Report for exercise 5 from group A

Tasks addressed:	5
Authors:	MELIH MERT AKSOY (03716847)
	ATAMERT RAHMA (03711801)
	ARDA YAZGAN (03710782)
Last compiled:	2022–07–06
Source code:	https://github.com/mlcms-SS22-Group-A/mlcms-SS22-Group-A/tree/exercise-5

The work on tasks was divided in the following way:

MELIH MERT AKSOY (03716847)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
ATAMERT RAHMA (03711801)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
ARDA YAZGAN (03710782)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%

numpy	1.20.3
matplotlib	3.4.3
scipy	1.7.1
sklearn	0.24.2
pandas	1.3.4

Table 1: Software versions.

Software that is used in this report with corresponding versions can be seen on table 1.

Report on task Task 1/5: Approximating functions

In this task we first download the given datasets with 1000 one-dimensional points in the euclidean space `linear_function_data.txt` (A) and `nonlinear_function_data.txt` (B) from moodle and load them using `numpy.loadtxt`. We immediately notice that the given data are two dimensional, meaning that the functions that we want to approximate can map one dimensional euclidean space $\mathbb{R}$ to another one dimensional euclidean space $\mathbb{R}$. So in this case we are given some data $X = \{x^k\}_{k=1}^{1000} \subset \mathbb{R}$ and function values $F = \{f(x^k)\}_{k=1}^{1000} \subset \mathbb{R}$ and we want to construct $\hat{f} : \mathbb{R} \to \mathbb{R}$ such that the error $e(\hat{f}) = ||f(X) - \hat{f}(X)||^2 = ||F - \hat{f}(X)||^2$ is minimized.

In the following we approximate these given functions using the observation (given sample points). Since we use least-squares minimization for all the approximations we use the same library function `lstsq` as suggested in the exercise sheet from the module `scipy.linalg`. We omit the bias terms since the linear data passes through the origin and it is also ignored in the lecture for simplicity. However we know that it can be easily added by appending a column of ones in the design matrix.

Also it is important to note that we split the given dataset into train (%80) and test (%20) and approximate the function using the train set. We do this in order to see if the approximated function is overfitting by plotting it on top of the test set.

First part

First we approximate (A) with a linear function. We could fit the data quite well as shown in figure 1. Radial basis functions are non-linear which means even if we use high degree polynomials we cannot fit the linear data well because of oscillations. We get very low validation loss as well as training loss even if we do not use any `cond` parameter mentioned in the part.

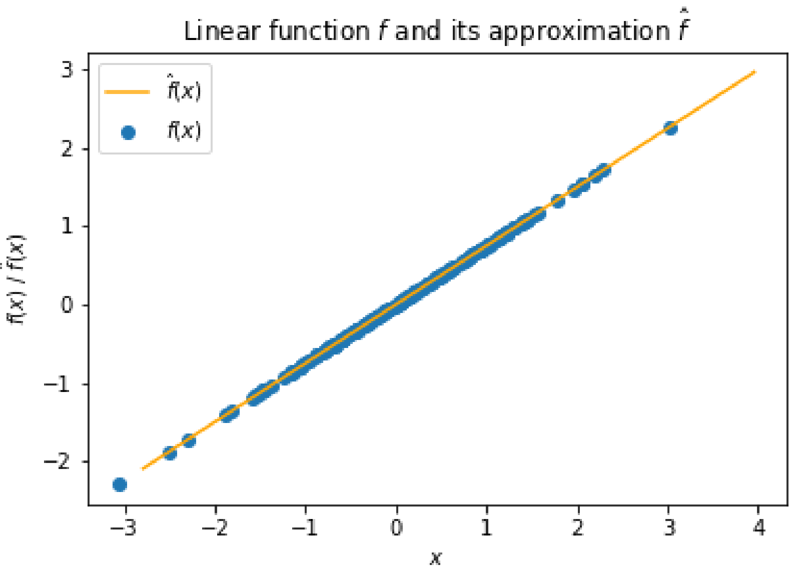

Figure 1: Linear function is approximated using a linear function.

Second part

Next we approximate the function (B) with a linear function in figure 2. Here we cannot fit the sample data well and get high errors as seen in figure since the given data points are produced (possibly) form a non-linear function. The `cond` parameter does not help us in this case because we are not able to overfit the training set. In this case we are actually underfitting since we are using a lower degree polynomial to approximate the dataset than its actual degree.

