Report for exercise 5 from group B

Tasks addressed:	5
Authors:	HORIA TURCUMAN (03752553)
	GEORGI HRUSANOV (03714895)
	UPPILI SRINIVASAN (03734253)
Last compiled:	2023–01–19
Source code:	https://gitlab.com/turcumanhoria/crowd-modeling

The work on tasks was divided in the following way:

HORIA TURCUMAN (03752553)	Task 1	33.3%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
	Task 5	33.3%
GEORGI HRUSANOV (03714895)	Task 1	33.3%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
	Task 5	33.3%
UPPILI SRINIVASAN (03734253)	Task 1	33.3%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
	Task 5	33.3%

Report on task 1, Approximating functions

This exercise entails coming up with close approximations for linear and non-linear functions. In order to read the data, compute the radial basis functions, approximate linear and nonlinear data, and plot the approximated function over the actual data, we developed a Python script that we called `task_1_function_approximation.py`. The script contains a series of functions created in order to complete the given tasks.

Part 1:

The first part of this exercise was to approximate the function in dataset `linear_ function_data.txt` with a linear function. In order to achieve this, we splitted the main functionality that is used into several smaller ones. First of all we would need a function that loads the data, that we are going to work it. After that we would have a function that deals with the linear approximation aswell. Last but not least, we would like to visualize the data and the newly created approximation, so we also implemented some plotting functionality. The concrete specification of the methods implemented is as follows:

- Importing the essential modules, such as `numpy, pandas,matplotlib` is the first step in the code. After that, it specifies many functions to carry out a variety of activities:
- By calling the `load_data` function, it will read the dataset from the supplied path and load it into two numpy arrays. The first array, 'x array,' will include the values of 'x,' and the second array, 'y array,' will contain the values of 'f(x),' respectively. These two arrays are what the method returns.
- In order to provide an approximation of the function f(x) based on the values x and f that are provided, the `linear_approximation` function applies the least-squares minimization formula. In order to avoid any uncertainties and mistakes, the module `scipy.linalg.inv` from Scipy was used for calculation of the inverse matrix. This method provides the approximation function f(x) hat in the form of a numpy array as the return value.
- The real x-values and f-values are plotted against the approximated f(x) hat values via the `plot_linear_ approximation` function. The resulting picture could be seen on Figure1

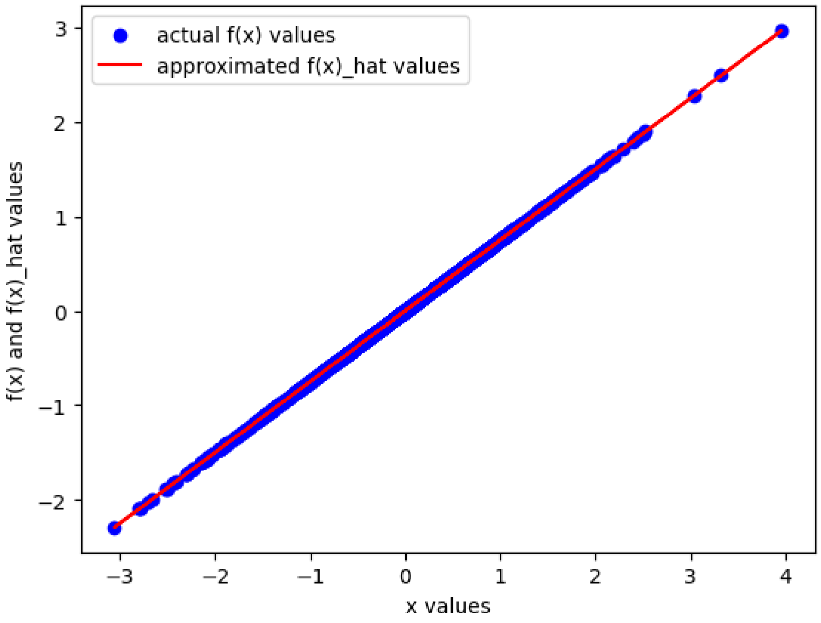


Figure 1: The linearly estimated function shown over the real linear data in the `linear_function_data.txt` file.

Part 2:

In the second part of this task, another linear approximation is made, but this time the data are nonlinear and can be found in the file `non_linearfunction_data.txt`. In order to load the data, we again use the `load_data` introduced in part 1, we just alter the path string for the new dataset. Otherwise, the procedure is precisely the same as it was for part 1. As one might expect, approximating such data with a straight line does not yield adequate results. Figure 2 shows a plot of the estimated function over the data.

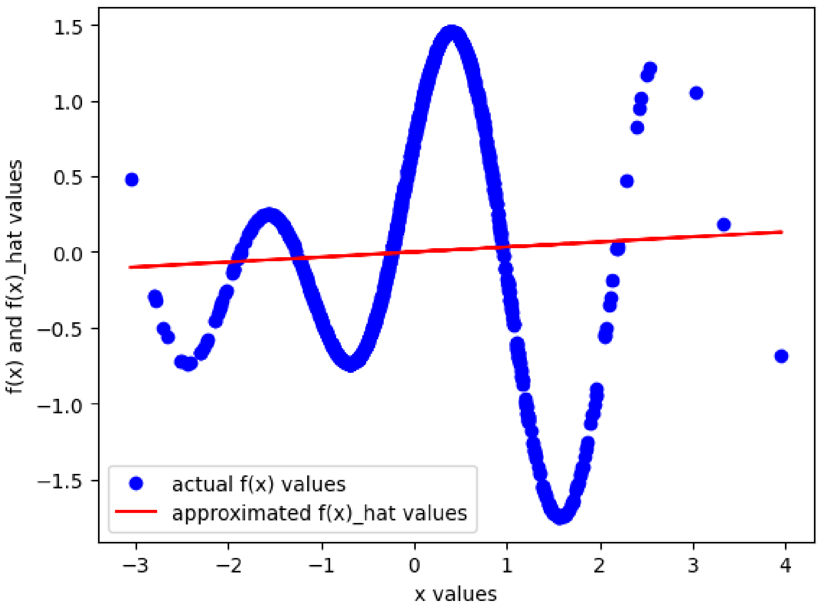


Figure 2: Plot of the linearly approximated function over the actual nonlinear data in the `nonlinear_function_data.txt` file.

Part 3:

The final task within the scope of exercise one was once again to approximate the function in dataset `non_linearfunction_data.txt` with a combination of radial functions. In order to be able to this we had to go through several things. As given in the materials for the exercise a linear decomposition into nonlinear basis functions is a representation of a nonlinear function that is utilized in a great deal of numerical techniques; we will investigate this particular representation of a nonlinear function. The fundamental concept is to represent the undiscovered function f as a composition of the functions already known as ϕ, in such a way that Eq1 is true.

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

This could be seen in various different examples.Here, we only considered special functions ϕ: so-called radial basis functions, defined by 2

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

The interesting part in the exercise was that we had to choose L and ϵ appropriately for this task. We also had to choose whether we should choose ϵ or ϵ^2 in the denominator for the radial functions.

In order to inspect different outcomes and options, we created an array with different and possible L and ϵ in order to be able to visually compare the generated plots.

