on it. The result is shown in Figure 2, and, as expected, approximating such data with a straight line does not provide good results.

```
Linear Function Approximation:
f(x) = 0.02873497516022028 * x + 0.11114247433936228
```

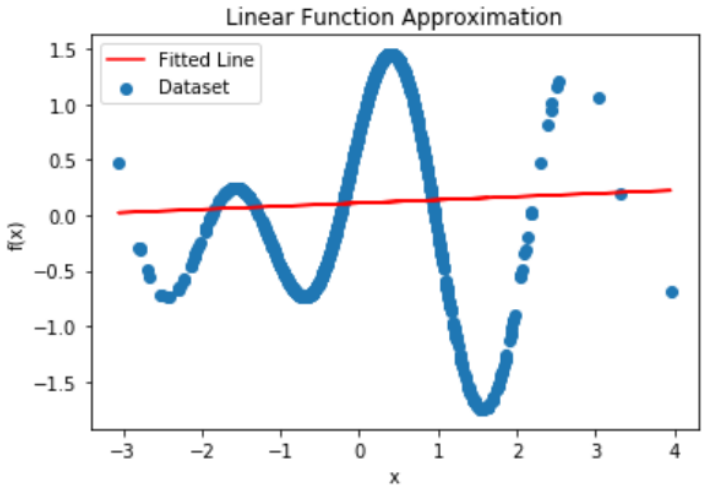


Figure 2: Non-Linear Function Approximation

Part 3

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

From the second part we can see that linear functions cannot fit nonlinear data well, so we should find a nonlinear function to fit this dataset. Here we use a linear combination of radial basis functions to fit the nonlinear data. The main principle is to map the original input space to a higher-dimensional feature space, and then use linear in this high-dimensional feature space. method to fit.

Discussed how and why you chose the values of L and ϵ for the radial basis functions? We set a value list of L (which is defined as n_bases_list in the code, which is (3, 6, 9, 12, 15)) and a value list of ϵ (which is (0.1, 0.4, 0.7, 5)), and then iterate to get approximation of non-linear data, and then calculate the difference between the approximation of non-linear data and the ground truth to get the MSE, the setting values, which lead to the smallest MSE, are what we want. We found that in the value range of L and ϵ, when L=12, ϵ=0.7, the MSE obtained is the smallest, which is 0.001. Additionally, we can also see from figure 3 that the approximation curve fit the dataset B generally well, So we choose these two values for the radial basis functions.

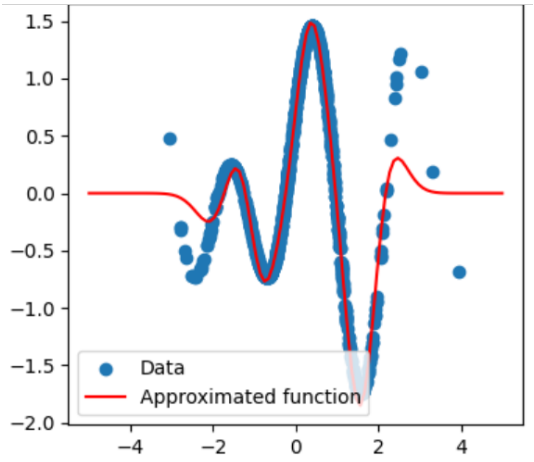


Figure 3: Nonlinear Function Approximation over dataset B under setting of L=12, ϵ=0.7

Why is it not a good idea to use radial basis functions for dataset A? First of all, we have to admit that radial basis functions can be used to fit linear data like dataset A, as shown in figure 4, where L=20, ϵ=100, it can fit dataset A well (but it obviously needs more calculations compared to the linear approximation method). However, when we let L=5, ϵ=0.1, we found that the function curve obtained by this method is very complicated, as shown in figure 5, but it fits the data set A very poorly, I think , this is because RBF may capture and adapt to the noise in the dataset, thereby learning an unnecessarily more complex model, that is, creating an overfitting problem. In general, it is not a good idea to use radial basis functions for linear dataset A, because of the computational complexity and overfitting problems.

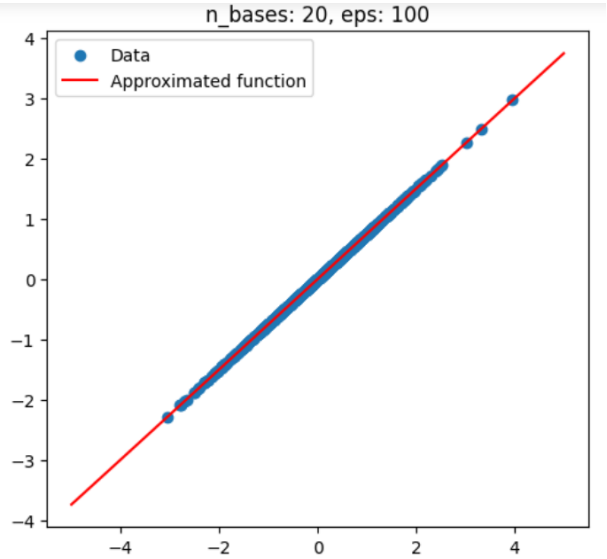


Figure 4: Nonlinear Function Approximation over dataset A under setting of L=20, ϵ=100

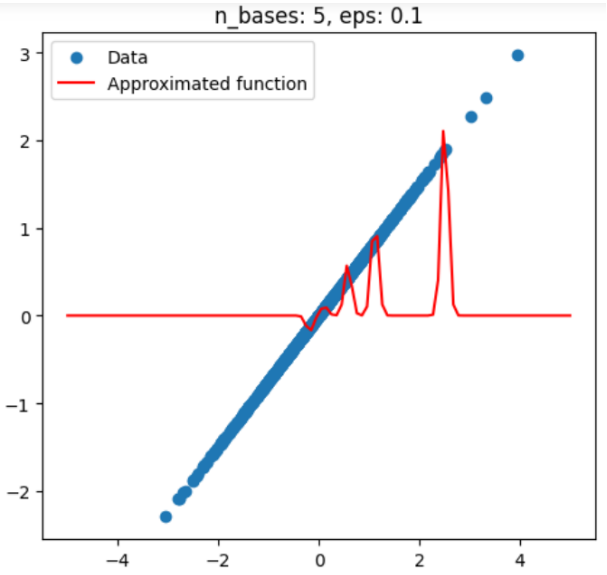


Figure 5: Nonlinear Function Approximation over dataset A under setting of L=5, ϵ=0.1

Report on task TASK 2, Approximating linear vector fields

Part1 Estimate the linear vector field that was used to generate the points x_1 from points x_0

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

The estimation of the linear vector field aims to capture the underlying dynamics of the system that generated the given datasets x0 and x1. By analyzing the relationship between x0 and x1, we can estimate the linear vector field that governs their behavior.