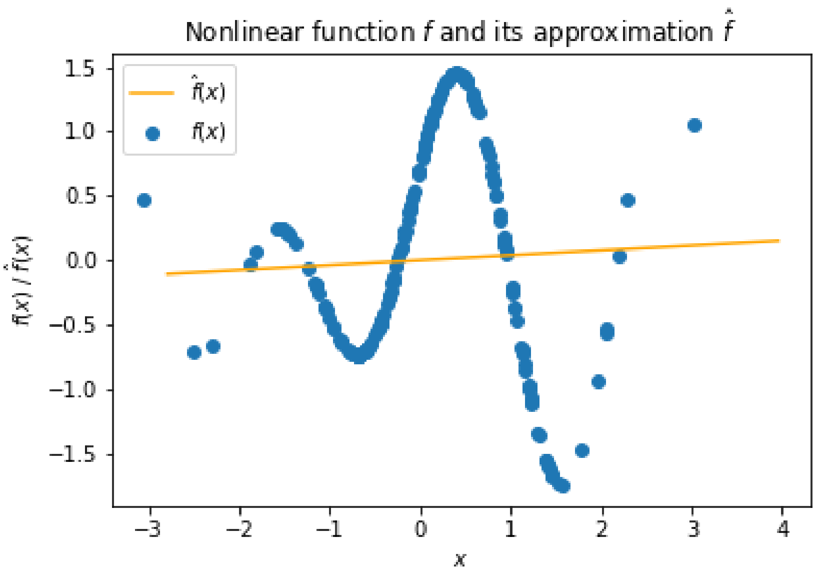

Figure 2: Non-linear function is approximated using a linear function (underfitting).

Third part

Now we use linear combination of radial basis functions given in the exercise sheet for the dataset (B). and get the results in figure 4. Here we compute the ϵ value using the same technique used in diffusion maps in the previous exercise. We have computed the distance between the centers of the radial basis functions and the data points which resulted in a non-square matrix that contained for each of the x_l its distance to all the points in the data. Then we find the maximum of these distances and take a percentage of this distance as the ϵ value. The percentage value is the hyperparameter in our method of calculating ϵ and the default value of the percentage is 0.05 as it was in the last exercise.

The number of basis functions actually determines the degree of the polynomial (linear in the basis functions) / function that we use to approximate the training data. If we select a very high number number then we overfit the dataset as depicted in figure 3.

Instead if we only use 7 basis functions we get the result in figure 4.

In order to visualize the effect of the `cond` parameter we have approximated the dataset using 8 basis functions and first we set the `cond` parameter to 0 and get the result in figure 5.

If we set the `cond` to 0.06 we get the visualization in figure 5. Here we can see that previously the function had a very steep down curve between 3 and 4 x values. But if we set the condition parameter this is avoid a little and we get a smoother transition – better generalization. Therefore we can use the `cond` parameter when lower numbers of radial basis functions are used to approximate the data (but still overfits it) for regularization. Since the given data points very close to each other and have a higher intensity, using higher number of basis functions can perfectly approximate a function without any `cond` value set.

Report on task Task 2/5: Approximating linear vector fields

We first download the dataset `linear_vectorfield_data_x0.txt` and `linear_vectorfield_data_x1.txt` using the `loadtxt` function from the `numpy` module. Datasets together contains 1000 vectors from x0 to x1. The vectorfield can be obtained by subtracting x0 coordinates from the x1 coordinates and dividing them by the timestep Δt.