The methods implemented for the task were:

- The function `radial_basis_function_approximation` employs the RBF approach in order to come up with an approximation of the function f(x) by making use of phi functions. The inputs that it requires are the path to the dataset, x array, L (the required number of ϕ functions), and ϵ(the bandwidth). After that, it shows the ϕ functions that were just generated for each point in the x array. After that, it determines the Coefficient matrix that will decide the peak of the ϕ functions in order to produce the $f(x)$ hat values that are approximately correct. In addition to this, it graphs the x-values in relation to the real and estimated f-values.

In the end a function called `approximate_data` was implemented and the whole idea behind it was to carry out the functionality of a "main" function. The path to the dataset is passed into the main function, which then calls the `load_data` function to load the dataset. After the dataset has been loaded, the main function runs the linear approximation function to estimate $f(x)$ by minimizing the sum of its squares. In order to plot the results, it sends the output of the load data function and the result of the linear approximation function to the plot linear approximation function.The main method has an argument called `method` which is to be set from the user, depending on what type of an estimation he wants to use, or is more suitable for his case. Depending on the value the `method` is set to, either the linear approximation and plotting or the radial basis function approach is executed. If the user tries to set it to something else, the `approximate_data` function will raise an error as this will not be supported by this program.

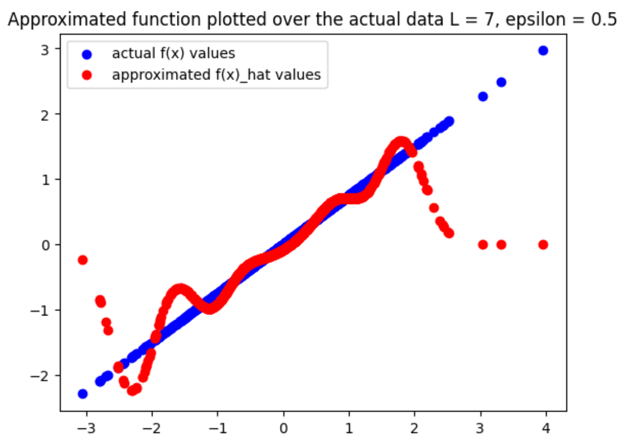


Figure 3: Approximated function plotted over the actual data L = 7, epsilon = 0.5

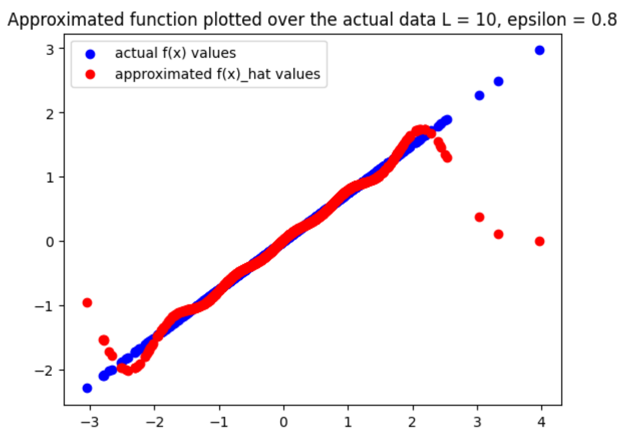


Figure 4: Approximated function plotted over the actual data L = 10, epsilon = 0.8

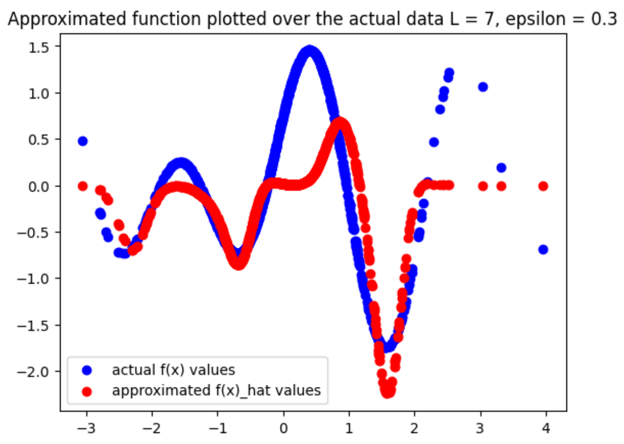


Figure 5: Approximated function plotted over the actual data L = 10, epsilon = 0.3

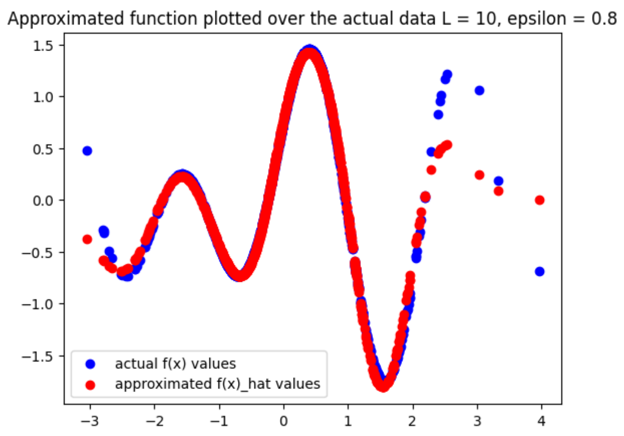


Figure 6: Approximated function plotted over the actual data L = 10, epsilon = 0.8

Report on task 2, Approximating linear vector fields

This task's primary objective is to generate a close approximation of a collection of two-dimensional linear vector field data. In specifically, we are provided with two points in time for the dataset, x0 and x1, with points in x0 arriving at x1 after a specific time that is denoted by the variable Δt.

To compute the normative solution for the supervised learning issue associated with the vector field is a simplistic but uncomplicated technique to transform this situation into the standard form of the problem was given in the exercise sheet itself, namely:

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

Part 1:
The computation of the V vector is done right after the data is loaded in the method `load_data`. Both `linear_vectorfield_data_x0.txt and linear_vectorfield_data_x1.txt` files are loaded and right after that the vector V is calculated with Eq 3. So this completes the first part of this task. In Figure9 we can see how the points in both datasets are allocated. Also the direction is displayed in Figure 37

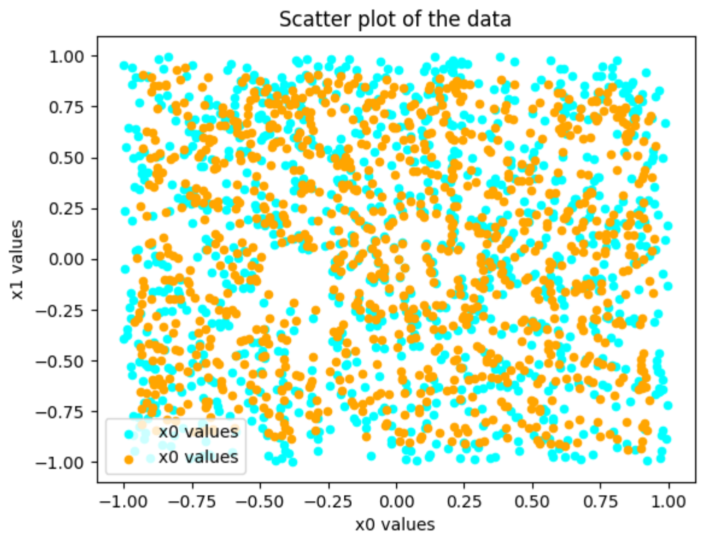


Figure 7: Datasets from both files x0 (cyan) and x1 (orange),scattered in two dimensions.

