Report for exercise 5 from group K

Tasks addressed:	5
Authors:	SONJA KRAFFT (03681252) LUDWIG-FERDINAND STUMPP (03736583) TIMUR WOHLLEBER (03742932)
Last compiled:	2022–01–20
Source code:	https://gitlab.lrz.de/00000000014A9334/mlcs-ex5-extracting-dynamical-systems-from-data

The work on tasks was divided in the following way:

SONJA KRAFFT (03681252)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%
LUDWIG-FERDINAND STUMPP (03736583)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%
TIMUR WOHLLEBER (03742932)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%

1 Introduction

Goal of this exercise

The real world can never be represented holistically, but only through observations quantified by sensor systems. Often it is the case that only a small number of parameters, sometimes even only one, can be observed. To be still able to inference the behaviour of the system, this exercise essentially deals with the topic of learning dynamical systems on the basis of given data records. In the first part, linear and non-linear functions as well as vector fields are to be extracted using the machine learning approach of regression. Furthermore, conclusions are to be drawn from single observations to the entire system behaviour using the time embedding approach. subsequently, the knowledge gained is to be used to analyse the pedestrian utilization of the campus of the Technical University of Munich.

Python dependencies

Due to the great selection of publicly available libraries, Python is chosen as the programming language for this exercise. The following is a listing of required software and Python packages in order to run and maintain the code.

As a container for our datasets, we make use of the `pandas`[1] library and its `pandas.DataFrame` class. For plotting, we decided to use `matplotlib.pyplot`[2] and `seaborn`[3]. `Matplotlib.pyplot` is the standard library to take, especially for 3D plots, while seaborn is very convenient to use together with `pandas.DataFrame`. `SciPy`[4] and `NumPy`[5] are used for the core mathematical operations, where especially `np.linalg.lstsq`[6] is of great value for the implementation of linear regression. Finally we use `scikit-learn`[7], to turn our own implementations of linear and rbf regression into a proper `sklearn.base.BaseEstimator` class that can be used together with their cross-validation and hyperparameter tuning capabilities.

`JupyterLab`[8] is used for prototyping and analysis, `black`[9] for style formatting and `mypy`[10] for static typing. These make up our development dependencies, only needed to run and develop the code, not necessarily dependencies of the internal codebase. The used version number can be found in the `requirements-dev.txt` file inside the root of the repository.

python	3.9.5
vscode	1.63.2

Table 1: Python version and code editor.

matplotlib	3.4.3
pandas	1.2.5
scikit_learn	1.0.2
seaborn	0.11.2
scipy	1.7.1
numpy	1.21.3

Table 2: List of required Python packages to run the code of the repository. Find the corresponding `requirements.txt` file inside the root of the repository.

[1] `https://pandas.pydata.org/`
[2] `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html`
[3] `https://seaborn.pydata.org/`
[4] `https://scipy.org/`
[5] `https://numpy.org/`
[6] `https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html`
[7] `https://scikit-learn.org/stable/`
[8] `https://jupyter.org/`
[9] `https://black.readthedocs.io/en/stable/`
[10] `http://mypy-lang.org/`

2 Individual Tasks

Report on task 1/5: Approximating functions

Notebook: `task_1.ipynb`
Code: `helpers/models.py`, `helpers/data.py`, `helpers/plot.py`

Motivation

In reality, both linear and non-linear relationships arise for observations. In the domain of Machine Learning, we want to be able to approximate both and generalise on unseen data. For this purpose, the following section will briefly explain the approach of ridge regression and regression with the radial basis function. To do so, both approaches will be applied to a linear dataset (A) and non-linear data set (B).

Ridge Regression

The simplest case of parameter correlation is the case where two variables linearly dependent on each other. If this is the case, the individual points of the data set can be approximated by a simple straight line. This learned straight line can now be used to make predictions about new points. The resulting straight line can be expressed by the following equation (absorbed bias term):

$$f_{\mathbf{w}}(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i \tag{1}$$

A common choice to solve this problem is to minimize the least squares loss function, which measures the error between our model (parametrized by w) and observed data stacked as rows into a data matrix $X = (\mathbf{x}_i, \mathbf{y}_i)_{i=1}^{\mathrm{N}}$ with dimension $N \times D$, where N denotes the number of samples and D the number of features. To ensure better generalisation for unseen data, a regularisation parameter λ is often chosen to avoid overfitting to the training data. In matrix notation, the following minimisation problem arises, which is better known in the literature as ridge regression:

$$E_{ridge} = \frac{1}{2}(X\mathbf{w} - \mathbf{y})^T(X\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \tag{2}$$

The optimal weight vector w^* then results from the closed-form solution as:

$$\mathbf{w}^* = (X^T X + \lambda I)^{-1} X^T \mathbf{y} \tag{3}$$

Radial Basis Function Regression

Most relationships in nature are not linear but more complex, which leads to non-linear relationships of the observed parameters. In order to be able to express these unknown functions, the approach has been established to decompose them into L number of known functions.

$$f(\mathbf{x}) = \sum_{l=1}^{L} \mathbf{c}_l \phi_l(\mathbf{x}), \qquad \mathbf{c}_l \in \mathbb{R}^d \tag{4}$$

A well-known and flexible function for the decomposition is the radial basis function as given in Equation (5). In this equation the bandwidth parameter ϵ influences the curvature of the curve section. The point x_l is the respective central point of the dataset used for approximation.

$$\phi_l(\mathbf{x}) = \exp(-||\mathbf{x}_l - \mathbf{x}||^2/\epsilon^2) \tag{5}$$

and allow to calculate the unknown coefficients c_l. For this purpose, the optimization procedure described in the upper section is used again. This results in the coefficient matrix $C \in \mathbb{R}^{d \times L}$ and the design matrix $\Phi(X) \in \mathbb{R}^{N \times L}$

N-Fold Cross Validation

As with most machine learning models, there are a number of hyperparameters to be determined for the models used and that cannot be learned directly during the mathematical optimization. For ridge regression, the choice of the regulation parameter λ arises, which can be interpreted as the weighting factor of the adjustment of the calculated weights. As a second model, the RBF approximation is used as described in the upper section. Here, both the number of basis functions L and the curvature (bandwidth parameter ϵ), must be chosen. For this purpose k-fold cross validation is a suitable and established approach. Here, the entire dataset is divided into k subsets. Then, k-1 are used for training and the remaining one for validation. Thus, the subsets are exchanged k times and then an average is calculated. In order to choose the optimal combination of hyperparameters, this process is repeated many times for different configuration of hyperparameters. Finally, one chooses the configuration with the best cross-validation score.

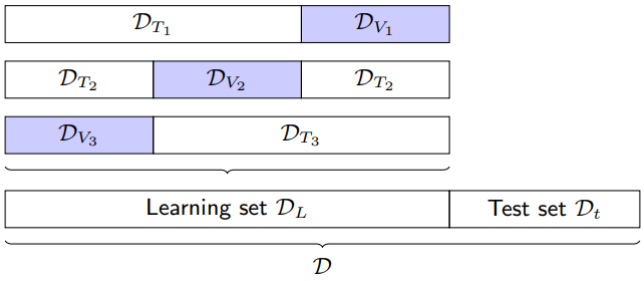


Figure 1: Shows an example illustration of K-fold cross validation with k= 3 folds. In each iteration a different subset is chosen as the validation set, which is highlighted in blue.

We decided to use randomized cross validation search of the python library from sklearn `RandomizedSearchCV`[11]. The following list gives a detailed overview about the hyperparameter optimisation process for all four models and their cross validation results:

- T = 200 search constellations for Figure 2:
  - k = 5 (folds)
  - number of sampled parameter combinations = 40
  - final set of hyperparameters: $\lambda = 1.96 \cdot 10^{-6}$
  - final R2 score: 0.99999999980
- T = 500 search constellations for Figure 3:
  - k = 5 (folds)
  - number of sampled parameter combinations = 100
  - final set of hyperparameters: $\lambda = 9.61 \cdot 10^{-4}$, $\epsilon = 14.4$, $L = 111$
  - final R2 score: 0.999999967
- T = 500 search constellations for Figure 4:
  - k = 5 (folds)
  - number of sampled parameter combinations = 100
  - final set of hyperparameters: $\lambda = 8.19 \cdot 10^{-2}$
  - final R2 score: -0.033
- T = 500 search constellations for Figure 5:
  - k = 5 (folds)
  - number of sampled parameter combinations = 100
  - final set of hyperparameters: $\lambda = 3.32 \cdot 10^{-5}$, $\epsilon = 9.82$, $L = 564$
  - final R2 score: 0.999976

[11] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html`

Part I - Linear Approximation Dataset A

In the first part of the task, the data from dataset (A) are to be approximated using a linear function. The result of the ridge regression model is shown in Figure 2. It turns out that the prediction of the model fits with the respective ground truth label in the entire interval $I \in [-3, 3]$. As expected, a linear model can fit a linear data set up to machine precision. The case of overfitting should not have a bad effect on the generalisation here, as it is to be expected that a linear dataset can be mapped with a linear model and because there is no additional noise. The cross validation led to a regularization factor of $\lambda = 1.96 \cdot 10^{-6}$ but since there is no noise on the line, a value of $\lambda = 0$ would theoretically achieve the best validation score.

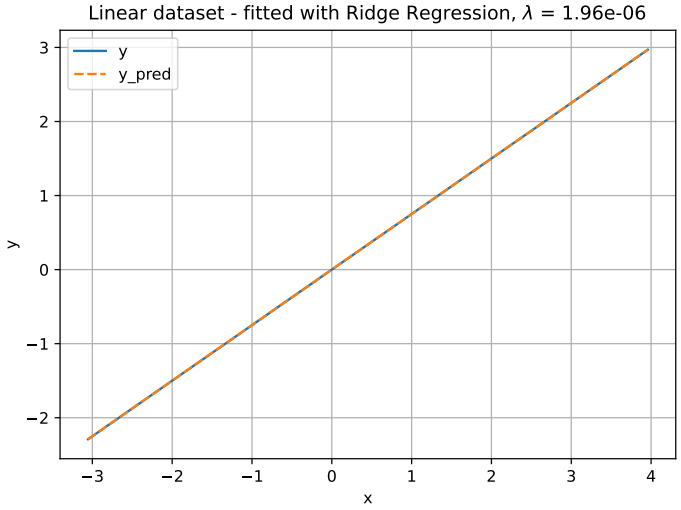


Figure 2: Shows the prediction of the ridge regression model for the feature x over the label y for the linear dataset (A). The prediction of the model is illustrated in orange and the ground truth is highlighted in blue.