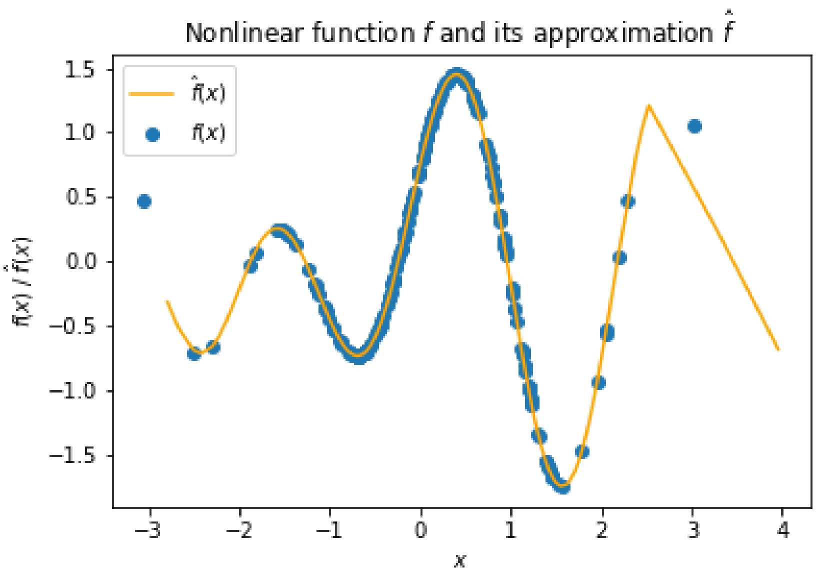

Figure 3: Non-linear function is approximated using a non-linear function with 16 basis functions.

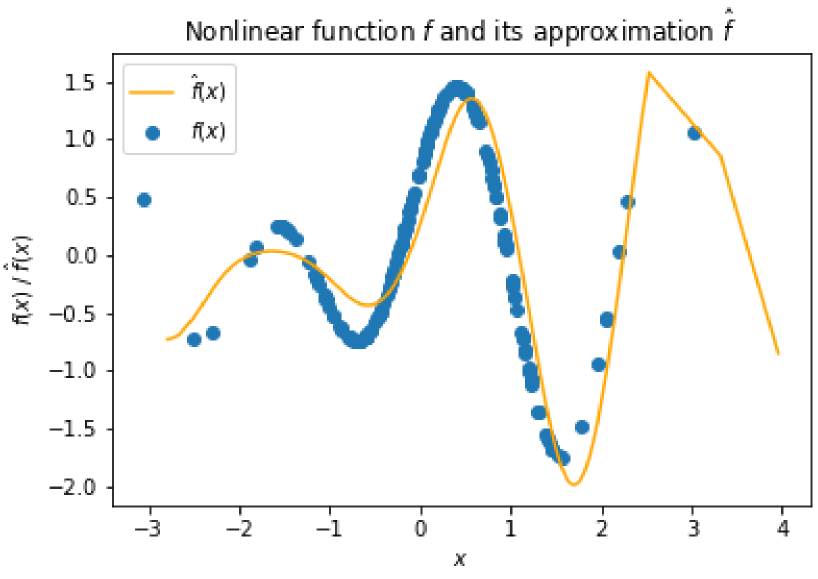

Figure 4: Non-linear function is approximated using a non-linear function.

Part one

In this part we have to estimate the vector field using the following equation:

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

In the exercise sheet it was asked to minimize the approximation error (MSE), where we should have come up with an optimization technique, but since we could not implement one, we have tried some Δt values in the range $[0.05, 0.45]$ with a step size of 0.05. Timestep of 0.1 has produced the minimum approximation error. We approximate the matrix A in:

$$v(x_0^k) = v^{(k)} = Ax_0^{(k)}$$

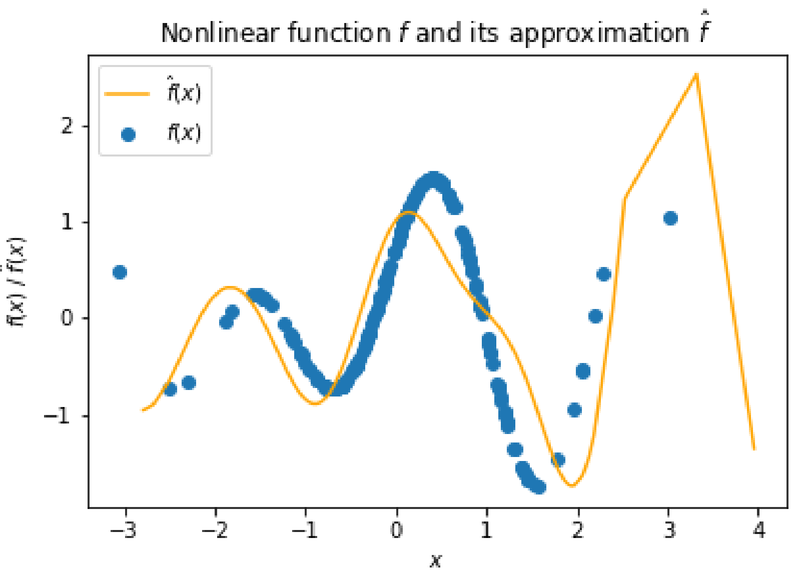

Figure 5: Non-linear function is approximated using a non-linear function with 8 basis functions and condition set to 0.