To estimate the linear vector field, we first load the two datasets linear_vectorfield_data_x0 and linear_vectorfield_data_x1 using the load_datasets function from the utils module. These datasets consist of two-dimensional points that represent the system's state at different time steps.(As shown in Figure 6)

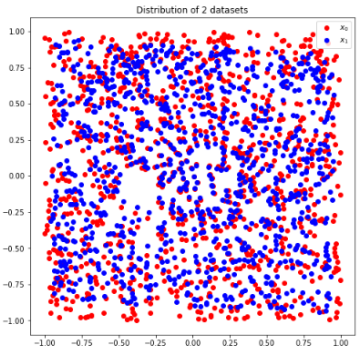


Figure 6: Distribution of 2 datasets

Next, we calculate the vectors v(k) by subtracting the corresponding points in x0 and x1. These vectors represent the rate of change or velocity of the system between consecutive time steps. By dividing the difference vectors by the time interval, we obtain an approximation of the derivatives of the system's state variables.

Finally, we estimate the matrix A by fitting a linear regression model between the points in x_0 and the velocity vectors v. This is achieved using the estimate_vectors function in the utils module. The function utilizes the least squares method to find the matrix A that best describes the relationship between x_0 and v.The corresponding vector field is shown in Figure 7.

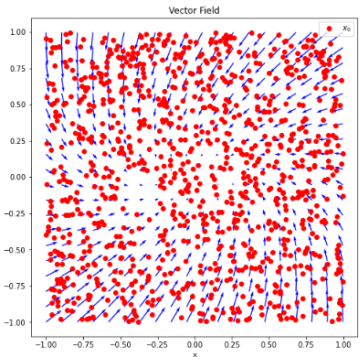


Figure 7: Vector Field

Approximating Matrix A with Supervised Learning

The process of approximating the matrix A with a supervised learning problem involves treating the estimation of A as a regression task. We consider the points in x_0 as the input features and the corresponding velocity vectors v as the target labels.

By formulating the problem as supervised learning, we can leverage various regression algorithms to approximate the matrix A. In this case, we use the least squares method, which provides a straightforward and efficient solution for linear regression problems.

The least squares method involves minimizing the sum of squared errors between the predicted velocities (obtained by multiplying the input features x_0 by the estimated matrix A) and the target velocities v. By

solving this optimization problem, we obtain the approximate matrix

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

The use of supervised learning techniques to approximate matrix A allows us to capture the underlying patterns and relationships within the given datasets. It provides a generalizable model that can be applied to unseen data points, allowing for predictions and further analysis of the system's behavior.

In conclusion, estimating the linear vector field and approximating matrix A provide valuable insights into the dynamics of the system. By analyzing the relationship between x_0 and x_1, we can estimate the linear vector field that governs the system's behavior. Additionally, by formulating the problem as a supervised learning task, we can approximate the matrix A and gain a deeper understanding of the system's dynamics and predictability.

Part2 Solve the linear system and compute the mean squared error

Solving the Linear System

The code utilizes the estimated matrix A_hat obtained from the previous step to solve the linear system. It defines a function solve_linear_system that represents the system dynamics. By passing this function along with the initial point and the time range to the solve_ivp function, the code numerically integrates the system over time. The resulting trajectory represents the evolution of the system from the chosen initial point.

Computing Mean Squared Error (MSE)

To assess the accuracy of the estimated system, the code calculates the mean squared error (MSE) between the estimated trajectory and the actual dataset x1. It uses the function calculate_mse, which iterates over each initial point in x0. For each initial point, it integrates the linear system using the estimated matrix A_hat and compares the final estimated point with the corresponding point in x1. The squared Euclidean distance is calculated, and the sum of these distances is divided by the number of initial points to obtain the MSE.

The obtained Mean Squared Error

After computing the MSE, the code prints the result, which in this case is approximately 0.0030599275959897303. The MSE quantifies the discrepancy between the estimated system trajectory and the actual dataset. A lower MSE indicates a better approximation and suggests that the estimated matrix A_hat captures the underlying dynamics of the system more accurately.

In conclusion, Part 2 of the problem involves solving the linear system using the estimated matrix A_hat and computing the mean squared error (MSE) as a measure of accuracy. By integrating the system and comparing the estimated trajectory with the actual dataset x1, the code provides insights into the quality of the estimation. The obtained MSE value of 0.0030599275959897303, which indicates a reasonable result..

Part3 Choose the initial point (10, 10) amd again solve the linear system with your matrix approximation

Choose the initial point (10, 10) and solve the system.

The code selects the initial point as (10, 10). This point represents the starting condition from which the system evolves over time. By passing the initial point and the estimated matrix A_hat to the solve_ivp function, the code solves the linear system and obtains the trajectory of the system.

Visualize the trajectory as well as the phase portrait

After obtaining the solution trajectory, the code proceeds to visualize it along with the phase portrait. The phase portrait is a graphical representation of the system's dynamics in the state space. It shows the flow of the system's trajectories and provides insights into the system's behavior.

To visualize the trajectory, the code plots the x and y coordinates of the solution trajectory using the plot function. The trajectory appears as a curve or a set of connected points that represents the path traced by the system over time. The initial point (10, 10) is also plotted as a marker to indicate the starting position.

The phase portrait is visualized using the phase_portrait function. This function plots the streamlines of the system, which are curves that follow the direction of the vector field defined by the matrix A_hat. The phase portrait provides a comprehensive view of the system's behavior, including stable and unstable points, limit cycles, and other characteristics.

By combining the trajectory plot and the phase portrait, the code enables a visual understanding of how the system evolves over time and its overall dynamics. This visualization is shown in Figure 8

In conclusion, Part 3 of the problem involves choosing an initial point, solving the linear system, and visualizing the trajectory and phase portrait. This allows for a visual representation of the system's dynamics and facilitates the analysis and interpretation of its behavior.