Next, the RBF approximation is to be applied to the same linear dataset(A). It can be seen in Figure 3 that this model also manages to approximate a large part of the data. The only noticeable deviation occurs in the interval $I \in [3, 4]$ near the upper input range. This makes sense since the L RBF functions are centered randomly only within the input range, leading to a bad approximation behaviour at both ends. Ultimately, the execution of this experiment shows that it is not a good idea to use this model for modeling linear relations, as in general a large number of basis functions L are necessary and RBF functions are not linear by nature. The cross validation for this dataset and the model showed the best parameters for L = 111 and $\epsilon = 14.4$. It therefore becomes apparent that a disproportionate number of basis functions L are necessary even for the small interval, as well as extremely large bandwidth parameters ϵ. The best regularisation parameter for the experiment was $\lambda = 9.61 \cdot 10^{-4}$.

Part II - Linear Approximation Dataset B

In the second part, the linear ridge regression model is used to represent the non-linear dataset (B). The result of the experiment is shown in Figure 4. It appears that the orange dashed line (prediction of the model) cannot represent the complexity of the given data (blue curve). This is a case of underfitting where our model is not complex enough to model all the complexity of the data.

Part III - RBF Approximation Dataset B

Repeating the experiment with the nonlinear RBF approximation, which was inferior for the linear data set, now shows the strength of this approach. For the entire interval, it is possible to capture the entire complexity of the data using $L = 564$ radial basis functions. In addition to the number, the cross validation also provided the bandwidth parameters $\epsilon = 9.82$ and $\lambda = 3.32 \cdot 10^{-5}$.

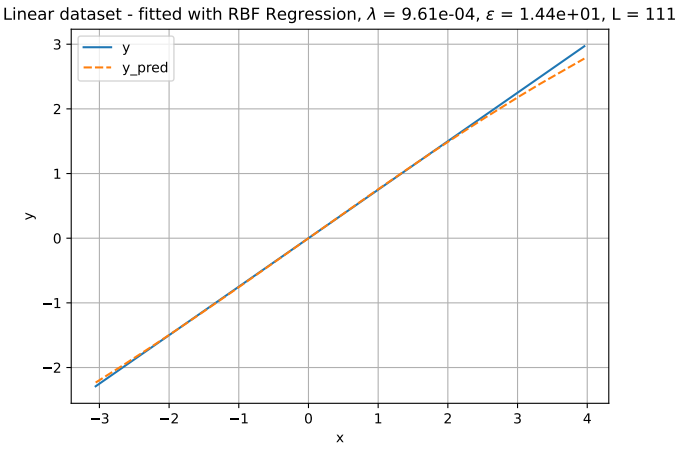


Figure 3: Shows the prediction of the RBF regression model for the feature x over the label y for the linear dataset (A). The prediction of the model is illustrated in orange and the ground truth is highlighted in blue.

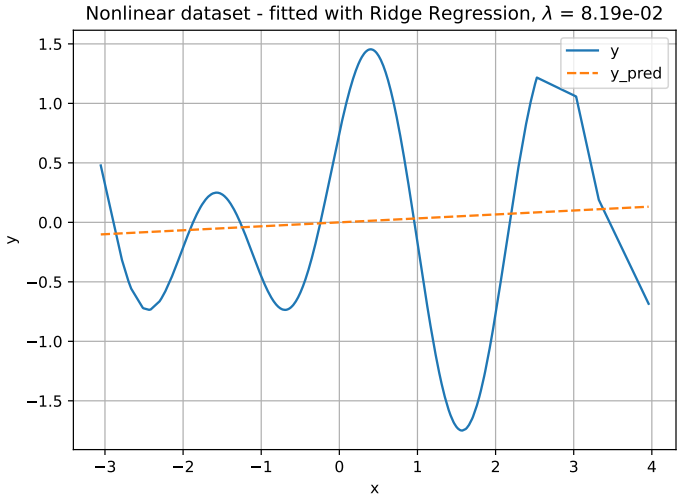


Figure 4: Shows the prediction of the ridge regression model for the feature x over the label y for the non-linear dataset (B). The prediction of the model is illustrated in orange and the ground truth is highlighted in blue.

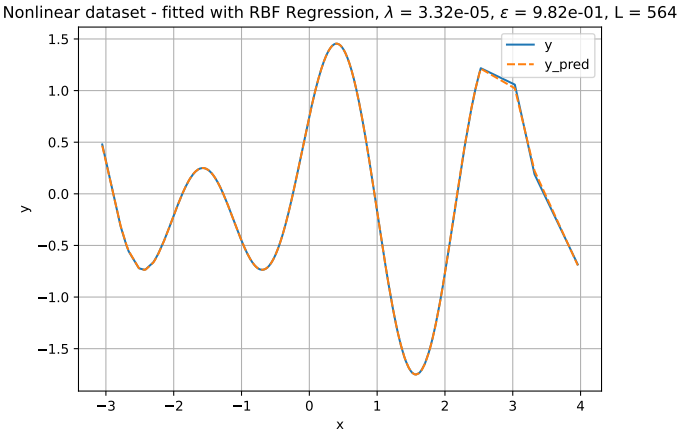


Figure 5: Shows the prediction of the ridge regression model for the feature x over the label y for the non-linear dataset (B). The prediction of the model is illustrated in orange and the ground truth is highlighted in blue.

Report on task 2/5: Approximating linear vector fields

Notebook: `task_2.ipynb`
Code: `helpers/vectorfields.py`, `helpers/data.py`, `helpers/plot.py`

Motivation

When analysing crowd dynamics, one might face the challenge that they want to predict states in the future with differential equations but lack the information about the existing vector field. If only two states at two different points of time $t_0 = 0$ and $t_1 = \Delta t$ are known, one has to approximate the underlying vector field in order to estimate states at different points in time. In this task, the vector field for two given states x_0 and x_1, that each contain 1000 samples of 2 dimensions, is approximated, the resulting linear system is solved and the dynamical system is analysed. The given data is illustrated in Fig. 6.

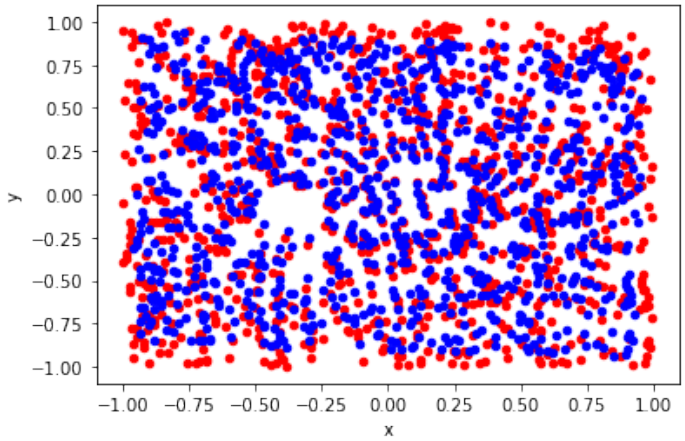


Figure 6: Visualisation of the input states x_0 ($t_0 = 0$, red points) and x_1 ($t_1 = \Delta t$, blue points) with 1000 samples each.

Part 1: Estimate the vector field

$$\hat{v}^{(k)} = \frac{x_1^{(k)} - x_0^{(k)}}{\Delta t} \tag{6}$$

$$\nu(x_0^{(k)}) = \hat{v}^{(k)} = A x_0^{(k)} \tag{7}$$

$$\hat{A}^T = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T v^k \tag{8}$$

The *finite-difference formula* (6) is used to approximate the vector field from the given states x_0 and x_1. $\Delta t = 0.1$ is chosen from the task description. Because the vector field is linear, it can be approximated with (7). This matrix $A \in \mathbb{R}^{2x2}$ is approximated with $\hat{A} \approx A$ by analytically computing (8) with the closed form solution.

Our computations lead to: $\hat{A} \approx \begin{pmatrix} -0.50 & 0.23 \\ -0.46 & -0.95 \end{pmatrix}$

Part 2: Solve the linear system

After analytically computing $\hat{A}^T$, the linear system (9) is solved in the interval $T = \{0, 0.1\}$ for all initial points x_0. This ordinary differential equation is solved by using the library function `scipy.integrate.odeint`.

$$\dot{x} = x\hat{A}^T \tag{9}$$

At $t_1 = \Delta t$, one receives the estimations $\hat{x}_1 \approx x_1$. In order to evaluate the quality of the results, we use the *Mean Squared Error* (10) that is minimal at the given t. We achieve a *Mean Square Error* of approximately $5 * 10^{-6}$ which is sufficiently small such that it can be assumed that the computation successfully approximates the linear vector field for the given data points.

$$e = \frac{1}{N}\sum_{k=1}^{N} ||\hat{x}_1^{(k)} - x_1^{(k)}||^2 \tag{10}$$

Part 3: Analyse the dynamical system

The results of the dynamical system are analysed by visualising the trajectory of the initial point (10,10) and by the phase portrait on a domain $[-10, 10]^2$ (see Fig. 7 and 8).