Part 2:
Now in this part the first thing we had to do is to calculate the matrix $\hat{A}$. This is done with the Eq 4.

$$\hat{A}^T = (X^T X)^1 X^T F \tag{4}$$

Now when we have successfully computed the matrix $\hat{A}$ A, we had to solve the linear system $x = \hat{A}x$ with all $x_0^{(k)}$ as initial points, up to a time $T_{end} = \Delta t = 0.1$. The result of this calculation will provide us with the values for points $x_1^{(k)}$. Once we have the these values we can compute the mean squared error to all the known points $x_1^{(k)}$.

For better visability within the created `.py` file we created three different methods that fullfill these smaller task in order in the end to have our final result. Once we run all the methods for this task, we get an MSE score of 0.0015299638198721132.

- The initial state and its velocity are provided as inputs to a hat. It then approximates the matrix A that represents the system by employing the least-squares method.
- We then take $\hat{A}$ which acts like an input to the compute `compute_X1_approx`function. The `solve_ivp` function from the `scipy library` is then invoked to estimate the system state for each initial state at the following time step ($t = 0.1$). The system then returns an array of these approximations of its state.
- The mean squared error between the current state and the expected state at the following time step is the result of the `calculate_MSE` function. The function is given as inputs the exact state at the following time step as well as the true state at that step in time.

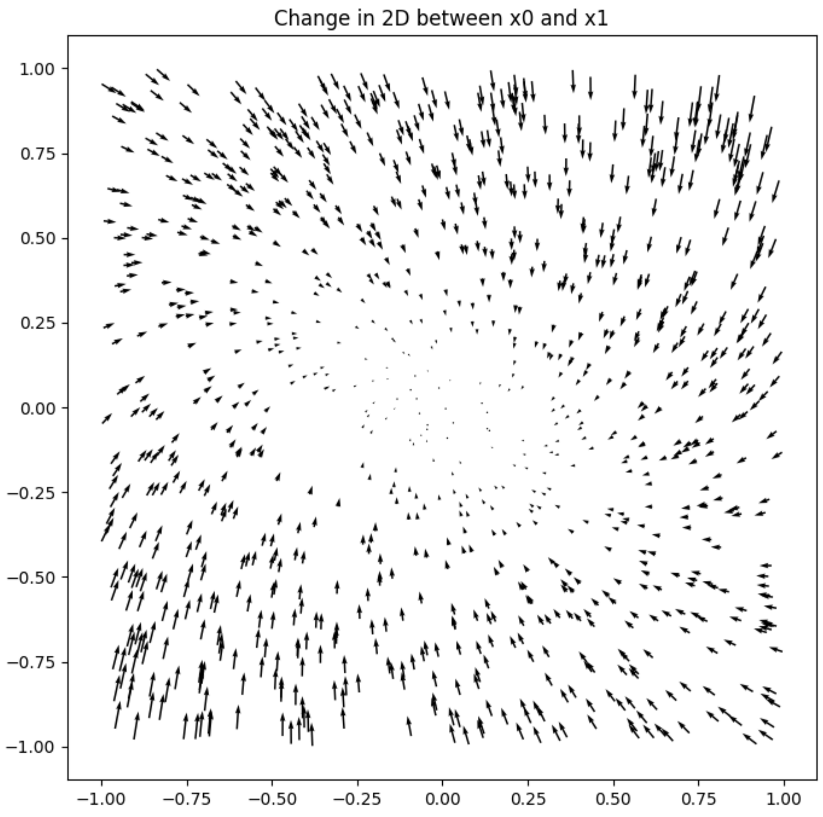


Figure 8: The simulated trajectory of the x0 datapoints.

Part 3:
In the final part of this exercise we had to select the initial point to be at (10, 10), which is located very far outside the initial data. Once again just like in the previous task we had to, solve the linear system using your matrix approximation for the case where $T_{end} = 100$ and visualize the trajectory as well as the phase portrait in a domain that ranges from -10 to 10. Using pictures 9,37,11, you can see the visual result of the task given. The first picture shows the plot of the points themselves. The second picture shows the trajectory of each point and then the final picture shows the behaviour with the help of the phase portrait.

Once again, we have a special function within the `.py` file which acts like a main function which delegate the behaviour of the code. In the scope of the method, we first load the data from the two given datasets with the V vector. After that all necessary values are calculated with the methods responsible for that. And after that the three visualizations are plotted for the user.

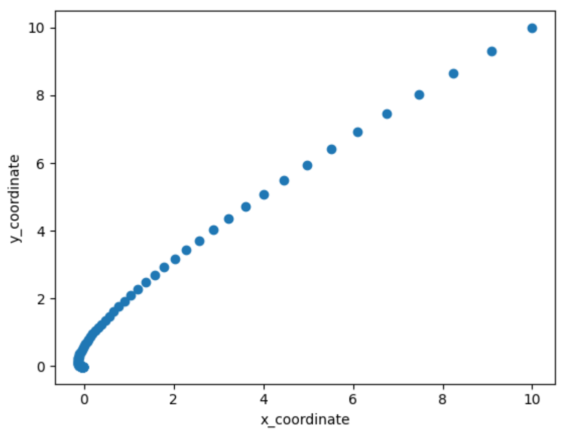


Figure 9: Linearly approximated vector field.

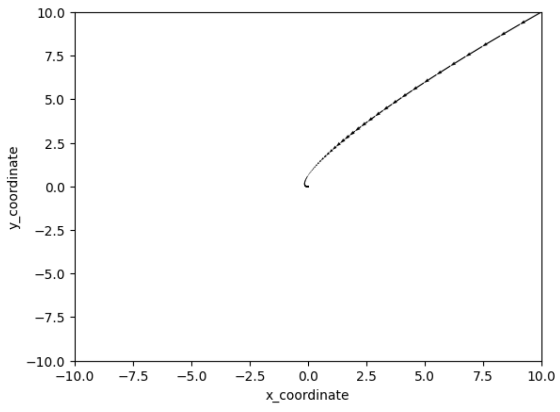


Figure 10: The trajectory of linearly approximated vector field.

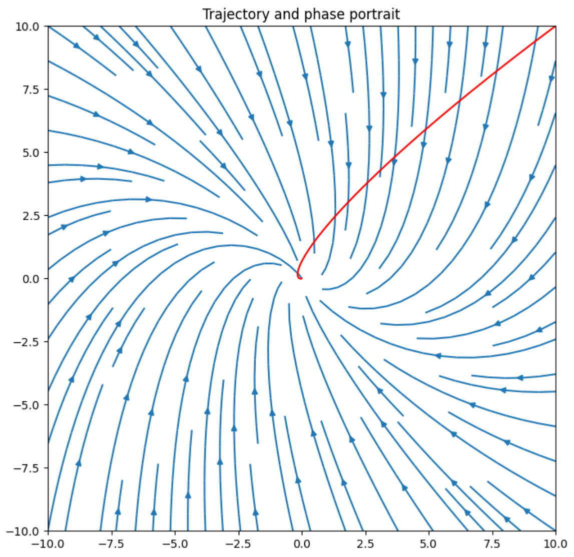


Figure 11: Phase Portrait and the linearly approximated trajectory of vector field.