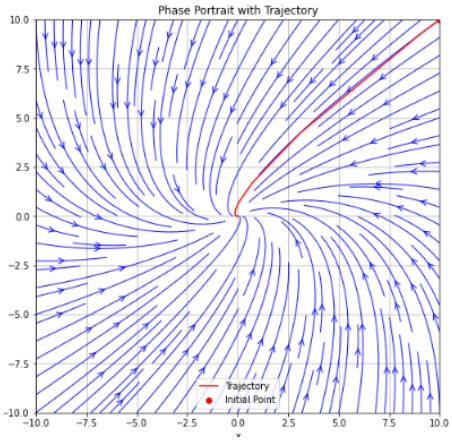


Figure 8: Phase Portrait with Trajectory

Report on task TASK 3, Approximating nonlinear vector fields

The goal of this task is to study and estimate the vector field of a 2D system. The task is divided into three parts. The first part is to use linear operators to estimate the vector field. The second part is to use radial basis functions to approximate the vector field. The third part is to study the steady state of the system. The specific procedures and results are as follows.

Part 1

Estimate the vector field with a linear operator. To estimate a vector field, we need to find a model describing its dynamics from the raw data. In this problem, we are asked to estimate a vector field using linear operators. Mathematically, a linear operator is a matrix that transforms the input space to the output space. In this problem, the input space is the initial points x0, and the output space is the points x1 moved forward by a small amount of time Δt. Our goal is to find a linear operator A such that $A * x0$ is as close to x1 as possible. A common method to solve this problem is the method of least squares. Specifically, we can think of this problem as a linear regression problem, where x0 is the independent variable, x1 is the dependent variable, and A is the parameter we want to solve for. In Python, you can use the lstsq function in the numpy library to solve this problem. After we use this function, we get A (and the residuals, which we will cover later). Figure 9 shows the approximation with such linear function, and figure 10 is the phase portrait of this function.

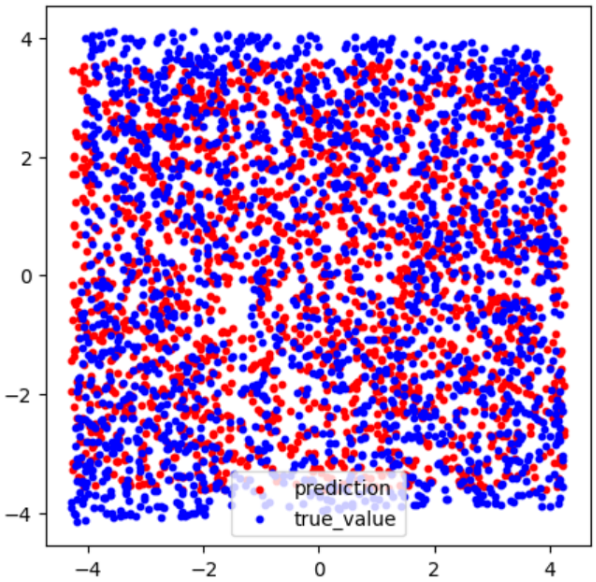


Figure 9: Linear Function Approximation over datasets

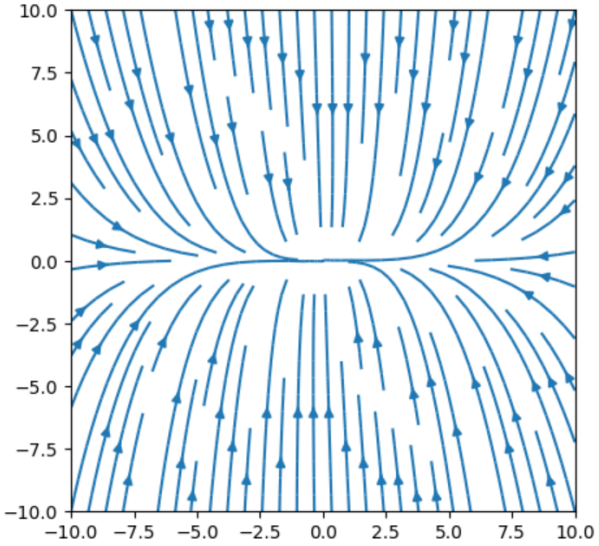


Figure 10: Phase portrait of this linear function

Compute the mean squared error to the solution after Δt as close as possible to x1. To calculate the mean square error of the solution after Δt time, we need two inputs, one is the real target value x1, and the other is the value predicted by the linear operator A. The predicted value can be obtained by multiplying A and the initial point x0. As in the previous question, we use the least squares method to obtain the residual errors. We obtained that when it is at t=0.5, the MSE=0.2,as shown in Jupyter Notebook. In addition, according to the results of the calculate_rajectory function, we found that the smallest MSE is obtained when Δt=0.1 within [0, 0.5].

Part 2

Approximate the vector field using radial basis functions. To improve prediction accuracy, we try to approximate the vector field using radial basis functions (RBF). By tuning the number of basis functions of the RBF and the scaling parameter ϵ, we optimize the RBF model to minimize the MSE when predicting x1. The prediction accuracy after using the RBF method should improve significantly compared to using the linear operator. However, unfortunately, as shown in figure11 the results of these two approximations vary little, I cannot figure out why. But at the same time, I find huge decrease of residual errors, which I will discuss later.

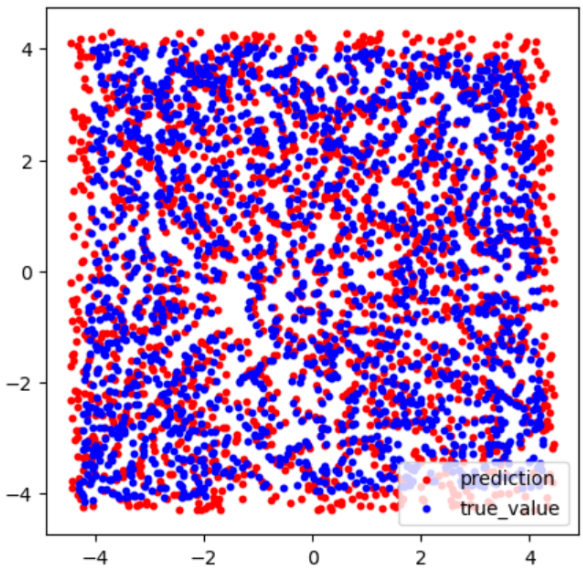


Figure 11: Approximation over dataset using the RBF method

How do the errors differ to the linear approximation? What do you conclude: is the vector field linear or nonlinear? Why? To further determine whether the vector field is linear, we compared the residual errors produced by the linear operator and the RBF method. The results show that the residual errors produced by the RBF method is significantly lower than that produced by the linear operator (near 4000 versus around 10), which proves that the vector field has nonlinear characteristics. Therefore, we can conclude that this vector field is non-linear.

Part 3

Where do you end up, i.e. where are the steady states of the system? As we can see from figure12, there are 4 steady states of the system, which on the corners.