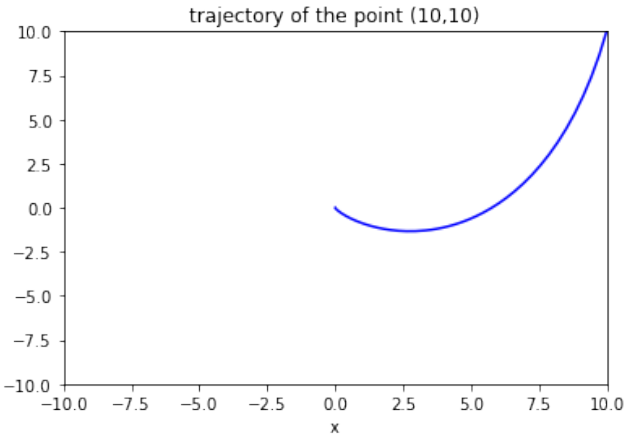


Figure 7: Trajectory of the initial point (10,10) in the time interval $T = \{0, 100\}$ with 1000 values moves to the origin (0,0)

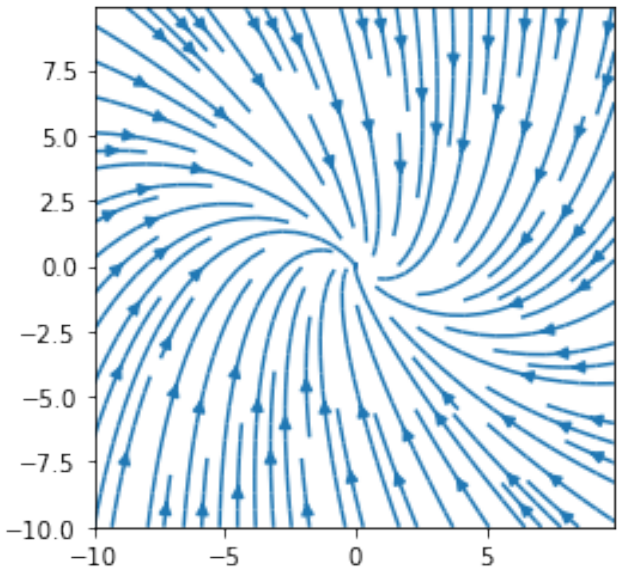


Figure 8: Phase portrait in a domain $[-10, 10]^2$ shows that there is an attracting focus at (0,0)

The trajectory is computed by using the approximated $\hat{A}^T$ in order to solve the ordinary differential equation for the given time interval. The initial point (10,10) moves towards the origin at (0,0) on a curve. The phase portrait is computed by creating a mesh grid on the domain with 100 values for each dimension and then computing the flow by multiplying each value pair on the mesh grid with $\hat{A}$. The results show that the dynamical system can be described with a single attracting focus. From bifurcation theory we know that this behaviour is observable if the eigenvalues of the matrix are both negative and have an imaginary part where the negativity leads to attraction and the imaginary part makes it a focus instead of a node. The fact that both eigenvalues have the same sign leads to a stable steady state. In this case, the eigenvalues are $\{-0.73 + 0.23i, -0.73 - 0.23i\}$.

All points will move towards this equilibrium. This result can also be observed from the visualisation of the input (see Fig. 6). For each red point, its blue counterpart at a later point of time is closer to the origin. So, the computation of the approximated dynamical system corresponds to the expectations.

Report on task 3/5: Approximating nonlinear vector fields

Notebook: `task_3.ipynb`
Code: `helpers/vectorfields.py`, `helpers/data.py`, `helpers/models.py`

Motivation

If the underlying vector field is not linear anymore, but nonlinear, it can always be decomposed into nonlinear basis functions. The goal of this task is to find the underlying structure of the vector field corresponding to two given states x_0 and x_1 and decide which method leads to a better approximation. With this estimation, the dynamical system is analysed in terms of steady states and topological equivalence.

The given states are illustrated in Fig. 9. Each dataset contains 2000 samples with 2 dimensions. The first state x_0 has evolved to x_1 with an unknown evolution operator $\psi : T \times \mathbb{R}^2 \mapsto \mathbb{R}^2$ that has to be estimated by a linear operator in Part 1 and radial basis functions in Part 2.

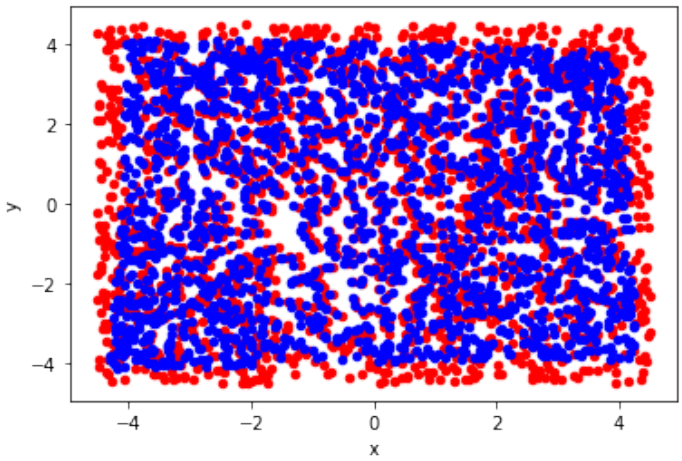


Figure 9: Visualisation of the input states x_0 ($t_0 = 0$, red points) and x_1 ($t_1 = \Delta t$, blue points) with 2000 samples each.

Part 1: Approximation with a linear operator

In the first part, we try to estimate the vector field by assuming it is a linear system. Similar to Task 2, a linear operator $\hat{A} \approx A$ is computed with equations (6), (7), and (8) with the inputs x_0 and x_1. Because Δt can be chosen arbitrarily for our combination of analytic computation of $\hat{A}$ and solving the differential equation with `scipy.integrate.odeint`, it is set to $\Delta t = 0.1$ like in Task 2.

Our computations lead to: $\hat{A} \approx \begin{pmatrix} -0.100 & -0.002 \\ 0.008 & -0.432 \end{pmatrix}$

Again, the ordinary differential equation (9) is solved with `scipy.integrate.odeint` for the time interval $T = \{0, 100\}$ and the initial points x_0. The resulting *Mean Squared Error* (10) is approximately 0.0186.

Part 2: Approximation with radial basis functions

Because the *Mean Squared Error* is significantly worse than in Task 2, it can be assumed that the system is not linear and cannot be estimated with a linear operator matrix $\hat{A}$. Therefore, we try to find a nonlinear approximation with radial basis functions.

First, the task has to be prepared such that it is a supervised learning problem. Thus, equation (6) with Δt is used to define the targets with x_0 as inputs. The implemented class `RBFRegression` from Task 1 is used to learn the radial basis functions. The differential equation (11) is solved with `scipy.integrate.odeint`. But instead of letting it compute the product Ax, it now calls the predict function `RBFRegression.predict(x)` for the integration.

$$\dot{x} = C\phi(x) \tag{11}$$

The best values for L and ϵ are chosen by using a randomised search with cross-validation with the implemented function `get_best_model`. The intervals, for which the parameters are evaluated, pick $L \in \{900, 1000\}$ and $\epsilon \in \{10^{-2}, 10^{2}\}$. During the process of hyperparameter tuning the complete intervals for L and ϵ were $\{100, 1000\}$ and $\{10^{-6}, 10^{2}\}$. In case of L, the results were slightly better for greater values, that is why the interval was reduced to $\{900, 1000\}$. For ϵ, the randomised search determined that it mostly lies in the interval $\{10^{-2}, 10^{2}\}$.

However the results have not been convincing, because the `RBFRegression` model only led to a *Mean Squared Error* of 0.0179. That is why, the dimensions of the input data x_0 and the finite difference $v^{(k)}$ have been trained separately with two models. Therefore, the differential equation also is solved for each dimension and a *Mean Squared Error* of 0.004 is achieved. For the two models, best results are achieved with $L_1 = 990$ and $\epsilon = 2.5$ for the first dimension and $L_2 = 919$ and $\epsilon = 6.2$

Because the evolution operator is no longer calculated analytically, the value for Δt is determined by minimizing over all results in the interval $\{0, 1\}$ with 100 steps. The best estimation $\hat{x}_1$ for x_1 is achieved for $\Delta t = 0.11$.

Due to the smaller values for the error, it can be concluded that the evolution operator and the underlying vector field are not linear. The system has a better approximation for a nonlinear estimation than for a linear one.

Furthermore, if one compares the input figures (see Fig. 6 and 9), it is already obvious that for the given input of Task 3 all points do not move towards a single point like they do for the input of Task 2. In the last part of this task, the assumption that there are multiple steady states is verified.

Part 3: Analyse the dynamical system

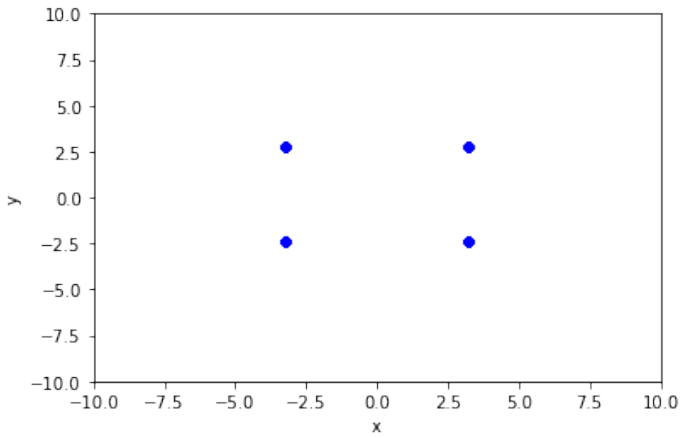


Figure 10: Attracting Equilibria visualised at $t = 100$: Each point of the initial points x_0 approaches one of the four points (-3.2, -2.4), (-3.2, 2.7), (3.2, -2.4), and (3.2, 2.7)

The trained model with radial basis functions leads to better results than the linear operator. That is why, the differential equation is solved over a longer period of time in order to determine towards which states the initial points of x_0 converge. The time interval is chosen as $T = \{0, 100\}$ with 10 time steps. For a higher number of steps, the trajectories are smoother but the final states remain the same. Fig. 10 illustrates that each point of x_0 approaches one of four different points (-3.2, -2.4), (-3.2, 2.7), (3.2, -2.4), and (3.2, 2.7).

These are the only attracting steady states of the system that are observable with the given data and approximation of the vector field. But this fact suffices that the system cannot be topologically equivalent to a linear system that only has one steady state.