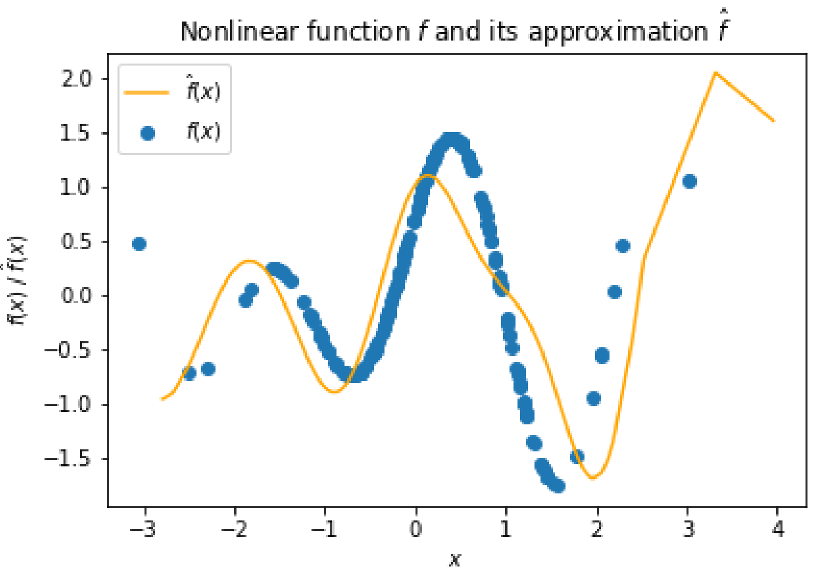

Figure 6: Non-linear function is approximated using a non-linear function with 8 basis functions and condition set to 0.06.

And we get:

$$A = \begin{pmatrix} -0.10967832 & -0.10307182 \\ 0.05153589 & -0.21275016 \end{pmatrix}$$

Part two

In this part we actually solve the linear system up to time $T_{end} = 0.1$ with all $x_0^{(k)}$ as initial points and get an approximation for $x_1^{(k)}$ from $x_0^{(k)}$. We have used the **solve_ivp** method from the **scipy.integrate** module to approximate the end locations. The mean squared error of this approximation to all the known points (N =

1000) is $9.96e - 06$.

Part three

The phase portrait and the trajectory visualization computed with the approximated vector field can be seen in figure 7.

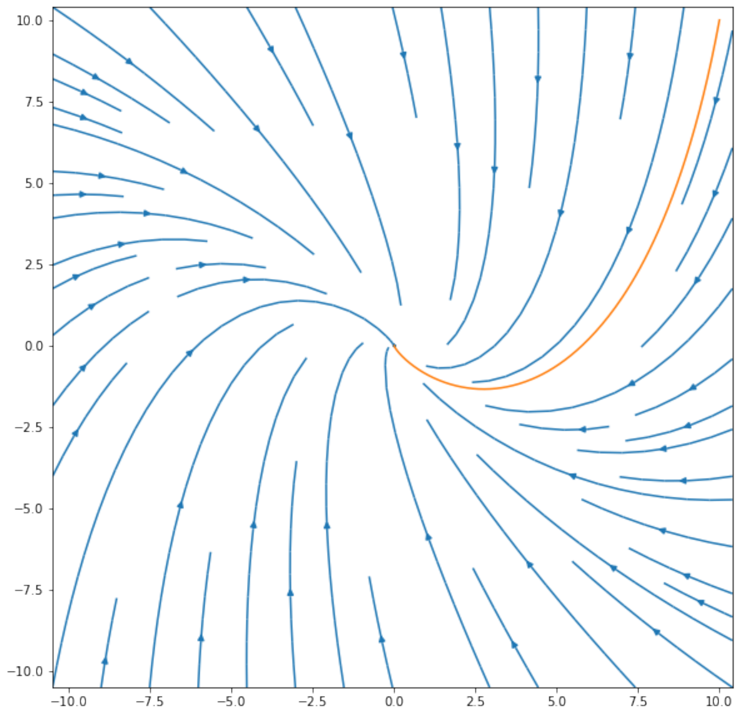

Figure 7: Approximated phase portrait of the linear vector field given in task 2 and the trajectory started at $(10, 10)$. On the x-Axis we have x0 and on the y-Axis we have x1.