Report on task 3, Approximating nonlinear vector fields

This task revolves around working on a given nonlinear vector field dataset composed once again of two moments in time, starting points x0 and points x1 after an unknown $\triangle t$.
The first task involves approximating the vector field mentioned in the file 'nonlinear-function-data.txt'. Below figure shows the plot of the vector field.

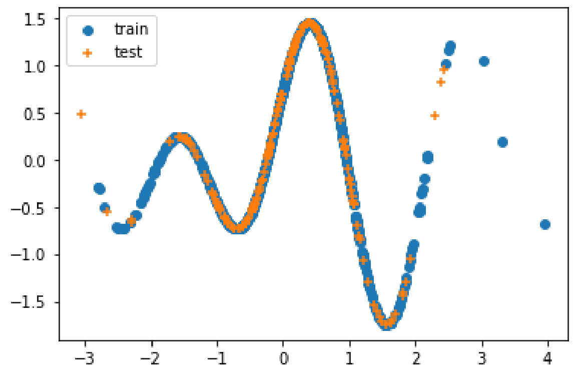
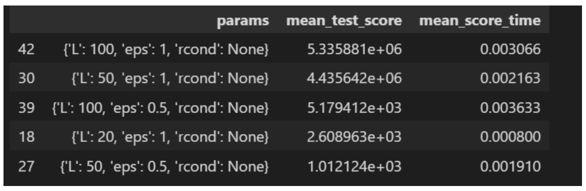


Figure 12: Plot of Train-test data

By using the RBF function and performing grid search, we find the best parameters based on low mean test score as shown in below image.
$params = {'eps' : [0.1, 0.5, 1, 5], 'L' : [10, 20, 50, 100], 'rcond' : [None, 0.005, 0.01]}$
$rbf = RBF(0, 0)$
$rbf - cv = GridSearchCV(rbf, params, scoring = lambdamod, , : 1./MSE(fB - test, mod.predict(xB - test)))$
$rbf - cv.fit(xB - train, fB - train)$



	params	mean_test_score	mean_score_time
42	{'L': 100, 'eps': 1, 'rcond': None}	5.335881e+06	0.003066
30	{'L': 50, 'eps': 1, 'rcond': None}	4.435642e+06	0.002163
39	{'L': 100, 'eps': 0.5, 'rcond': None}	5.179412e+03	0.003633
18	{'L': 20, 'eps': 1, 'rcond': None}	2.608963e+03	0.000800
27	{'L': 50, 'eps': 0.5, 'rcond': None}	1.012124e+03	0.001910

Figure 13: Mean test scores

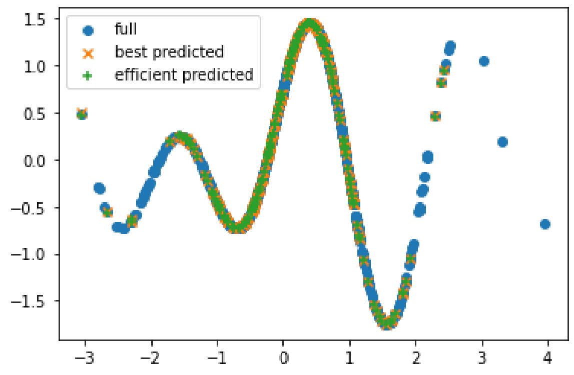


Figure 14: Plot showing best predicted

The next task is to approximate the 2 datasets nonlinear-vectorfield-data-x0.txt and nonlinear-vectorfield-data-x0.txt and estimate x1 with respect to x1 using a linear operator. We fix $dt = 0.1$ and train the dataset. Below are the images of scatter plots and quiver plots of datasets.

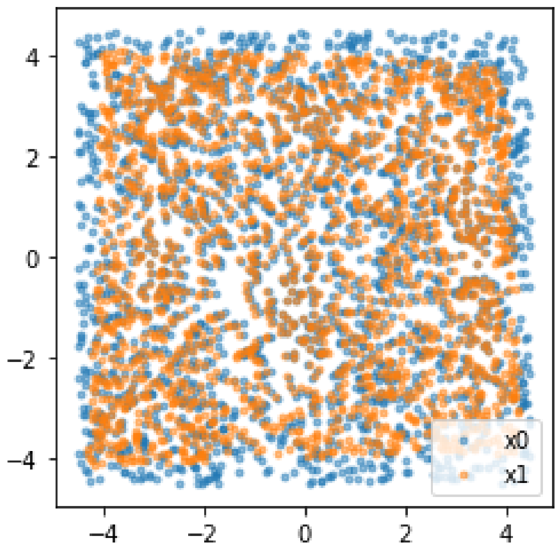


Figure 15: Scatter Plot of x0 and x1

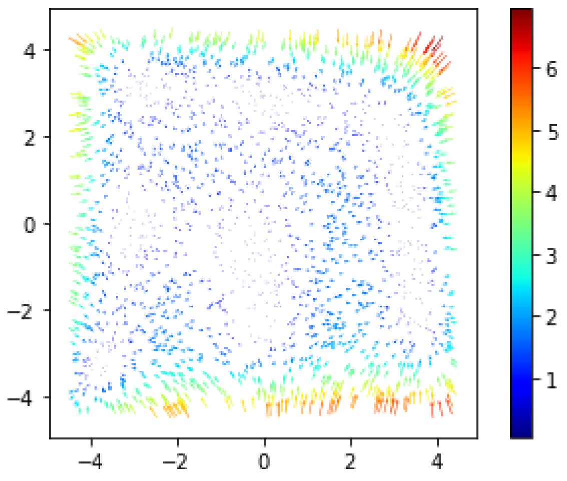
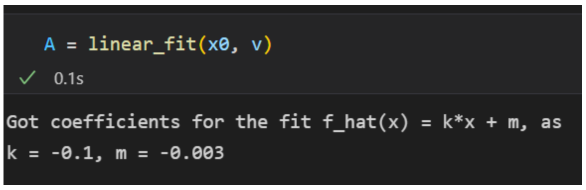


Figure 16: Quiver plot

We define $v = (x1 - x0)/dt$ and linearly fit v with x0. Following are the results for the same.



```
A = linear_fit(x0, v)
✓ 0.1s
Got coefficients for the fit f_hat(x) = k*x + m, as
k = -0.1, m = -0.003
```

Figure 17: Linear fit results for v and x0

Following is the quiver plot of the approximated function.