Additionally, there has to be another, but repulsive, equilibrium in the area of between the points (-0.1, 0.1), (-0.1,-0.5), (0.1,-0.5) and (0,0) because their trajectories all move towards the four attracting equilibria (see Fig. 11).

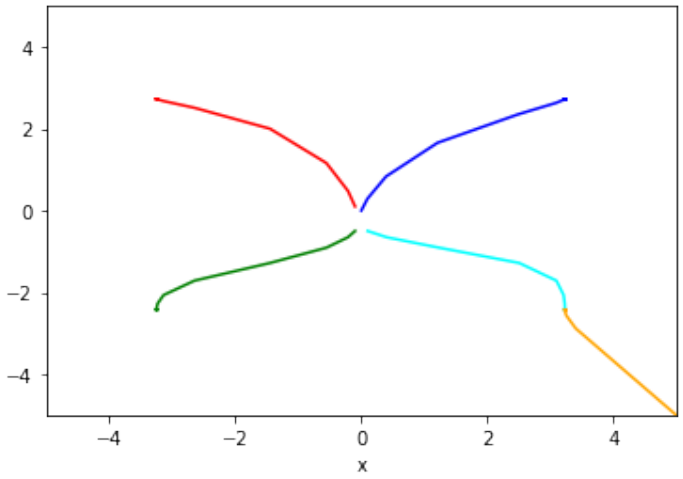


Figure 11: Trajectories of the initial points (-0.1, 0.1), (-0.1,-0.5), (0.1,-0.5), (0,0), and (5,-5) for the time interval $T = \{0, 100\}$ with 100 time steps: the trajectories show the previously identified four attracting equilibria and indicate an additional repulsive equilibrium in the area between these points

Even though, the results for the nonlinear approximation is better than the linear one, the approximation of the vector field could be improved. Exemplary, a higher value for L improves the approximation but the resulting model does not generalise as good. Furthermore, the implementation of the radial basis function regression might be better, such that it can easily handle multidimensional inputs.

Report on task 4/5: Time-delay embedding

Notebook: `task_4.ipynb`
Code: `helpers/embeddings.py`, `helpers/data.py`, `helpers/plots.py`

Motivation

As already described in the introduction of this exercise, it is often the case in real life that only one or a few dimensions of the entire statespace can be measured at once. In the context of this task, the minimum number of dimensions necessary to correctly embed the periodic manifold should therefore be determined using Takens' theorem. In addition to this, it should also be shown using the Lorenz attractor that often fewer dimensions than the limit discussed by Takens are already sufficient.

Part I - State Space Inference - Periodic Signal:

In the first part of the task, the Takens dataset is to be examined more closely. It contains 1000 data series with two columns, which represent the coordinates of a one-dimensional manifold in euclic space. If the first column is viewed over time, which can be interpreted as the number of rows, it becomes apparent that the data seems to yield in a periodic signal. The resulting graph is shown in Figure 12.

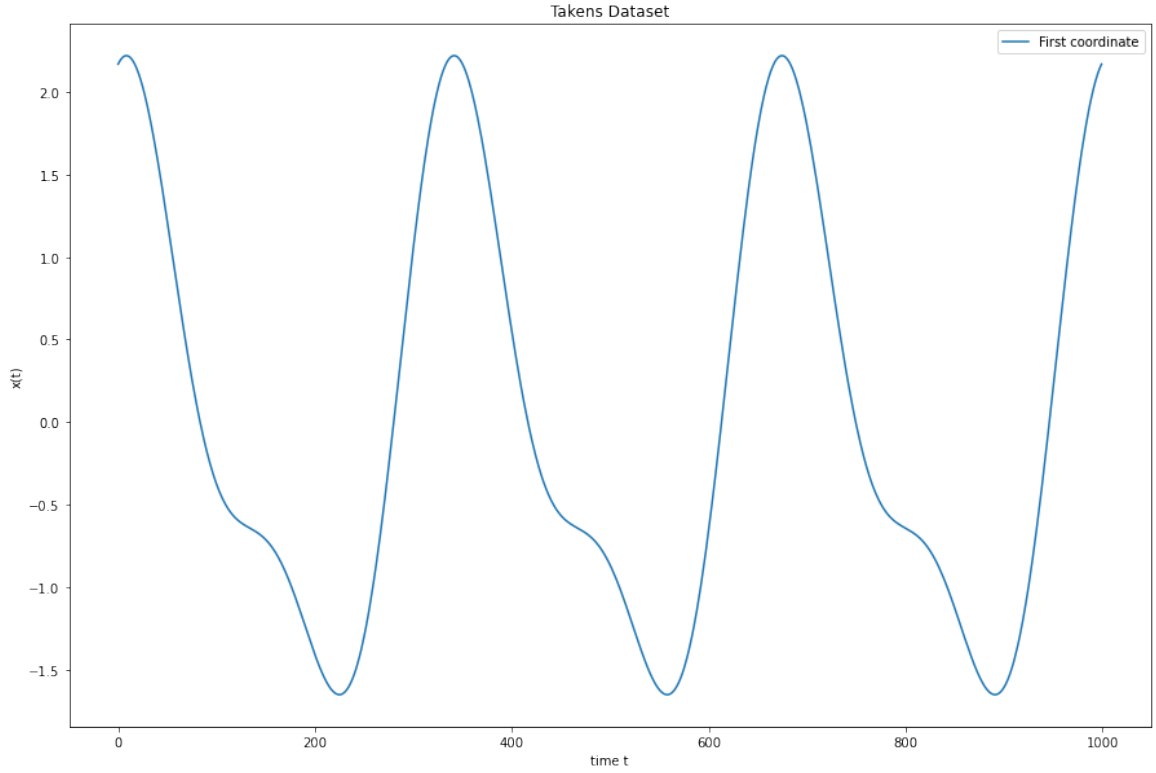
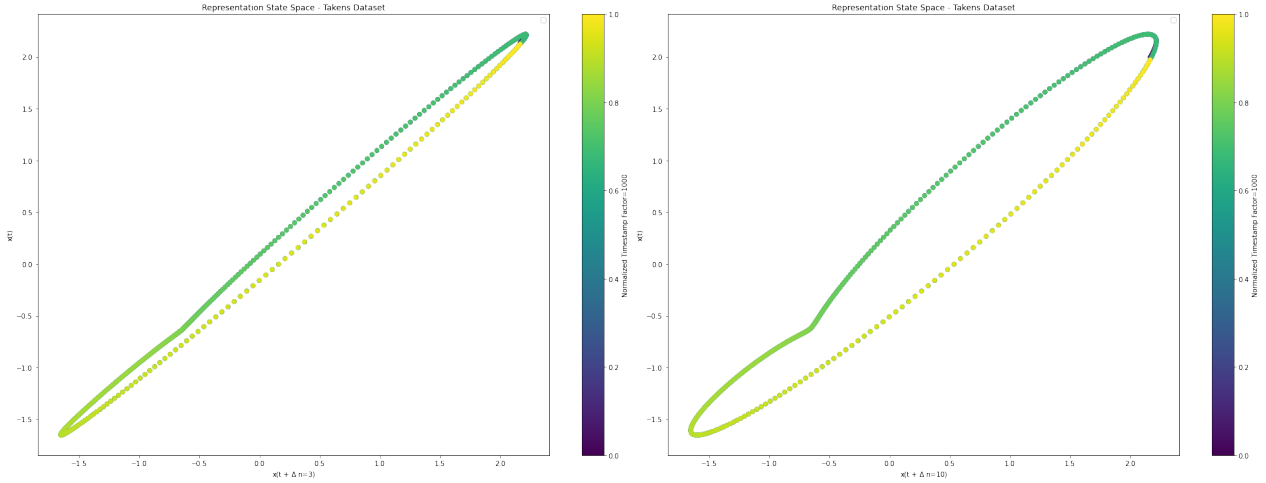


Figure 12: Shows three periods of the periodic signal from the Takens dataset.

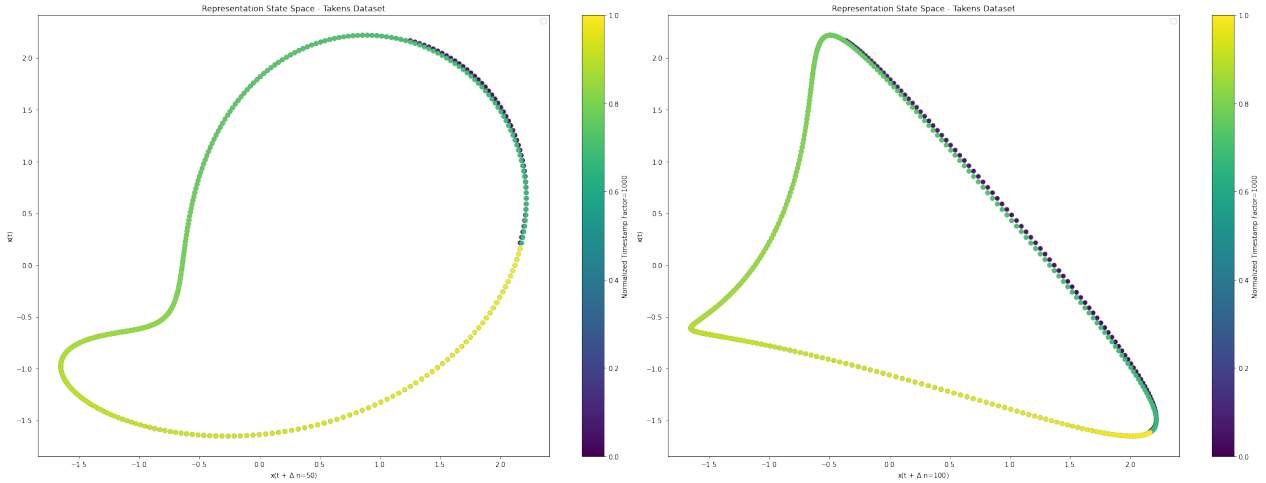
It can be deduced that it is a non-linear process, which has a period of about $t = 350$ (unexact value read from the graph). Over the period of $t \in [0, 1000]$ this results in three periods. The interesting thing about the periodic signal is that it seems to be a modified version of a cosine signal, since at the beginning it already has a value of $x(t = 0) = 2.25$. The modification can be seen in the lower valley of the graph, since for an interval of about $t = 50$ it seems to have a saddle point-like plateau, which then drops to the lower reversal point.