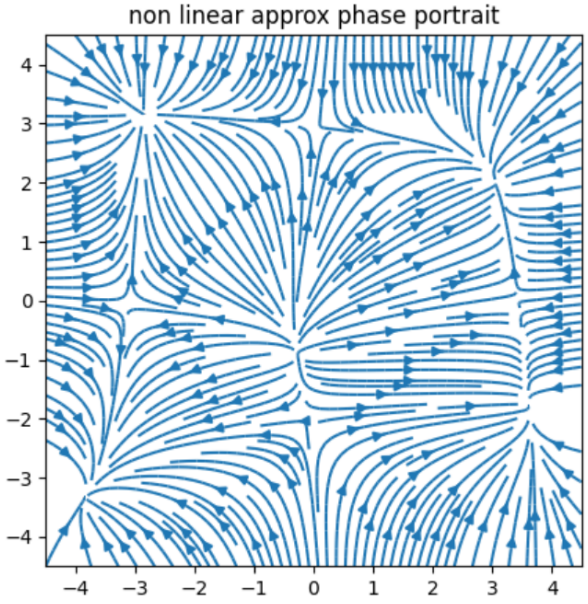


Figure 12: Phase portrait of the system

Are there multiple steady states? Hint: yes, but less than 10. Yes, as I mentioned in last question, there are 4 steady states in total.

Can the system be topologically equivalent to a linear system? Why (not)? No, this is because, for topological equivalence, two systems need to exhibit similar dynamical properties, such as trajectory shape, number and location of steady states, etc. However, in our case, the nonlinear system has multiple steady states while the linear system has only one, which is an important distinction.

Discussed how and why you chose the values of L and ϵ? We use a grid search strategy to find the optimal n_bases (which stands for L in prompt) and eps(which stands for ϵ in prompt). The basic idea of grid search is exhaustive search, that is, in the parameter space, try every possible set of parameter combinations and select the optimal set. In the find_optimal_rbf_config function, for n_bases, its possible value is selected from a series of preset values (from 100 to 1001, 20 values in total), and this series is generated by numpy's linspace function. For eps, its possible values are chosen from a preset list (0.3, 0.5, 0.7, 1.0, 5.0, 15.0, 25.0). The code first randomly selects n_bases center points, and then for each possible combination of n_bases and eps, the corresponding coefficient matrix C is calculated through the nonlinear_least_squares function, and then this matrix is used to calculate the predicted final data and calculate the predicted data Mean Squared Error (MSE) from the real data. The code then checks to see if the newly calculated MSE is smaller than the current optimal MSE. If the new MSE is smaller, then update the optimal MSE and the corresponding configuration parameters. In this way, after all possible parameter combinations have been tried, the parameter combination that minimizes MSE can be found, that is, the optimal n_bases and eps. Finally, the code running results show that when n_bases=337(which may change after each running), eps=15, we get the smallest MSE. Therefore, we chose them.

Report on task TASK 4, Time-delay embedding

Part 1 : The dataset `takens_1` was imported to the Python program by using NumPy's `loadtxt` method. Then, by using the `matplotlib` library, we plotted the first coordinate against the line number from the imported dataset, as shown in the figure13. We used a delay ($\Delta n = 5$) of rows and again plotted the coordinate against its delayed version, as shown in the figure14. As discussed in lecture 5, Takens' theorem states that $> 2m$ coordinates, where m is the manifold dimension, are sufficient for the manifold to be embedded correctly. So, for our case where manifold dimension = 1, as we need at least $2m + 1 = 3$ coordinates so that the periodic manifold is embedded correctly. But, as shown in the figure, two coordinates are sufficient to embed this periodic manifold correctly.

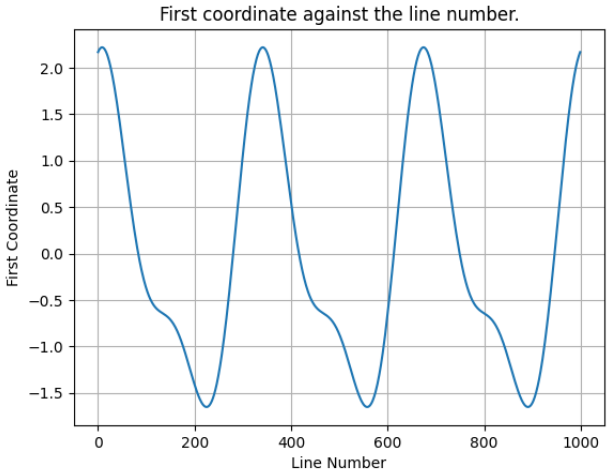


Figure 13: Plot of imported takens data.

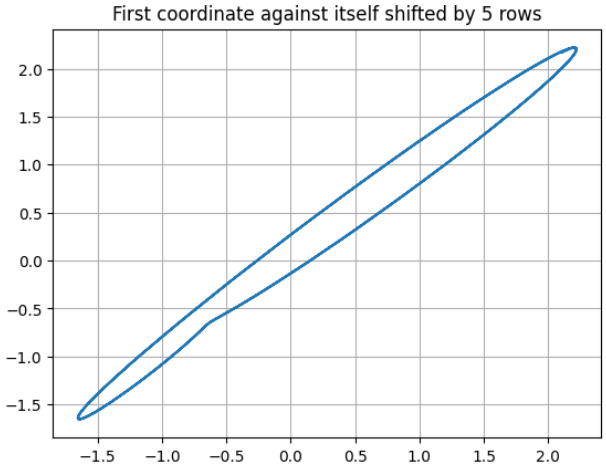


Figure 14: Plot of first coordinate against itself with shift

Part 2 : We plotted the Lorenz attractor as in exercise three with the parameters $\sigma = 10$, $\rho = 28$, $\beta = \frac{8}{3}$, and the starting point of $(10, 10, 10)$15. This is a chaotic dynamics system, and in this exercise, we test the Takens theorem in such a scenario. We first start by plotting $x(t)$ against $x(t + \Delta t)$ and $x(t + 2\Delta t)$ in a three-dimensional plot, with a suitable choice of $\Delta t = 50$. This is shown in the figure16. We can observe that this reconstructed three-dimensional plot is a reasonable approximation of the original Lorenz attractor's shape and dynamics. By choosing an appropriate embedding dimension and time delay ($d = 3$ and $\Delta t = 50$, in this case), Takens' theorem allows for the close reconstruction of the Lorenz attractor. The next step is to try the time-delay embedding for the z-coordinate. We again start by plotting $z(t)$ against $z(t + \Delta t)$ and $z(t + 2\Delta t)$ in a three-dimensional plot, with a suitable choice of $\Delta t = 50$. This is shown in the figure17. However, in this case, the reconstructed structure is not similar to the original Lorenz attractor's shape and dynamics. One of the probable reasons for this failure is that the z-coordinate for the Lorenz attractor is dependent on both the x and y coordinates, and their interactions play a vital role in the Lorenz attractor's shape and dynamics. So, when we reconstruct using only the z-coordinate solely based on the x-coordinate, the full dependencies are not properly represented and thus can lead to potential inaccuracies in the reconstructed attractor.