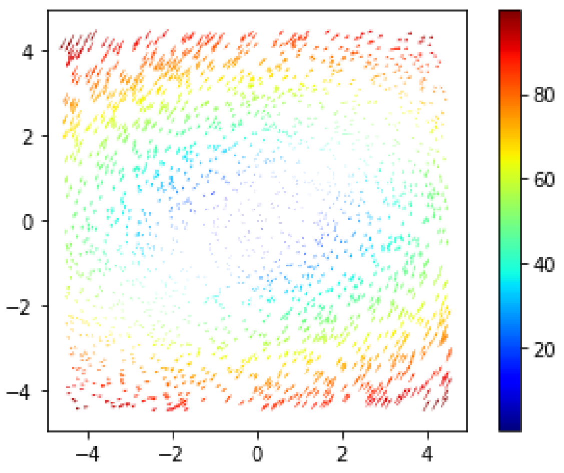
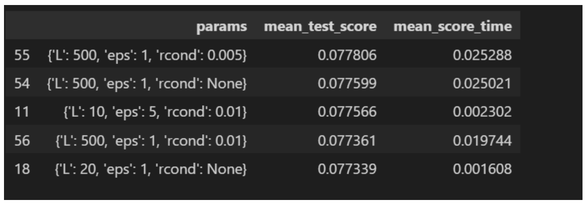


Figure 18: Quiver plot of approximated function

We use the same method as before to calculate the best parameters using grid search and mean test score as criteria.



	params	mean_test_score	mean_score_time
55	{'L': 500, 'eps': 1, 'rcond': 0.005}	0.077806	0.025288
54	{'L': 500, 'eps': 1, 'rcond': None}	0.077599	0.025021
11	{'L': 10, 'eps': 5, 'rcond': 0.01}	0.077566	0.002302
56	{'L': 500, 'eps': 1, 'rcond': 0.01}	0.077361	0.019744
18	{'L': 20, 'eps': 1, 'rcond': None}	0.077339	0.001608

Figure 19: Mean test scores

After trying to linearly approximate the vector field, we are asked to approximate it non-linearly, using the same RBF function we described before. To find the best parameters, we applied a grid search. We use the RBF(20,5) for the following results. We obtain an MSE of 0.050457 and below is the quiver plot for the same.

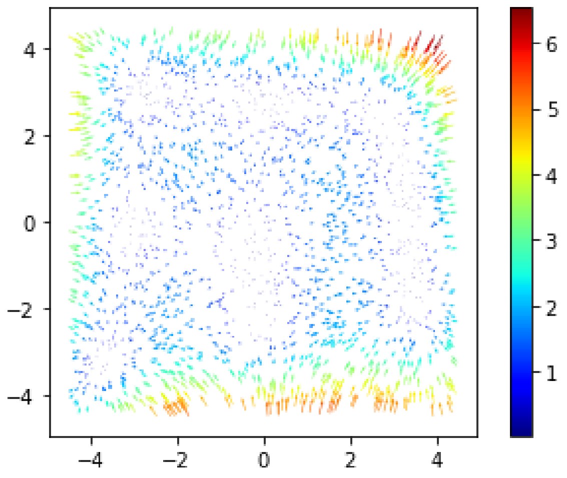


Figure 20: Quiver plot for RBF(20,5)

Eventually, since the results proved the validity of the non linear approximation, we can solve the system starting from all the initial points x0 for a much longer time. The non linear vector field shown in below figure already shows five apparent steady states, with four of them being close to the corners (attracting steady states) and one being near the origin (repulsive steady state). To answer the final question, the non linearly approximated system cannot be topologically equivalent to the linearly approximated one because of their different number in steady states, as dynamical systems theory asserts.

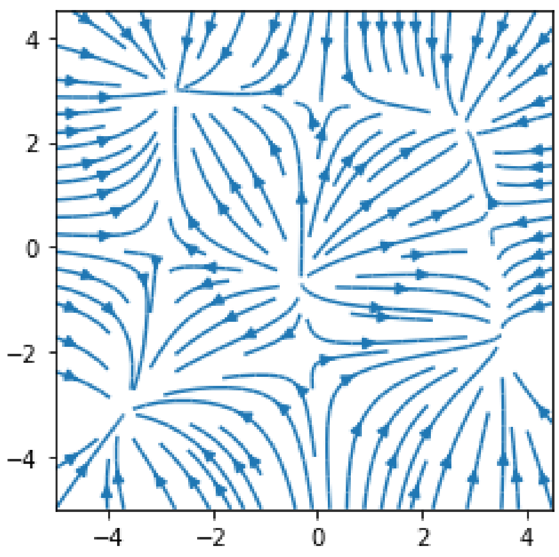


Figure 21: Best non linear approximation vector field

Report on task 4, Time-delay embedding

Part 1:
After reading the dataset takens-1.txt, we plot the closed 1D manifold and plot first coordinate with respect to time (without delay)

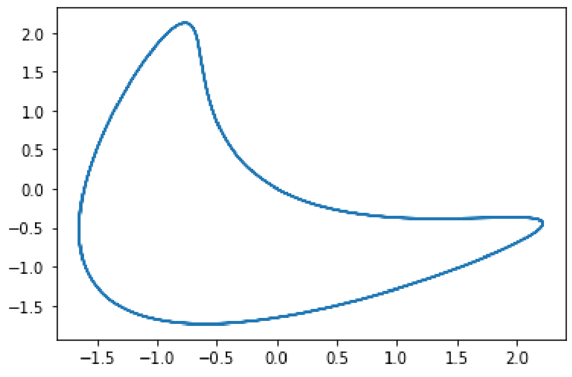


Figure 22: Closed 1D manifold

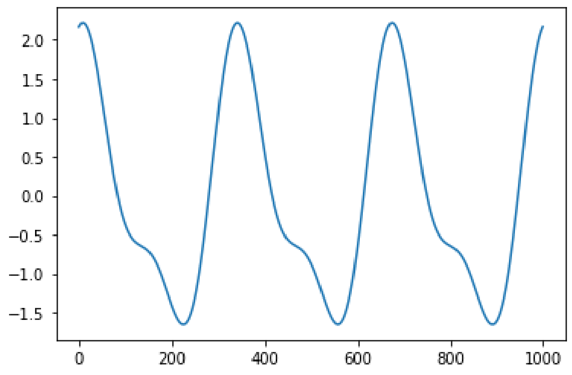


Figure 23: First coordinate wrt time (without delay)

Subsequently, we plotted the same data, but this time not against the line number, but against a delayed version of itself, namely $x(t + \triangle t)$. We choose a delay of 10 rows. Following are some of the images of plots with delay.

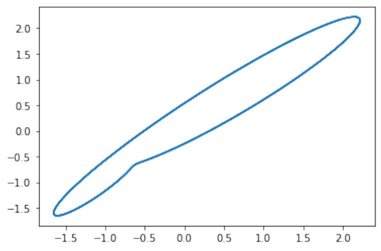


Figure 24: First coordinate against itself delayed by 10

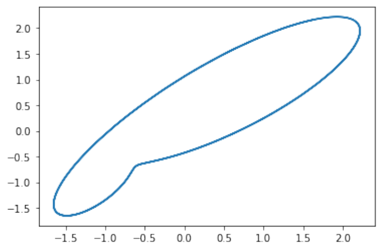


Figure 25: First coordinate against itself delayed by 20.