According to the concept for scalar-value observations, it is not possible to recognise which direction of movement appears to be present in the actual state space based on the measured periodic signal. Nevertheless, making assumptions about the direction of movement, the signal of the measured individual variable can be plotted over the same variable with a time delay according to Takens. In order to be able to implement this time lag, a time lag parameter Δn is defined, which delays the signal x(t) with x(t + Δn). To determine the time lag parameter mathematically, one could determine the rank of the Jacobi matrix of the embedding function, but since this would clearly go beyond the scope of this sub-task, configurations for four different parameters are shown in Figure 13 below:



(a) Time lag Δn=3

(b) Time lag Δn=10



(c) Time lag Δn=50

(d) Time lag Δn=100

Figure 13: Shows four configurations of the first coordinate from the Takens dataset over its delayed version. The colour scale encodes the time normalised between zero and one, where one corresponds to the time $t = 1000$.

Please note: it is unclearly formulated in the task whether Δn is to be implemented as lagging or leading, which is why both configurations can be called up in the notebook via the `delay mode` flag. However, initial results here have only given the impression that the same graph is only mirrored, so this should not have any significant effect on the course of the results.

According to Takens' theorem, a correct embedding is achieved by at least 2D+1 coordinates. For the case of the present one-dimensional manifold in euclidean space, a minimum requirement of $2 \cdot 1 + 1 = 3$ coordinates results.

Part II - Lorenz Attractor - Time Delay Embedding:

In the second part of the task, the Lorenz attractor from the third excercise should be used to show that in practice, often less embedding dimensions are already sufficient. If one would apply Takens' theorem, one could derive $2 \cdot 3 + 1 = 7$ necessary dimensions for the three dimensional Lorenz system. In practice, however, it turns out that if one has calculated $x(t)$, one embeds the coordinates $y(t)$ over $x(t + \Delta t)$ and $z(t)$ over $x(t + 2\Delta t)$, which again results in a chaotic attractor. Such an example configuration with time lag parameter $\Delta t = 10$ is shown in Figure 14.

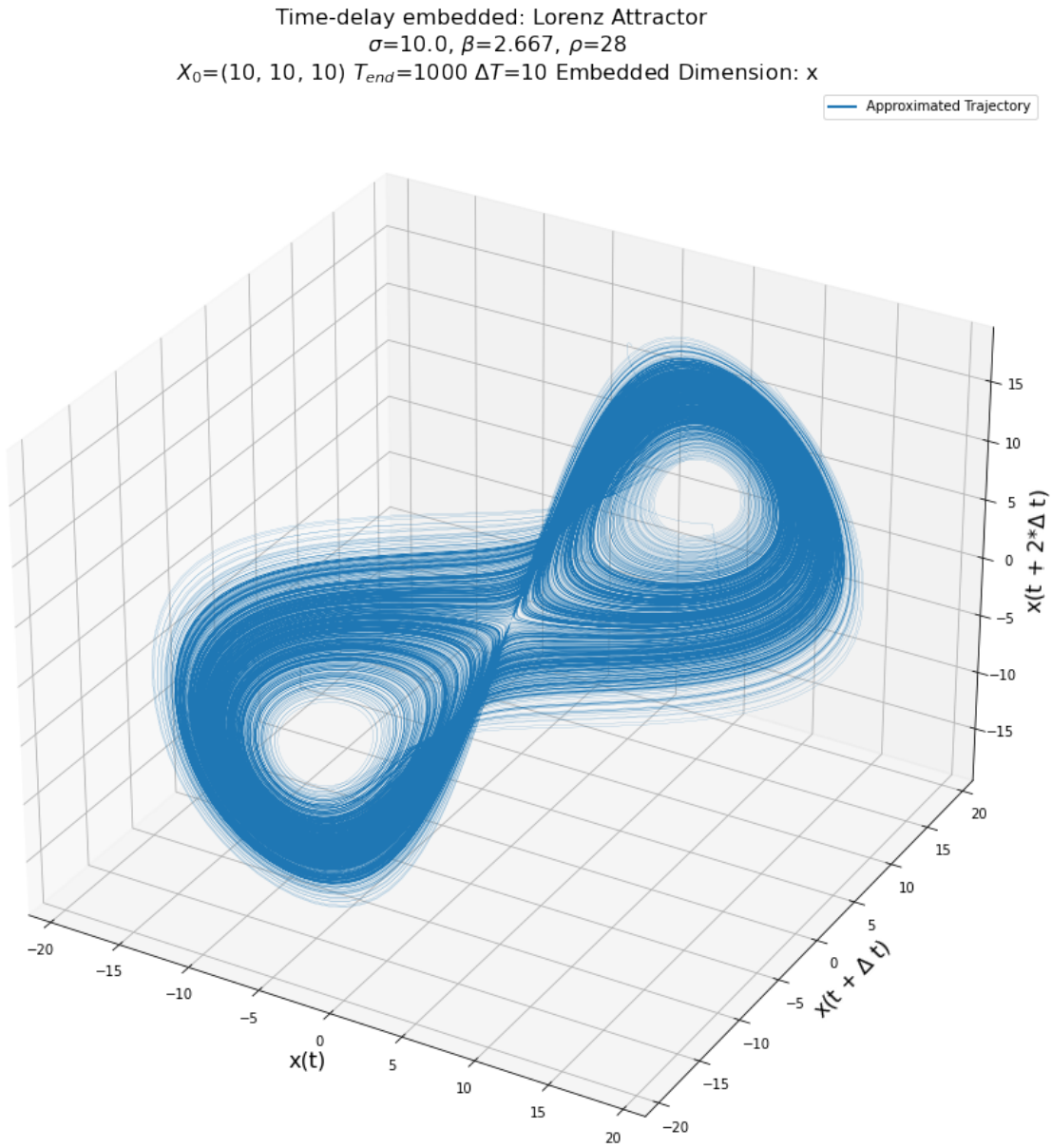


Figure 14: Shows the approximation of the Lorenz attractor for the chaotic parameter set $\sigma = 10, \beta = \frac{8}{3}, \rho = 28$ using time-delay embedding on the calculated x-coordinate. The approximated trajectory is marked in blue and winds from the starting point in $x_0 = (0, 0, 0)$ in an irregular course around the two limit cycles.

Comparing Figure 14 with the actual attractor shown in Figure 15 it becomes clear that the position of the trajectory in space is nevertheless different from the original. This becomes especially apparent by comparing the positioning of the limit cycles in the space and also because the resulting lying eight seems to be stretched along an imaginary diagonal from the lower left to the upper right compared to the ordinary Lorenz attractor. Nevertheless, it is astonishing that already the temporal course of a single coordinate is sufficient to establish an approximation of the attractor.

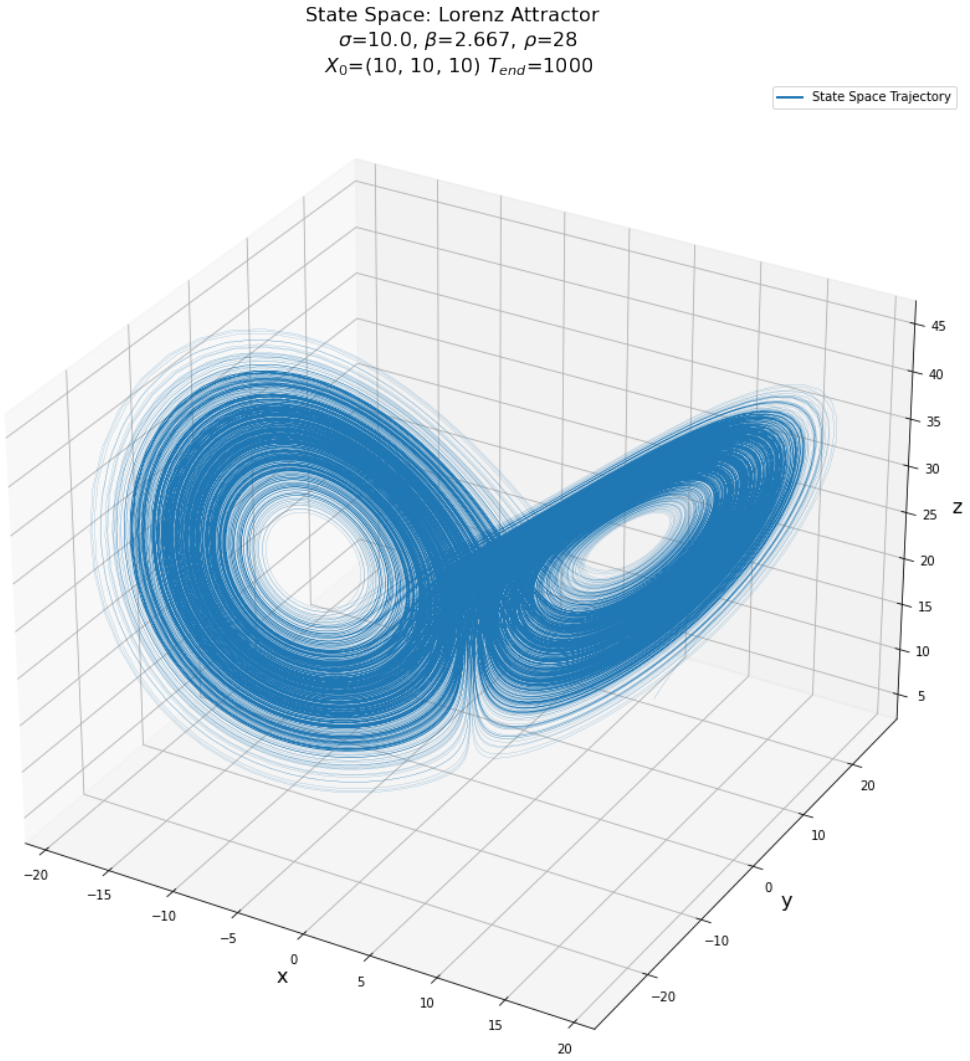


Figure 15: Shows the state space of the Lorenz attractor for the chaotic parameter set $\sigma = 10, \beta = \frac{8}{3}, \rho = 28$. The state space trajectory is marked in blue and winds from the starting point in $x_0 = (0, 0, 0)$ in an irregular course around the two limit cycles.