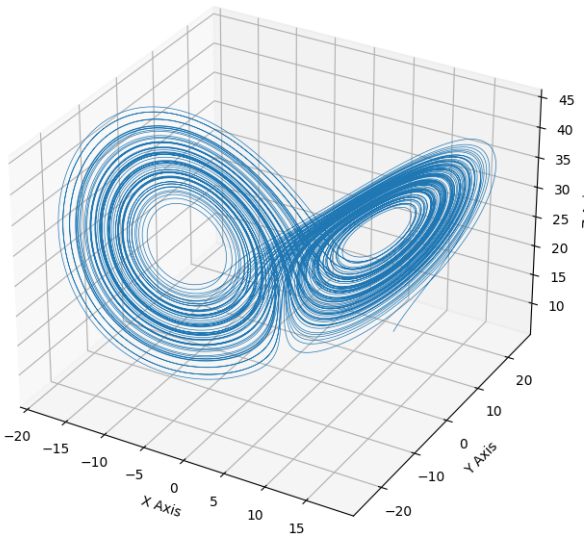


Figure 15: 3d Plot of Lorenz attractor.

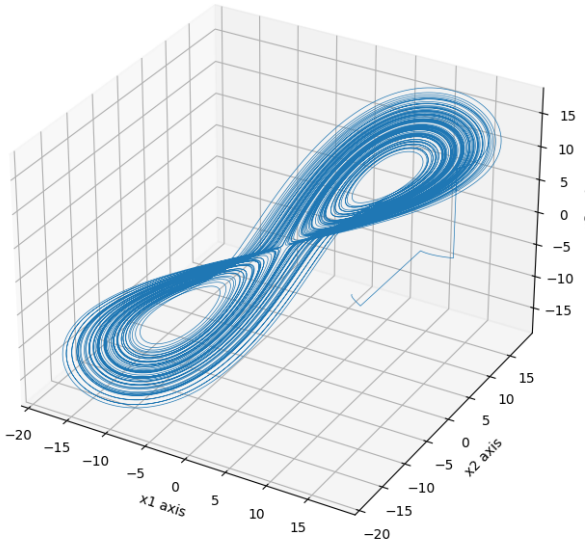


Figure 16: 3d Plot of reconstructed Lorenz attractor using x-axis.

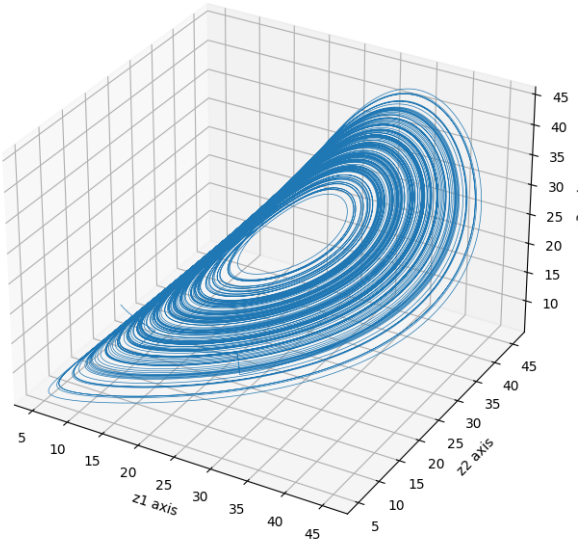


Figure 17: 3d Plot of reconstructed Lorenz attractor using z-axis.

Report on task TASK 5, Approximating nonlinear vector fields

The last task describe a situation where the data of the people flow in nine different areas of TUM Garching Campus, containing the MI Building and the 2 canteens, are given. More clearly, it describes the system over the course of 7 days and is divided into two parts: first column containing the *time index* (which is simply a natural number going up one by one and starting from 1) and nine other columns containing the number of people per area at that certain time. The task is to learn a dynamical system from it. Before any operation, we need to first read the data file and visualize them. In **Figure 18** are shown nine different points at each time step, defining the amount of people per area.