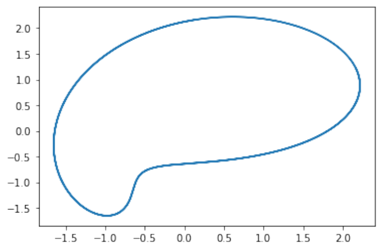


Figure 26: First coordinate against itself delayed by 50.

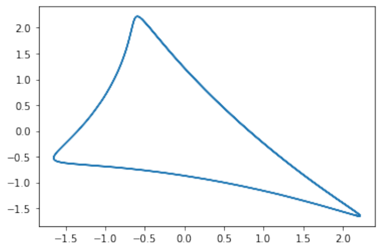


Figure 27: First coordinate against itself delayed by 220

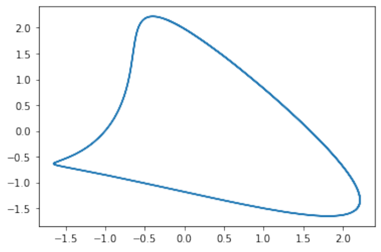


Figure 28: First coordinate against itself delayed by 240.

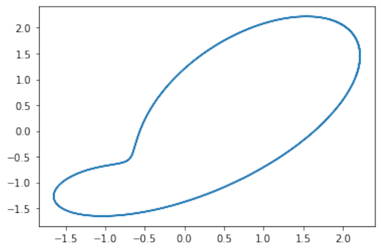


Figure 29: First coordinate against itself delayed by 300.

Following is the embedded plot of time delay of 0, 10 and 20 rows.

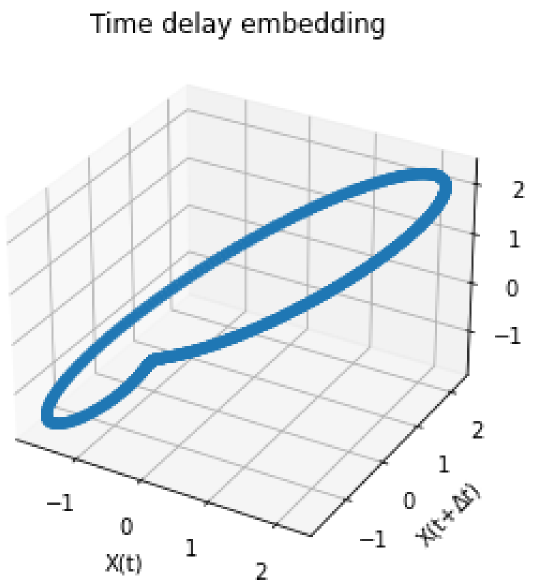


Figure 30: Embedded Plot manifold

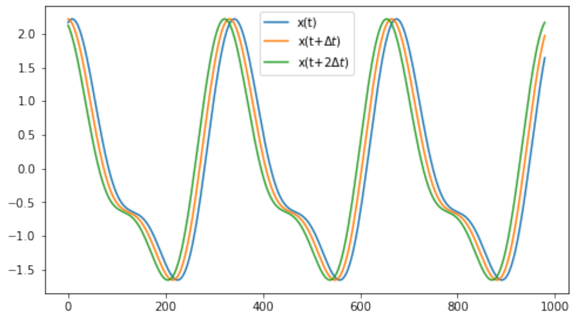


Figure 31: Embedded Plot

Following is the embedded plot of time delay of 0, 150 and 300 rows.

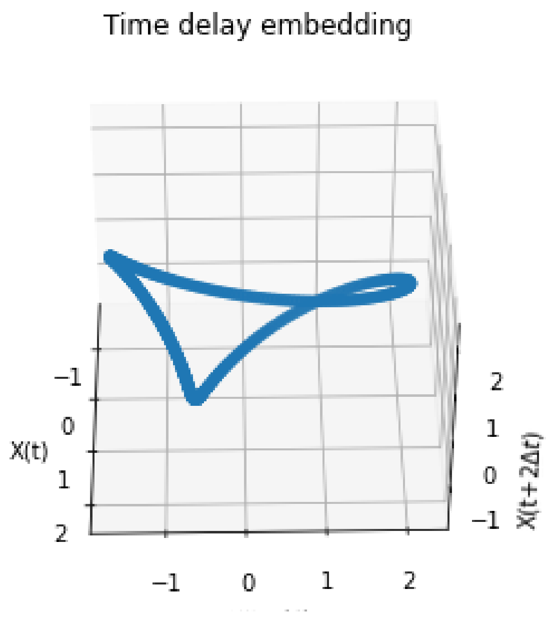


Figure 32: Embedded Plot manifold

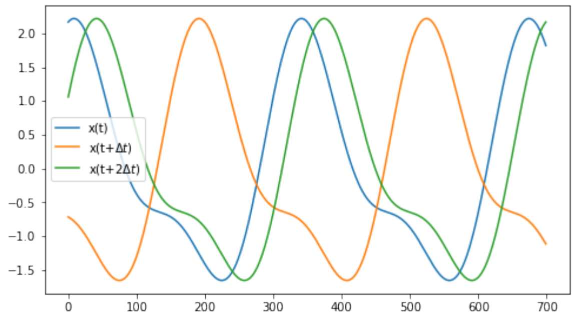


Figure 33: Embedded Plot

According to Takens theorem, being the manifold of dimension $d = 1$, it is assured to get an embedding using $2d + 1 = 3$ coordinates. Since there are no intersection in the plot, two coordinates are sufficient to embed this manifold, if a proper time delay is chosen.

Part 2:

In this part, we test Takens theorem with the chaotic system called the Lorenz attractor. Setting $\sigma = 10, \rho = 28, \beta = \frac{8}{3}$ and the starting point equal to (10, 10, 10). Now we test Takens theorem by choosing a $\triangle t$ and plotting x(t) against $x(t+\triangle t)$ and $x(t+2\triangle t)$, being x the first coordinate of the Lorenz system. As it it possible to observe, using the x coordinate we still get a reasonable shape that resembles the original one, while with the z coordinate this does not happen, so the latter is not suitable to represent the original system through a time delay, at least when using two delays. This probably happens because for each value of z there are multiple values of the system, and these are found in the two lobes. In some sense, the two lobes overlap when seen from z, and that is why we get a circle-like shape.