However, that not every coordinate is suitable for this approach is shown in Figure 16, where the time delay embedding is repeated using the z coordinate instead of the x coordinate for embedding. Instead of the expected two limit cycles around which the trajectory winds, only one cycle remains. Therefore it can be deduced that the dynamic system yields a fundamentally different behaviour. The chaotic parameter set therefore no longer behaves in a truly chaotic way, since the trajectory is no longer attracted between the two poles.

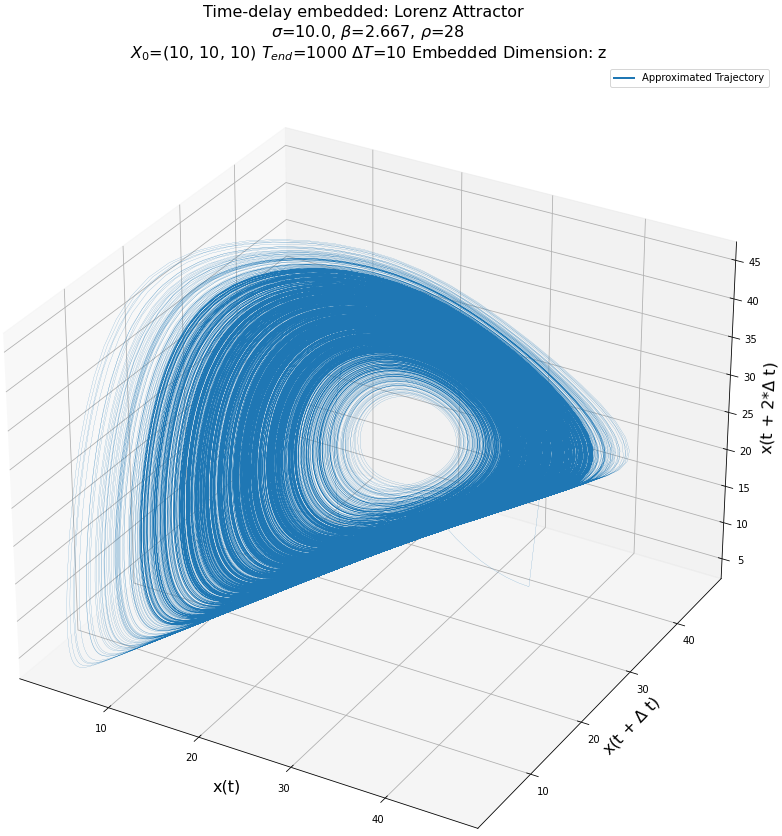


Figure 16: Shows the approximation of the Lorenz attractor for the chaotic parameter set $\sigma = 10, \beta = \frac{8}{3}, \rho = 28$ using time-delay embedding on the calculated z-coordinate. The approximated trajectory is marked in blue and winds from the starting point in $x_0 = (0, 0, 0)$ in an irregular course around one limit cycle.

A more detailed examination of the Lorenz system in Equation (12) reveals that the failure is due to the symmetry only for the x and y dimension of the Lorenz system. The solutions $x(t), y(t), z(t)$ produce the same result as for the mirrored x and y coordinates $-x(t), -y(t), z(t)$. To show that one can see that both solutions describe the same system.

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z \end{aligned} \tag{12}$$

By setting $x \to -x$ and $y \to -y$ and if one would transfer the minus one from the left side to the other side, one can clearly see that in Equation (13) the same system results as in Equation (12):

$$\begin{aligned} -\dot{x} &= \sigma((-y) - (-x)) = \sigma(-y + x) \\ -\dot{y} &= -x(\rho - z) - (-y) = x(-\rho + z) + y \\ \dot{z} &= (-x)(-y) - \beta z = xy - \beta z \end{aligned} \tag{13}$$

This attempt does not succeed for the z coordinate, as different systems would emerge here.

Report on task 5/5: Learning crowd dynamics

Notebook: `task_5.ipynb`
Code: `helpers/data.py`, `helpers/models.py`, `helpers/plots.py`, `helpers/utils.py`

Motivation

In the following section, we analyze a dataset about the campus utilization at TUM and show that a small subset of measurement-dimensions often suffices to capture the intrinsic dynamics of an entire d-dimensional system that can then be used to make predictions for the future.

Introduction to the campus dataset

The dataset `MI_timesteps.txt` contains space utilization values for the TUM campus at Garching, taken from nine different measurement areas over the course of seven weekdays.

Shape of the data

When reading the data, the first 1000 rows are considered a burn-in-period and ignored, which then results in a total of 14001 usable timesteps where there are nine measurements each. This makes a shape of `(14001, 9)` where the first axis is the time-dimension and the second axis is the measurement area.

Visualising the data

Plotting the measurements over time, one can clearly see the periodic nature of the data. The plots gives rise to the assumption that one day is equal to 2000 time steps. Figure 17 shows the full dataset as well as the change in campus utilization over the first day only.

Designing a state space of the system

Since the dataset is rather high dimensional with a total of 9 dimensions. it is useful to think of a compact way to represent the state of the system without losing much information about its intrinsic dynamics and structure. Not only does this help for visualization purposes, but furthermore simplifies the task of fitting any kind of model to the data in order to generalise well.

Assumptions on the system

Due to the results shown in figure 17, we assume that the 9-dimensional data is periodic over each day (= 2000 time steps). One can think of a 9 dimensional ring where each measurement point (= row in the dataset) is equal to one single point on the ring in a 9 dimensional euclidean space. This assumes that the datapoint during timestep 1 equals the position at timestep $\mathrm{k} \times 2000; \mathrm{k} \in \mathbb{N}$. Furthermore, we assume that there are no parametric dependencies in the data, meaning that every pedestrian behaves equally. Both assumptions together mean that the original state space is considered to be a one-dimensional closed loop, embedded in a nine-dimensional euclidean space.

Applying Takens Theorem

According to Takens [1], one requires a maximum of $2d + 1 = 3$ dimensions in order to fully describe an intrinsically $d = 1$ dimensional system. Therefore our final state representation should at least contain 3 dimensions, which represent three new columns in our `pd.DataFrame` object.

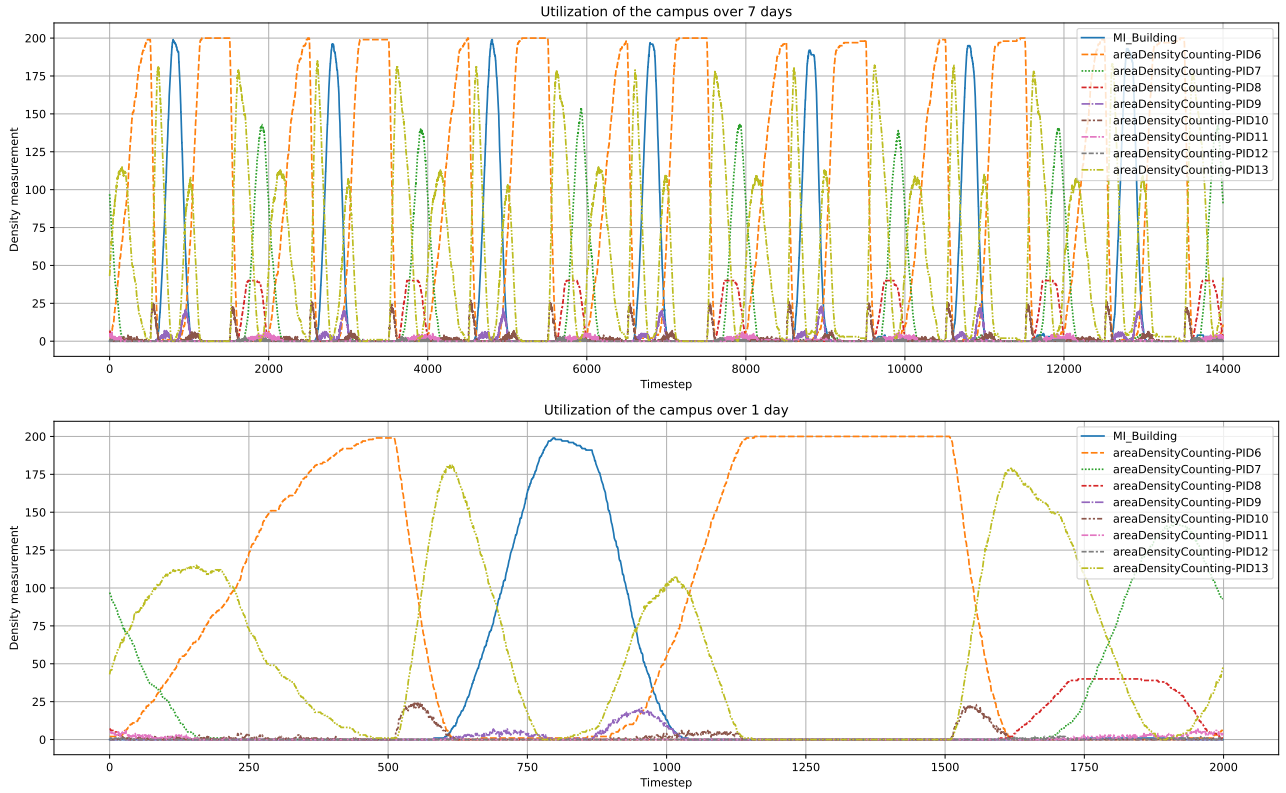


Figure 17: Visualization of the campus dataset, showing pedestrian utilization values accross nine different measurement areas. Shown is the full dataset (top) as well as only the first day (bottom).