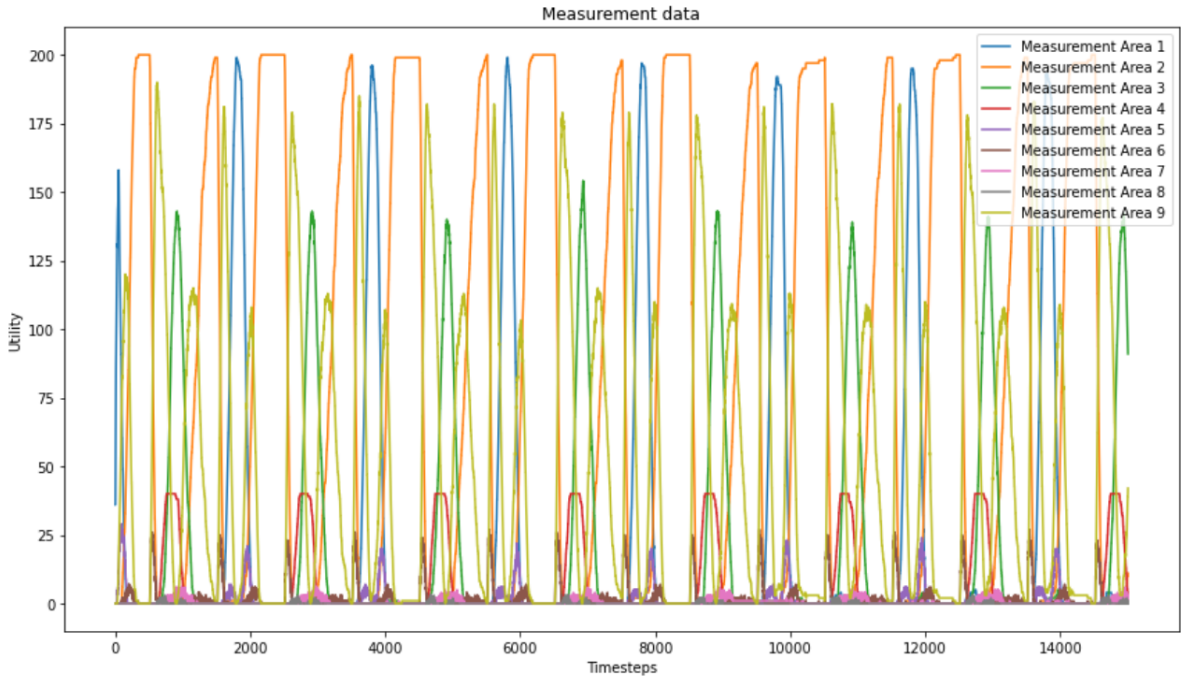


Figure 18: MI building data

Part 1 The first part of the task gives us some important information about this dataset, such as it is periodic (he total time span is seven days and it can also be easily seen from Figure 18 where the same pattern repeats seven times) and with no parametric dependence, therefore being a nine dimensional closed loop which can be reduced to be one-dimensional. According to the Takens theorem, since we already know that d=1, we need at least **2d+1** dimensions to get a reasonable state space. It is also said that we can ignore the first 1000 rows of the dataset because they are data in burn-in period. In Figure 19 you can see the simplified curves with only 3 measurement area.

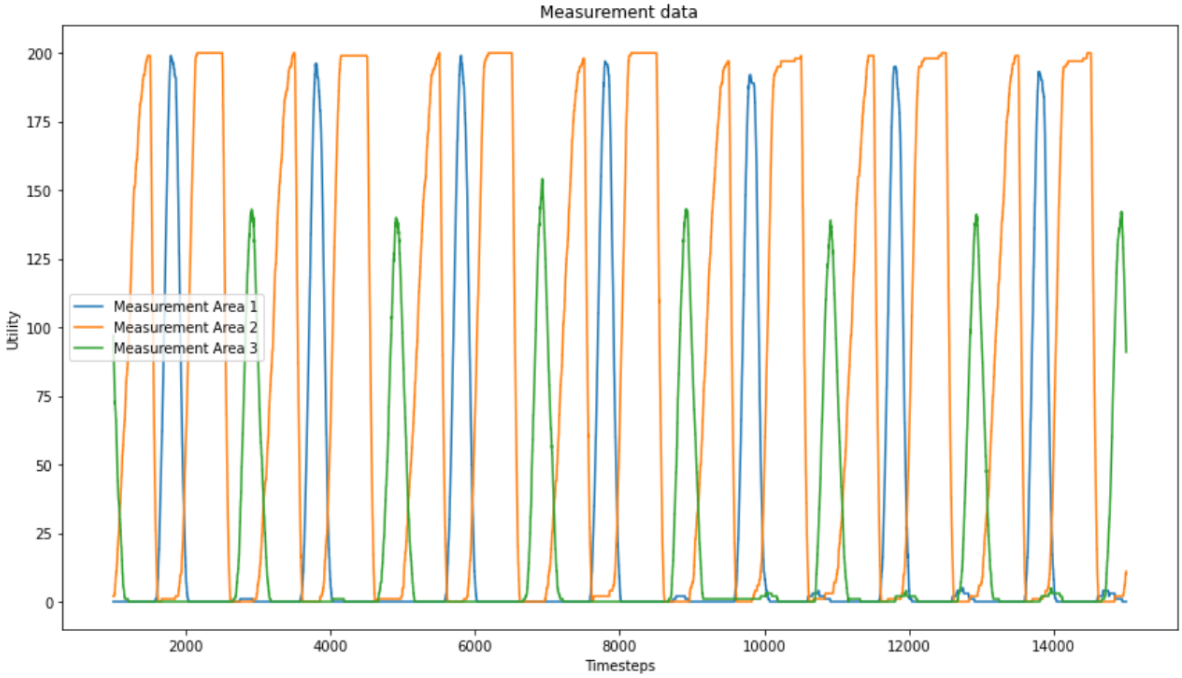


Figure 19: MI building data (only first 3 measurements)

After reducing the nine measurements zones to three, we can do following steps to reach our goal:

- create window that containing the first three measurements.
- create time delay embedding by taking flattened windows of 351 consecutive points for each column.
- create M of those points by moving the window from start, moving each time by one row till the very end. Then create a new array in size of this.

(Here is important to mention that after performing the time-delay, the total amount of the data should be 15000-1000-350=13650.)

After those steps, we can finally see the first 3 principal components in plot(shown in Figure 20)

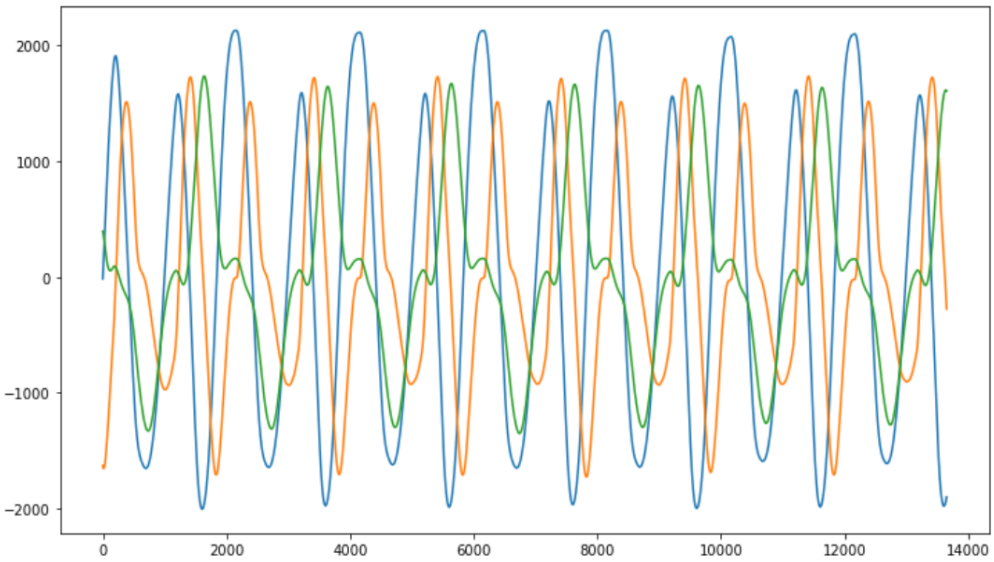
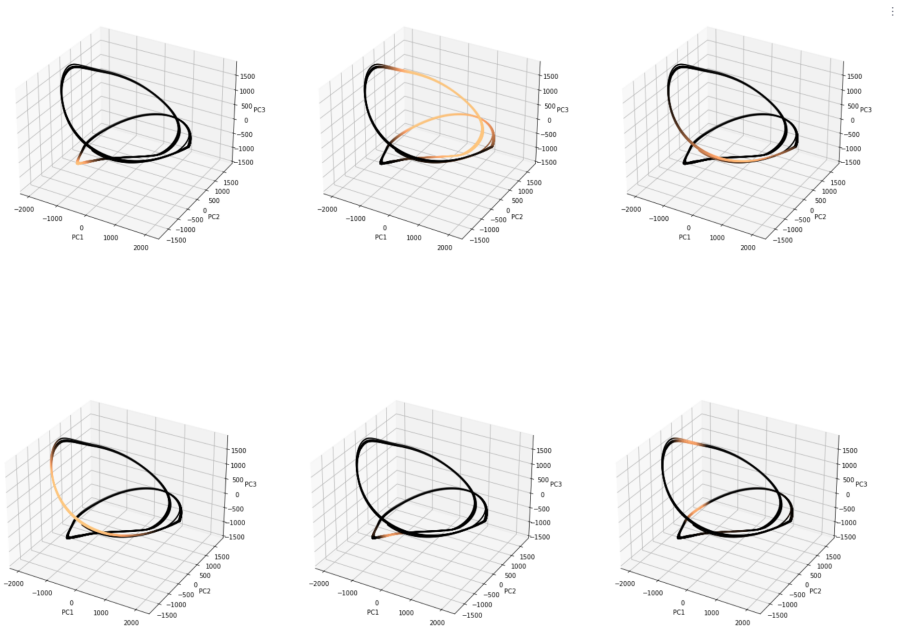