Report on task Task 3/5: Approximating nonlinear vector fields

In this task we obtain two **.txt** files, containing 2000 points and as the filenames also suggests ("nonlinear_vectorfield_data_x0" etc.), these data points form a nonlinear vectorfield. The evolution operator and the time-step Δt is unknown to us and we have to approximate them using different techniques. Our main goal is to study the underlying dynamics of this process as in the first tasks.

First part

In this first part we have to do a linear approximation to the given dataset, again obtaining matrix $A \in \mathbb{R}^{2x2}$, such that it describes the evolution operator of the vectorfield ψ. After the approximation of A with least squares minimization, we solved the linear system $\dot{x} = \hat{A}x$, where $\hat{A}$ represent an approximation of A and $\dot{x}$ is the partial time derivative, $\delta x/\delta t$. We have solved it up to a small time Δt, where we have chosen it similar to the last exercise, we analyzed the approximation error when using a specific time step and decided on the time step that causes the least approximation error, which is calculated by mean squared error (MSE). In the end we are left with estimates of end points $\hat{x}_1^{(k)}$, and a MSE error of app. 0.03.

Second / Third part

We could not find a way to approximate the vector field using non-linear radial basis functions since we could not make the plot using the approximated coefficient matrix since the functions are fed into the radial basis functions before the multiplication with this coefficient matrix: $C\phi(x(t))$ where C is the coefficient matrix. Also we got a higher MSE error when we tried to approximate this dataset using radial basis functions but could not find the bug.

Report on task Task 4/5: Time-delay embedding

First part

We first download the dataset `takens_1.txt`, which contains the data matrix $X \in \mathbb{R}^{1000x2}$. We first plot the first coordinate given in the first column of the matrix against the line number (which corresponds to the 'time') in the dataset and get the resulting plot in figure 11.

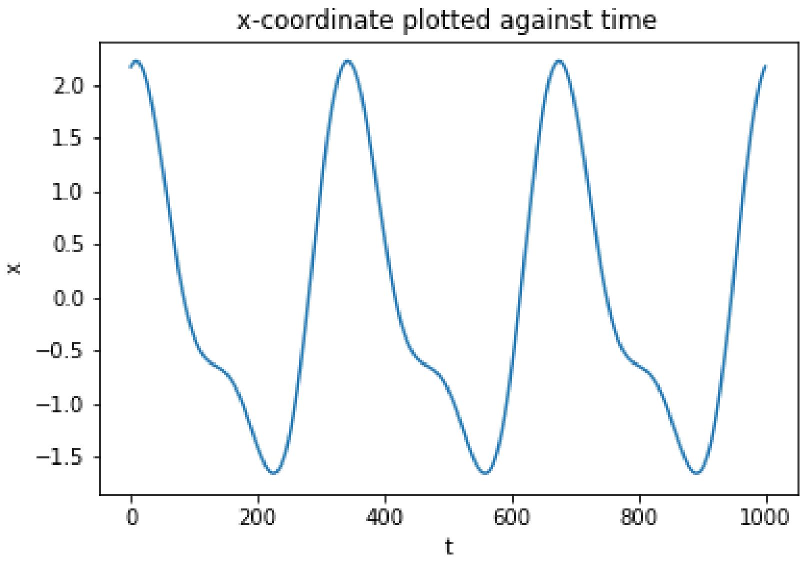

Figure 8: `takens.txt` dataset first coordinate plotted against time.

Now we choose different values for the delay Δn of rows ('time') and plot the delayed versions of the dataset's first coordinate against the actual values in figures 9.