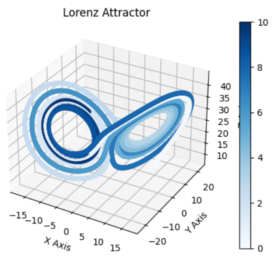


Figure 34: Lorenz Attractor

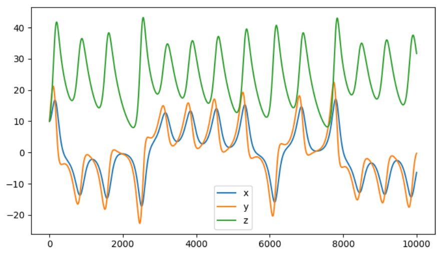


Figure 35: Plot of x,y,z with t

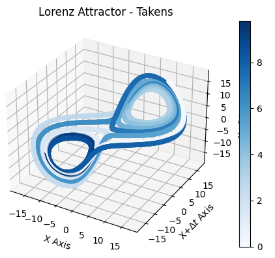


Figure 36: The Lorenz attractor when plotted using time delays on the x coordinate

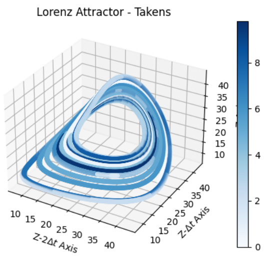


Figure 37: The Lorenz attractor when plotted using time delays on the z coordinate

Report on task 5, Learning crowd dynamics

1. The given file contains 10 columns, 9 of which represent the number of people in different buildings of the campus. Therefore the R^9 is a natural choice for the state space. The exercises statement suggests that the manifold is probably one dimensional $d = 1$, therefore, according to Takens theorem one can embed it $2 \cdot d + 1 = 3$ dimensions. The pca values over time are shown in figure 38

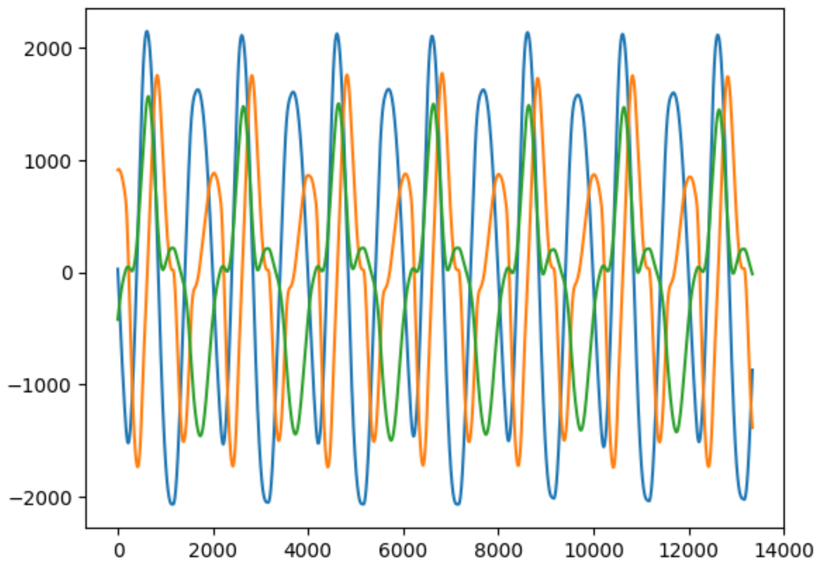


Figure 38: PCA values over time

2. Coloring the points allow us to visualize which represent high occupancy of rooms and which don't. The results are shown in 39.

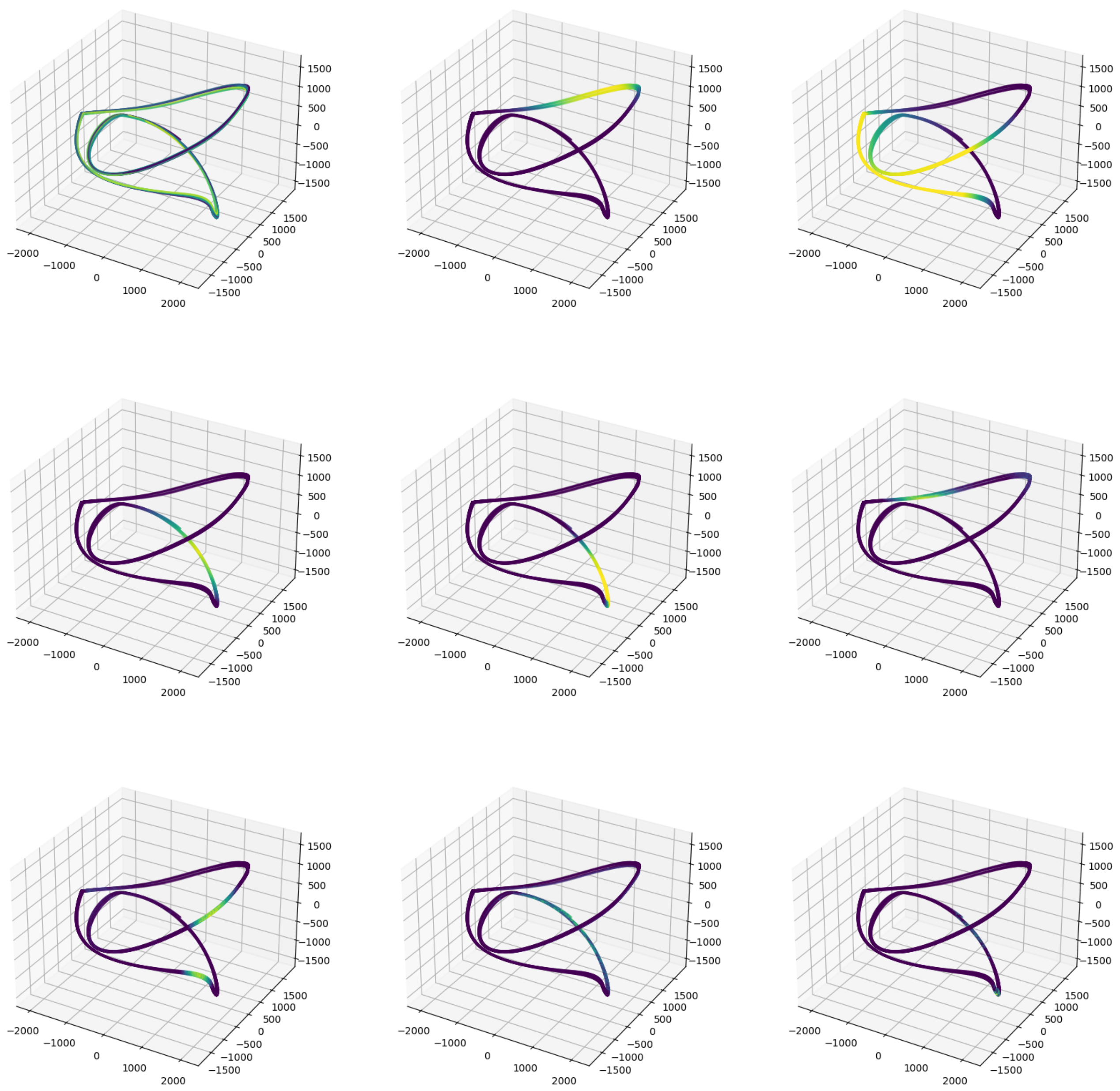


Figure 39: Occupancy of each of the rooms.

3. The arclength of each point can be computed by summing up the distances from consecutive points. The speeds are then the arclengths divided by the respective time. The result is shown in 40 together with the predicted version using non-linear approximation 41.