Figure 20: 3 principal components after PCA

Part 2 In this part, We have to create 9 plots of the PCA 3D curves, each representing a different measurements area and being colored respectively. To accomplish this, we need to pass the different columns as the color parameter to the scatter function. The results are shown in Figure 21



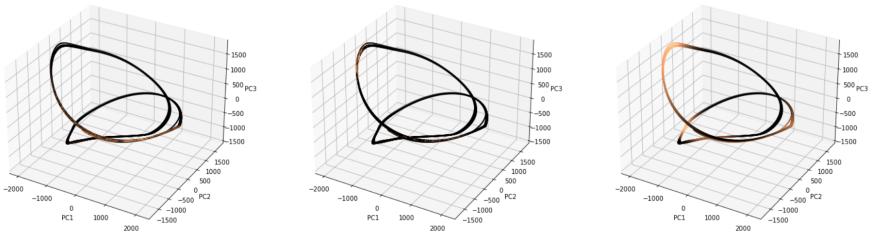


Figure 21: different measurement areas with different colors

Part 3 In this part, we are required to learn the dynamics on the curve embedded in the principal components, which is shown in the previous part. To do so we compute the arc-length of the curve, which is done by accumulating its value, so that being able to see its change over time. We also need to obtain the speed of arc-length, i.e. its first derivative in time, which is the vector field that we want. To do so we follow these steps:

- Iterate from the starting point and go through all points.
- for each couple of data we calculate the arc length by accumulating the L2 norm of consecutive points from the first point up to the current point.
- divide the arc-length by (dt = 1) or (i+1) (one for the instantaneous speed, another for the average speed.

By following the upper given step, we first obtain two figures, which are the cumulative Arc-length and the Instantaneous velocity over each time step.(they are shown in Figure 22 and 23, we only show one period here.)

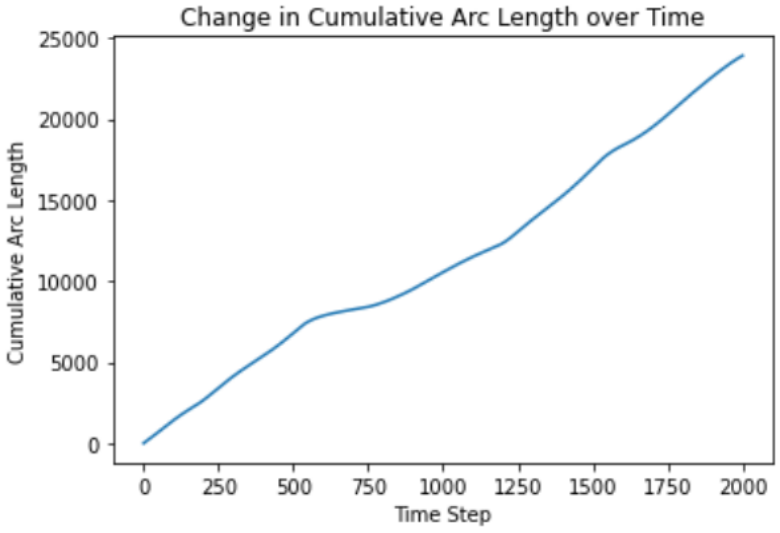


Figure 22: the cumulative Arc-length

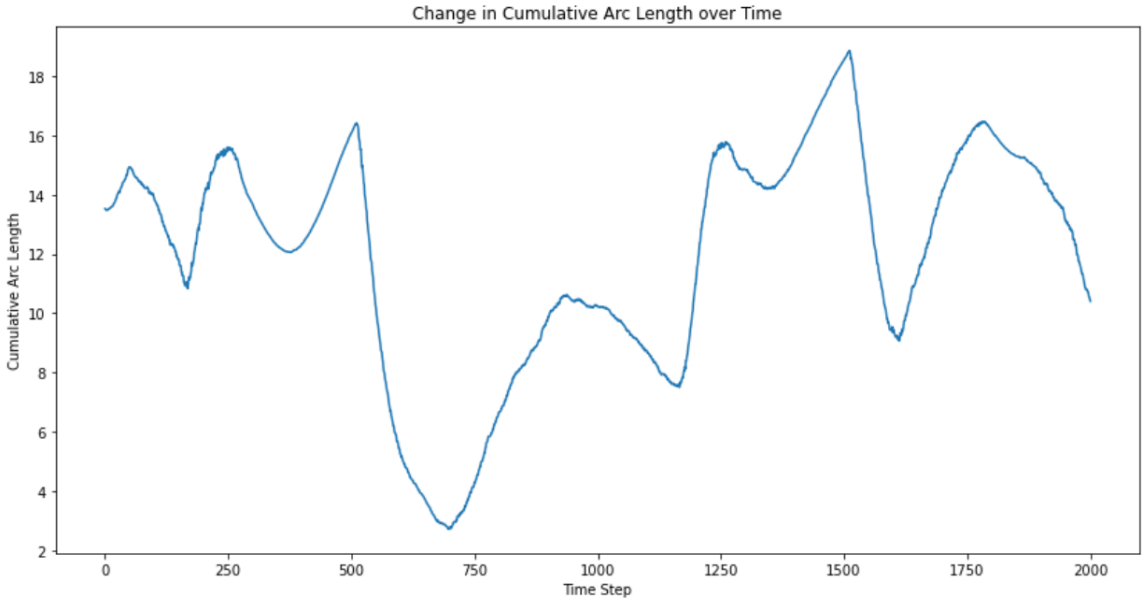


Figure 23: the Instantaneous velocity over each time step

But considering the actual state in our case, it's more reasonable to use the average speed, so we calculate it, convert it in to on period and plot it in Figure 24