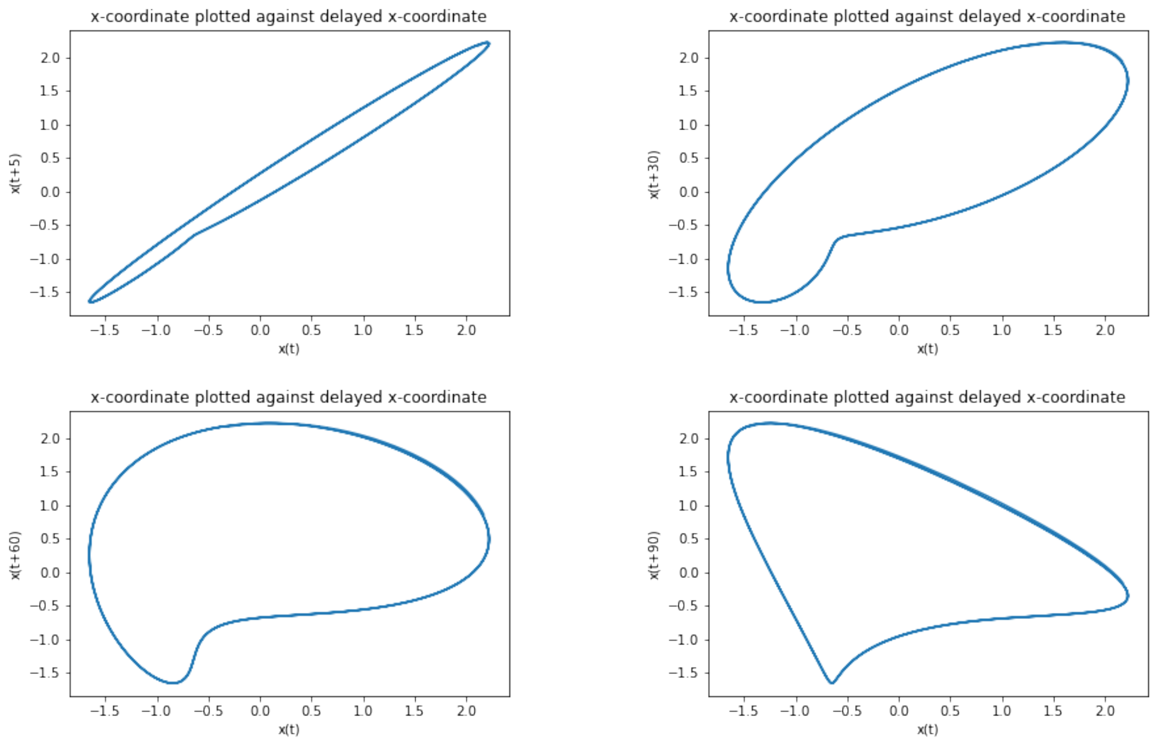

Figure 9: Visualization of the 25-th image of the MNIST training set.

Second part

Part two of this task involves approximating chaotic dynamics from a single time series. We have already plotted the Lorenz attractor in exercise three where we have used the parameters $\sigma = 10; \rho = 28; \beta = 8/3$ to be in the chaotic regime, from starting point (10; 10; 10). In this exercise, we had to test the Takens theorem. We first do it for the coordinate x and visualize $x_1 = x(t)$ against $x_2 = x(t + \delta t)$ and $x_3 = x(t + 2 * \delta t)$ in a three-dimensional plot, with $\delta t = 2.$. In that case we can clearly see that the visualisation of the lorenz attractor is very similar to the non-approximated one. When we try to do the same for the coordinate z, we clearly see that it can not approximate it well. The reason behind that is probably that the variance of the data in the z-coordinate meaning the energy in that dimension is very low and therefore it is not enough to approximate the data with this dimension using time-delay embedding.

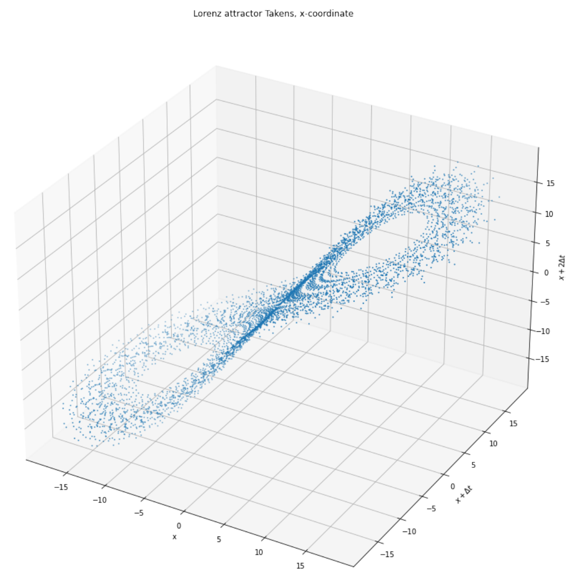

Figure 10: `takens.txt` dataset first coordinate plotted against time.