Creating a delay embedding of the first three columns

Although directly taking only the first three columns from the orignal data would in theory suffice to fulfil Takens theorem, we want to be a little bit more careful and choose a different approach:

1. Create 1053-dimensional delay embeddings and stack them as rows of an intermediate state representation
2. Apply PCA to only retain the first three principle components

In order to create the delay embeddings, we take a window of (351, 3) dimensions and top-align it over the first three columns of our original data to obtain a 1053-dimensional vector $\in \mathbb{R}^{1053}$ of data which represents one new measurement. We then shift the window one step forward in time (= down in the data matrix) to repeat the process for a total of 13000 times to finally get an intermediate state representation of (13000, 1053).

Apply PCA to get final state representation

Taking Takens theorem into account, we finally apply PCA to reduce the final state representation to three dimensions which leads us to a final shape of (13000, 3) of the internal state of the campus system. Applying PCA and only taking the top three principle components explains 82.8 % of the total variance of the intermediate data we get from the delay embeddings.

Visualising the new state space

Figure 18 shows the just created state colored by the index of the points. This representation helps to indicate the periodic nature of the data.

Next in figure 19 we again show the transformed data points, but now colored by the values of the original measurements. One can see that spikes on the state space in each measurement region (= column in the original dataset) corresponds to spikes in the original data as shown in figure 17. This also indicates that the neighborhood of points seems to be preserved in the new compact state representation.

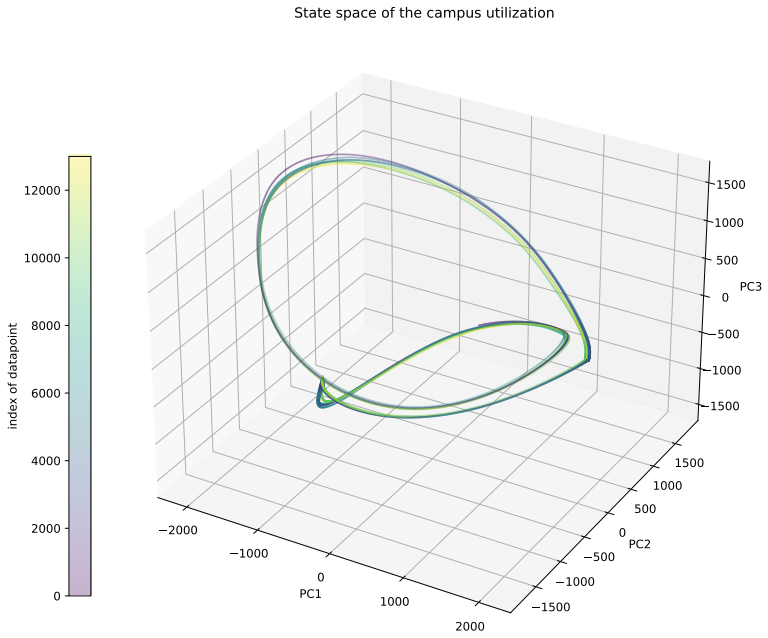


Figure 18: Visualization of the internal state space of the campus dataset. Points are colored based on their index showing the periodic nature of the data.

Analyzing the systems dynamics in the crafted subspace

In order to understand the campus dataset, we analyze the dynamics of the state representation by capturing how fast the state traverses on the space. In order to do so, we calculate the arc length of the trajectory over time and calculate its derivative.

The arc length is estimated numerically following the points on the 3-dimensional circle and adding up their L2-norms while the derivative in time is then calculated using central differences.

Figures 20 and 21 show the results of the analysis specially pointing towards the periodic nature of the state space.

Predicting the utilization of the MI building for the next 14 days

Next, we want to use the compact state space representation of the campus data to draw conclusions back to the original data. In particular, we want to predict the utilization for the MI building over the next 14 days. This corresponds to the first column in the original `(14001, 9)` shaped data, where 14 days equals a total of 28000 timesteps. Note that in our given data, we only have a total of 14001 timesteps of utilization values of which we only have a corresponding state representation for the first 13000.

This prediction task is achieved in the following steps:

1. learn a mapping from the state space back to the original data.
2. forecast how the state space will look like over the next 14 days.
3. transform this forecasted state space representation into the original data by making use of the learnt mapping from state space back to original data.

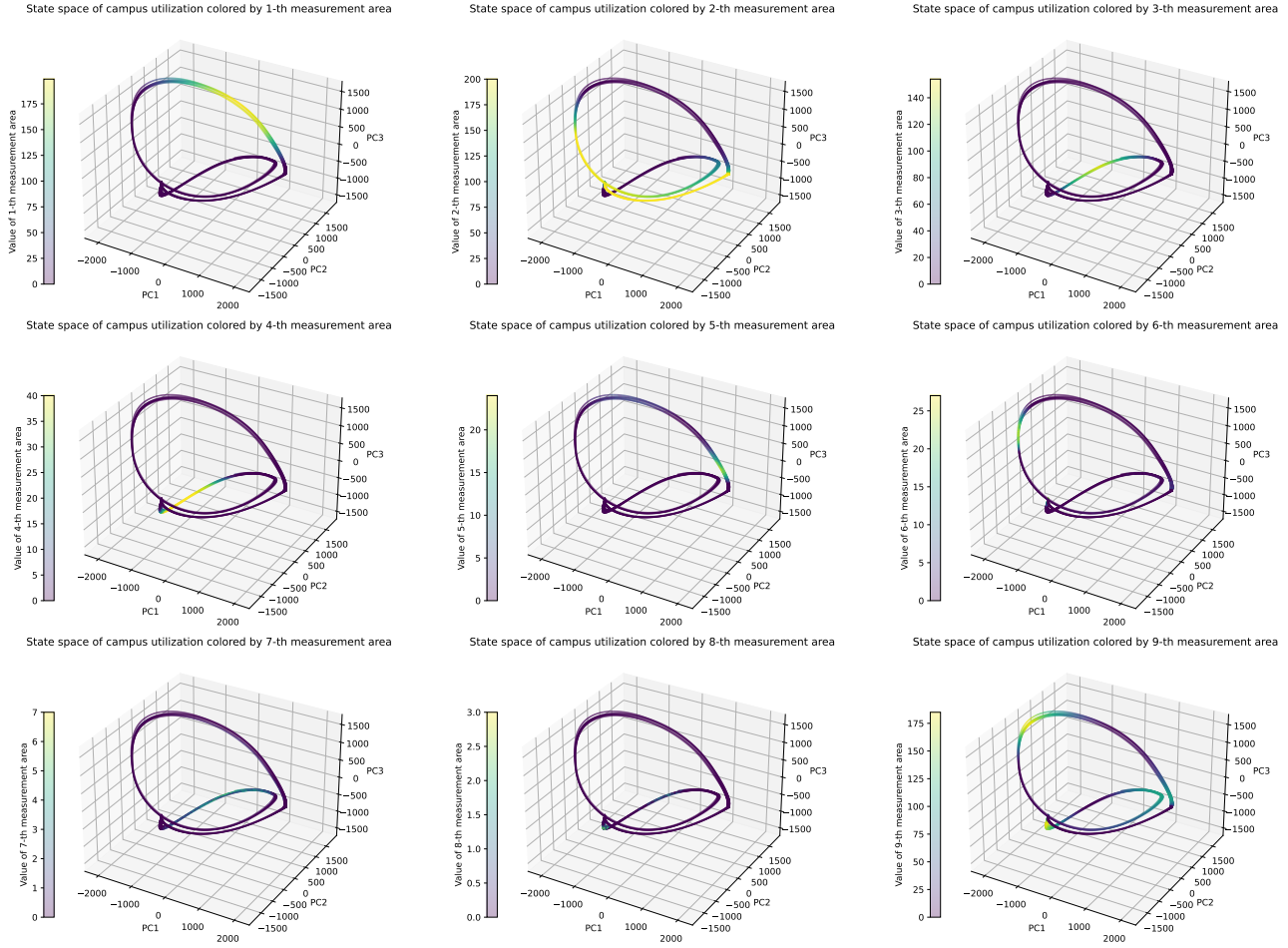


Figure 19: Visualization of the internal state space of the campus dataset. Points are colored based on the original values of the different measurement areas.

Learning a mapping from the state space back to the original data

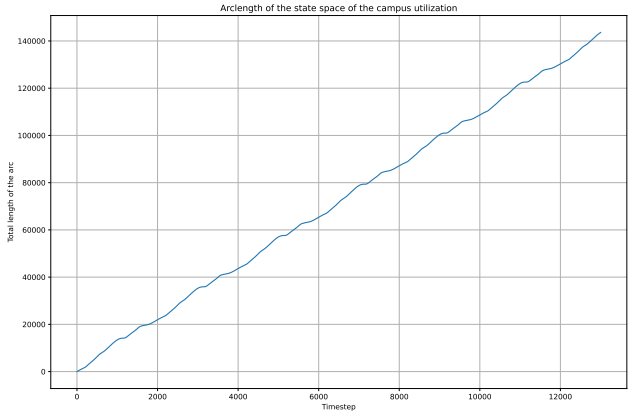
In order to learn the nonlinear mapping from the state space to the utilization values of the MI building we are making use of our RBF regression implementation from task 1. Since both the arclength in our state space and the corresponding target utilization values are known for all M = 13000 timesteps, this is a classical supervised problem.

In order to validate the generalisation performance of our fitted RBF model, we:

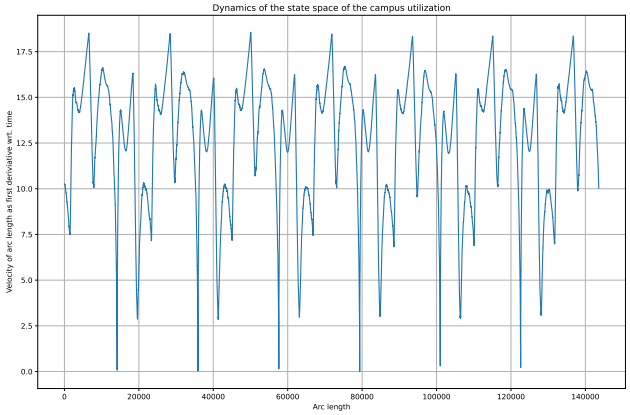
1. split the data into a `train` and `val` split.
2. fit the model on `train`.
3. check its performance on unseen data `val`.
4. adapt parameter settings L (number of randomly placed RBF functions), ϵ (bandwith of all RBF functions) as well as λ (regularization strength) in order to maximise for the validation score.