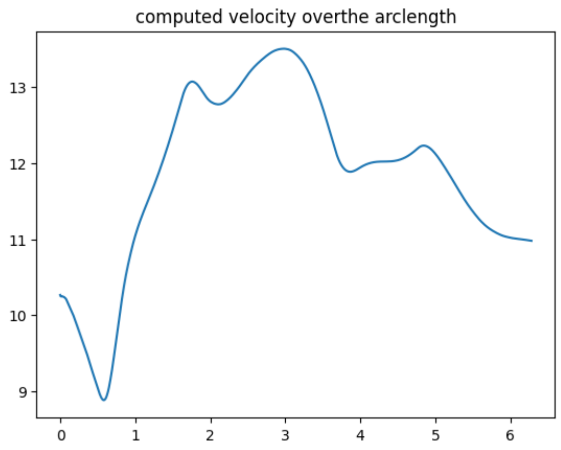


Figure 40: Computed arclength speed over arclength.

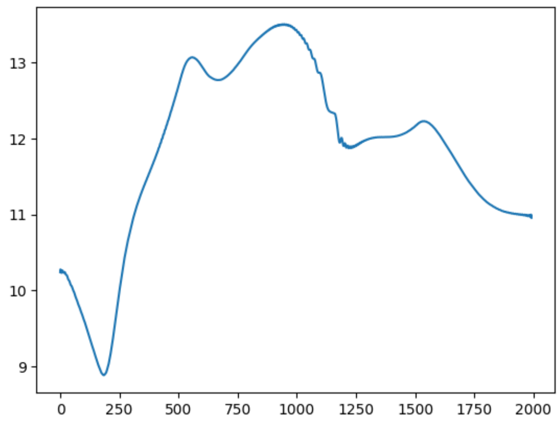


Figure 41: Approximated arclength speed over arclength.

4. To solve this task we predict the arclengths over 14 days starting with the first data point 42. After this we use non linear approximation to create a mapping from arclength to the occupancy of the first room 43. Last figure 44 shows the difference between the real measurements and predicted occupancy and the predicted occupancy over all 14 days.

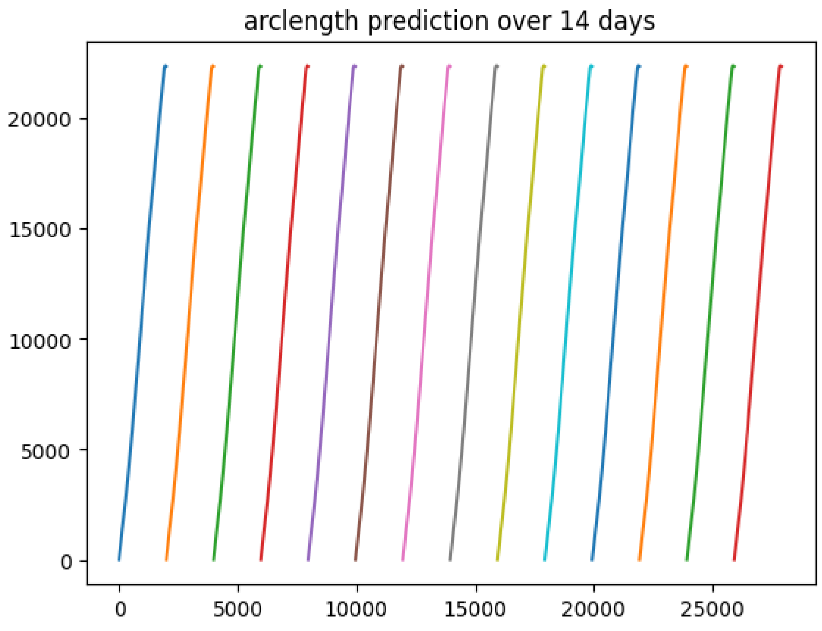


Figure 42: Occupancy of each of the rooms.

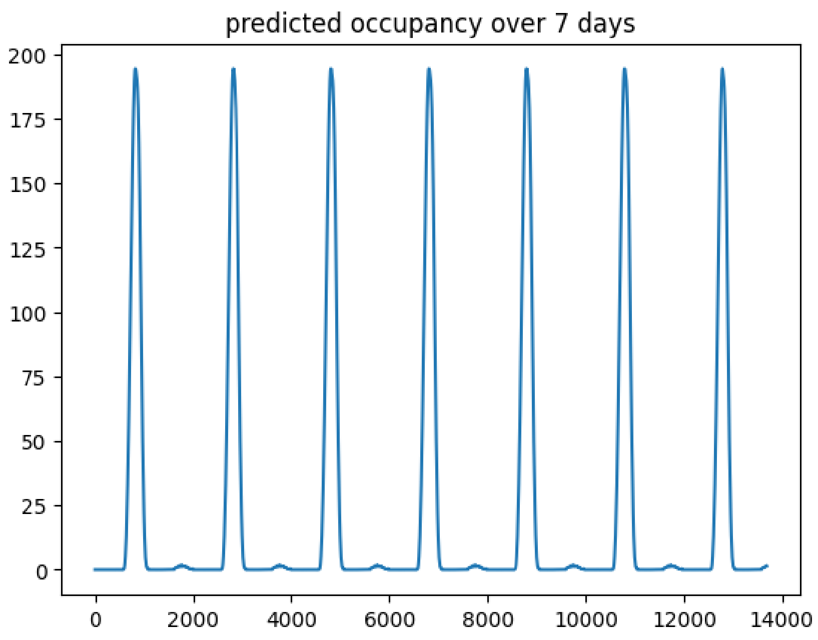


Figure 43: Occupancy of each of the rooms.

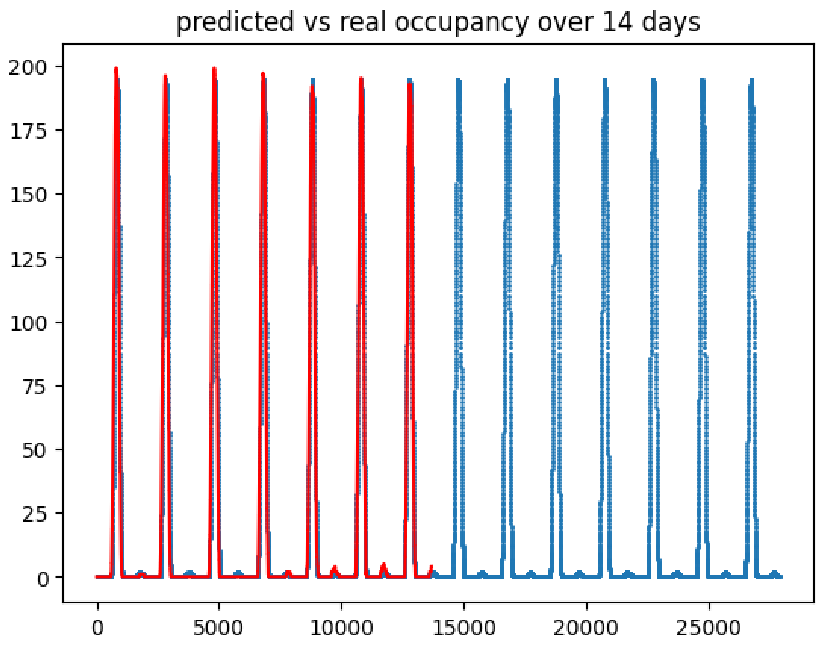


Figure 44: Occupancy of each of the rooms.

References