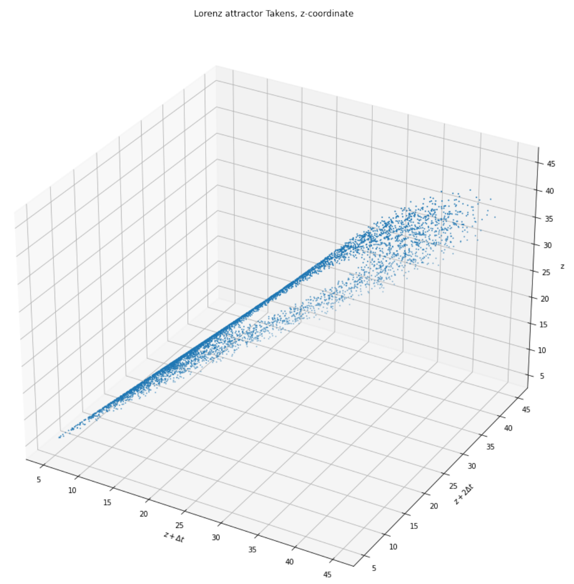

Figure 11: `takens.txt` dataset first coordinate plotted against time.

Report on task Task 5/5: Learning crowd dynamics

Again as always we first download the given dataset `MI_timesteps.txt` which is actually in csv format. We load this dataset using `pandas` module, `read_csv` function. The data has 15001 rows and 10 columns, where the first column specify the timestep and the other 9 columns specify the number of people in specific areas in the TUM Garching Campus [1]. We ignore the first thousand datapoints, so we work with a 14001 by 10 matrix.

Part one

As already mentioned in the previous sections, we could not figure out the Takens Theorem. However we could create a delay embedding with delay 350. We save the delays for each data point into a matrix of shape $(N - \texttt{delay}, (delay + 1) * 3) = (13651, 1053)$. Now we apply PCA on this embedded matrix using 3 top principal components and plot the resulting 'reduced' matrix as in figure 12.

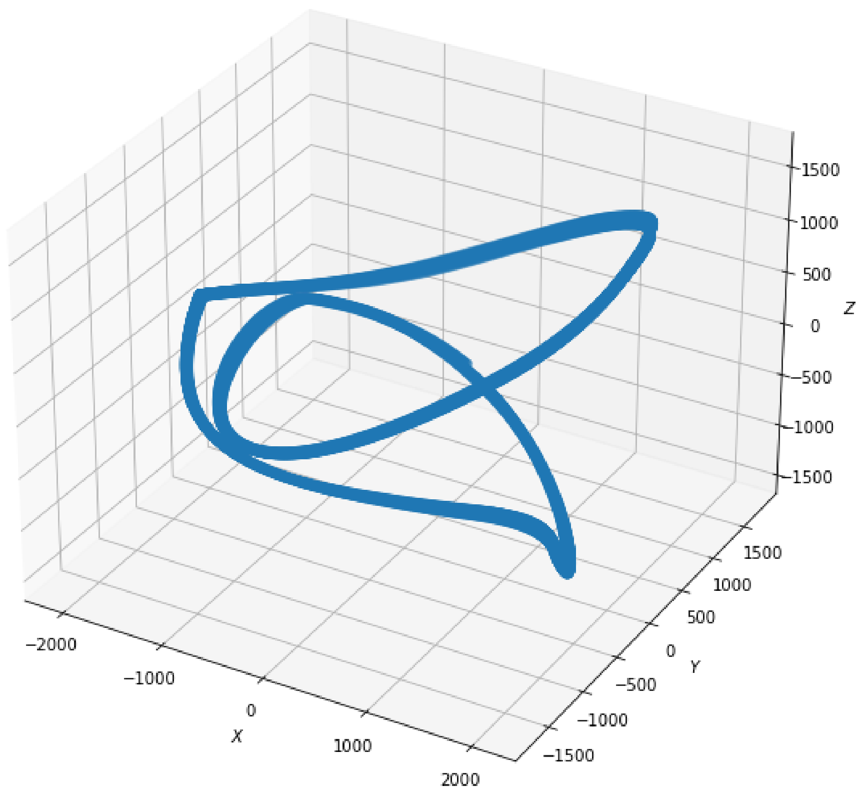

Figure 12: Visualization of the reduced matrix of the embedding matrix created by delaying the actual data and taking the first three measurement columns.