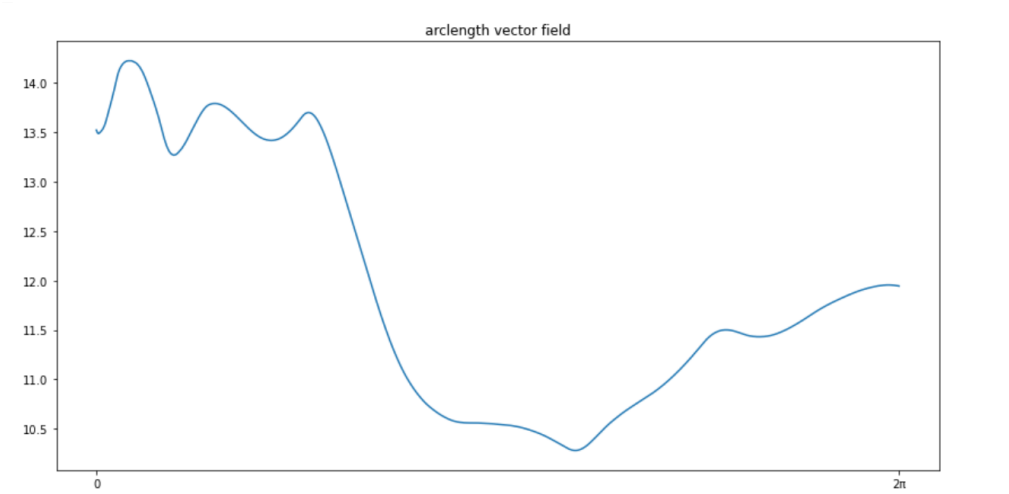


Figure 24: arc-length vector field (one full period)

Part 4 In the last part of task 5, we are required to predict the utilization of the MI building for 14 days, hoping the system learns the curve and correctly estimates future unknown points.

In order to finish this, we need to make use of the vector field that we created in the last part. Here we will make use of the **RBF** implementation, treating the arc-length data as points and the vector field as targets for the least square method to get the value pf our predicted are-length.

As instructed, once we have the arc-length values over time, we can start to create a mapping back to the original utilization value by using radial basis function approximation from arc-length space to the utilization value. Here we need to construct a radial basis function approximation from arc-length space to the utilization

value, so we can estimate future values.

Again we treat the arc-length values as points and the utilization values as targets for the least squares algorithm, effectively finding a function (written as ΦC) which takes one arc-length and gives one utilization value. The final result is shown in Figure 25

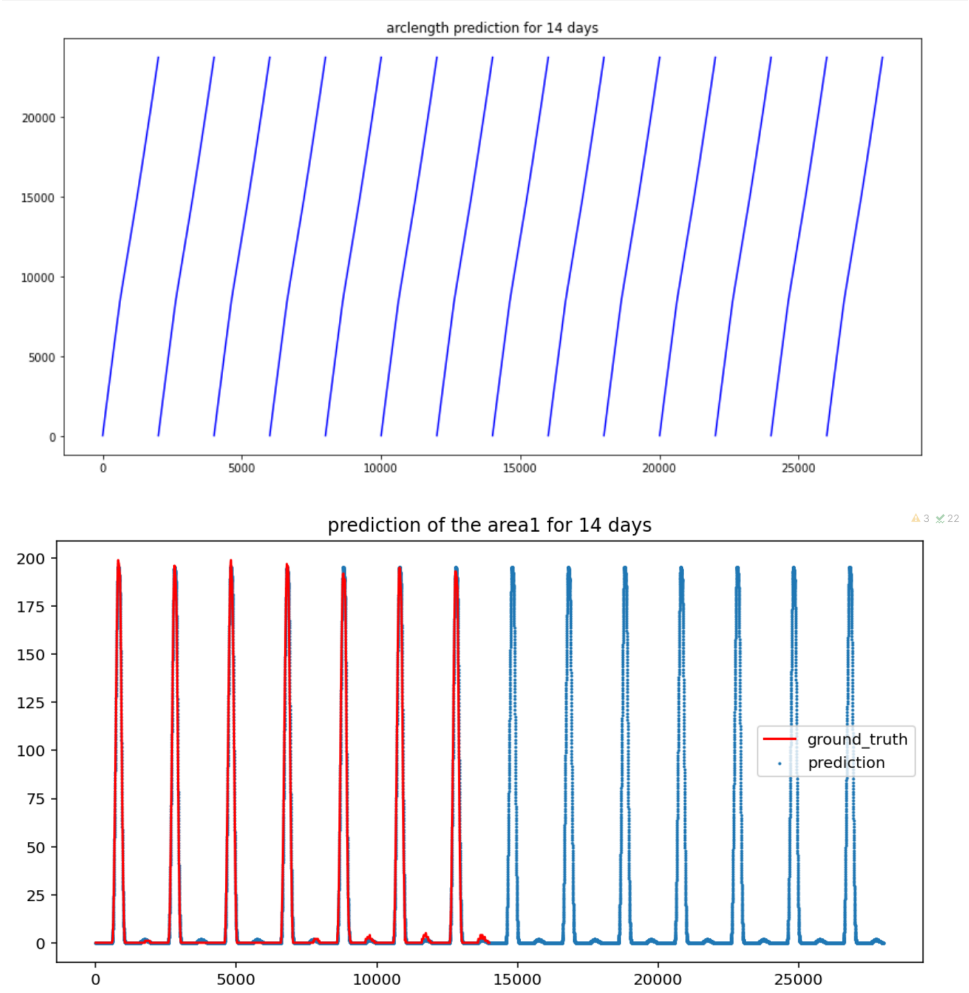


Figure 25: predicted arc-length and its corresponding utilization for 14 days