Part two

In this part we have colored all the points in our PCA space by all the measurements that are taken at the first time point of the delays, and we have done this process for all the nine measurement areas. We have created nine different plots for that, where the position of the points in each subplot is same, however the color changes (see Figure 13). To accomplish this, we have followed the explanations on the exercise sheet, where we have passed the column i of our original data starting from the first time point of the delay as the color parameter to the scatter function. We have passed the points in the PCA space as the data to be scattered so that they stay fixed throughout each subplot.

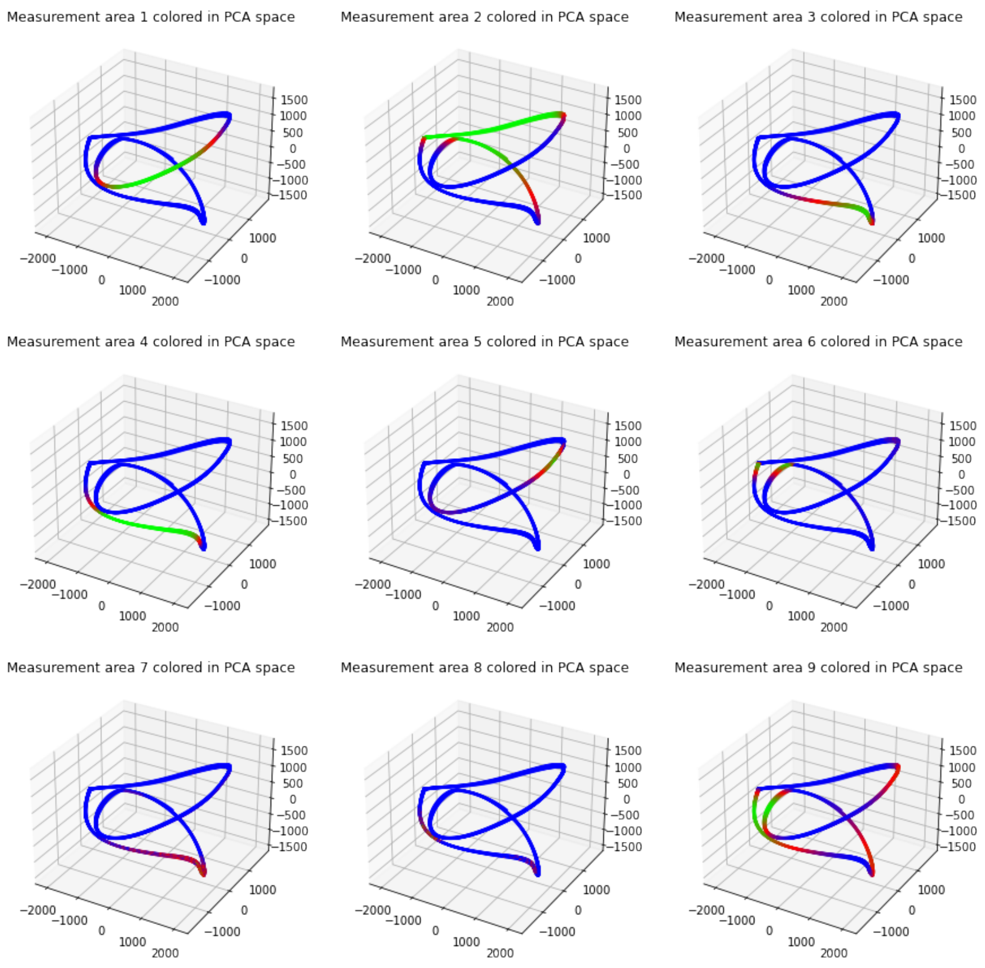

Figure 13: In each subplot a different measurement area is colored in the PCA space.

References

[1] https://tum.sexy/. *TUM Garching Informatik*. July 2022.