During this experimental search process, the ranges for ϵ have been chosen to be in the range of the spike in utilization (see figure 22) and L around 1000. Table 3 shows the final set of hyperparameters and their score on the validation set. We choose to use the R2 score, also know as *coefficient of determination*[12] as it is independent of the scale of the values and therefore allows to compare different models across different tasks very easily. The achieved high score close to the maximum score of 1 indicates a good fit.

[12] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html`

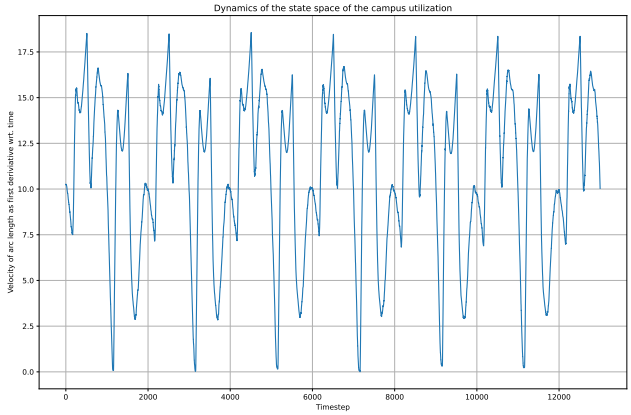


(a) Visualization of the arc length of the state space over the timestep. Note that we are accumulating rounds over the periodic circle.

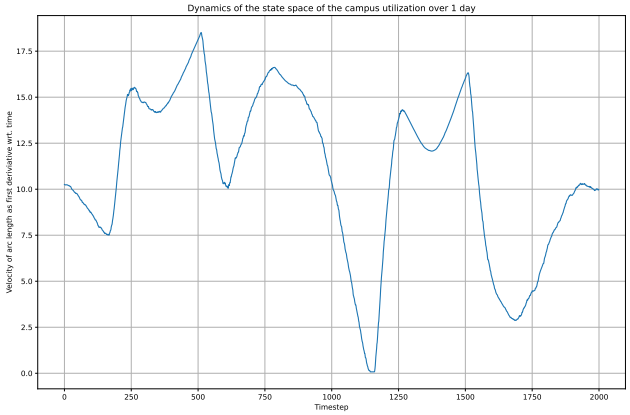


(b) Visualization of the velocity of arc length over the length of the arc, clearly showing the periodic nature of the data. Velocities are calculated based on central differences.

Figure 20: Dynamics of the state space captured by the arc length.



(a) Visualization of the velocity of arc length over time, for all M = 13000 timesteps in the state space.



(b) Visualization of the velocity of arc length over time, for the first day only.

Figure 21: Showing the periodic nature of the state space.

Figure 22 shows the relationship between the arc length and utilization of the MI building as well as the achieved prediction on the full data.

Forecasting the dynamics for unseen future periods in time

In order to predict the utilization of the MI building for the next 14 days, we first need to create a corresponding input vector that we can then input into our learnt mapping from arc length to utilization value. A naive approach is to simply integrate the estimated velocity of the arc length (see figure 21b) over our new timespan of 14 days (= 28000 time steps), accumulating the arc lengths over multiple periods of the circle. This approach is shown in figure 23a.

However, due to the periodic nature of the state space, one should not accumulate over one period, since the state spaces across multiple iterations on the state circle, encoded now in the arc length, are actually considered to be equal. Therefore, 24a shows the results when integrating over the arc length velocity but resetting to zero for each full period (every 20000 timesteps).

ϵ	667
L	1000
λ	$1e^{-3}$
R2	0.99985

Table 3: Best setting of hyperparameters and the achieved validation score.

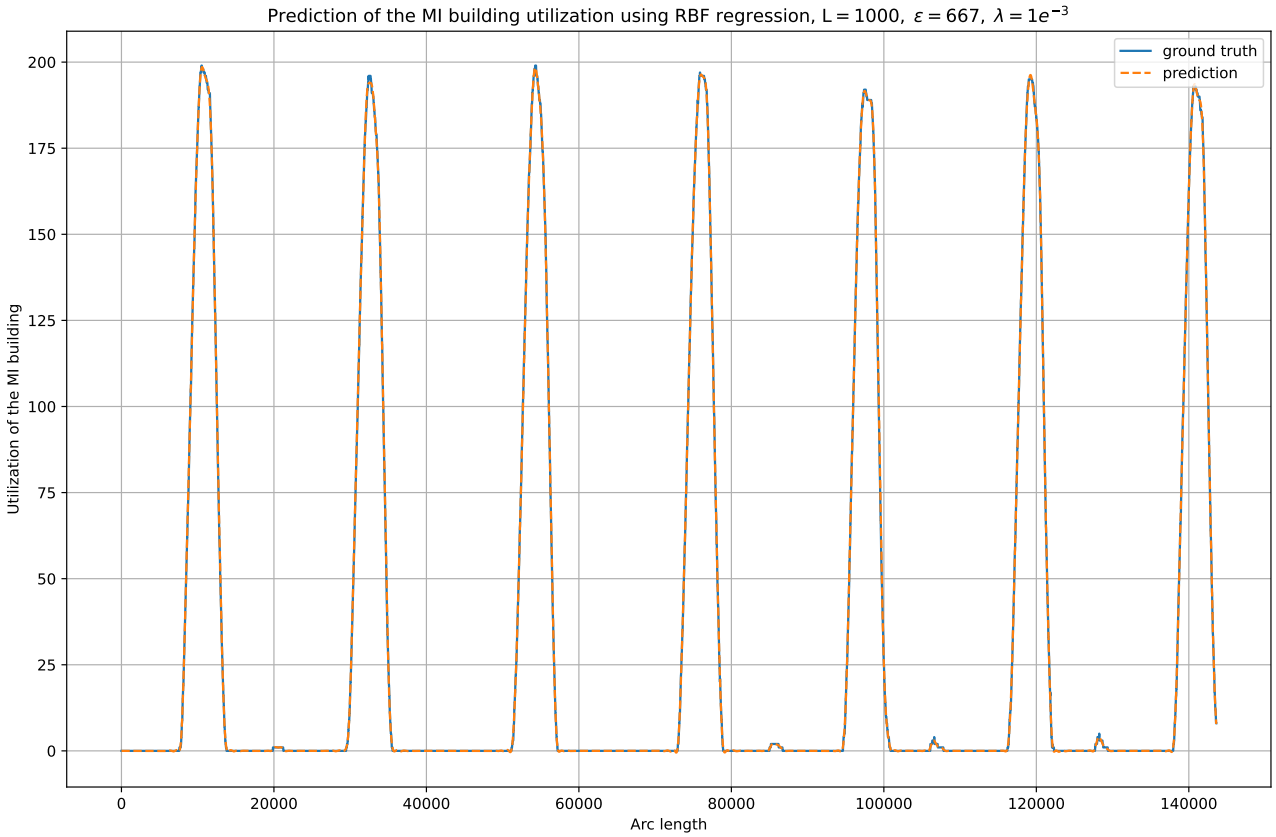
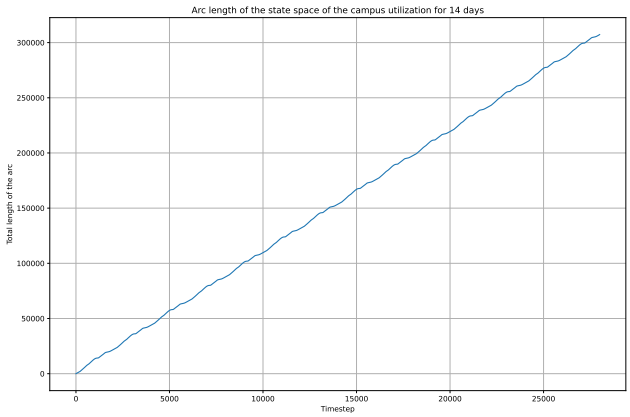


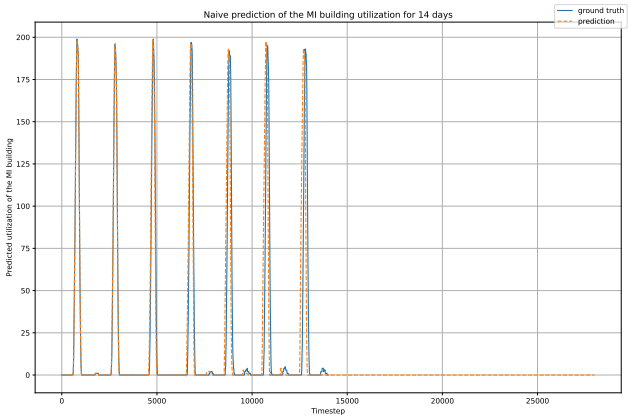
Figure 22: Nonlinear relation between arc length in the state space and the utilization of the MI building as well as the achieved prediction.

Using the learnt mapping to predict measurements in the future

After our new input vector for the prediction of the utilization of the MI building has been crafted in the previous sub-chapter, one is now able to insert it back into our learnt mapping using the RBF regression. Figures 23b and 24b show the final results, depending on the input vector to put into the mapping. One can clearly see, that taking the non-adjusted accumulated arc length over many iterations leads to bad results.This effect can be explained in terms of our RBF implementation not being able to extrapolate which is due to the center of the RBF functions chosen at random only withing the input range of our training data and the asymptotic convergence against zero of our RBF functions away from their centers. As shown in 24b, this issue is no longer appearing when resetting the arc length values after each full period, since then we are only interpolating.

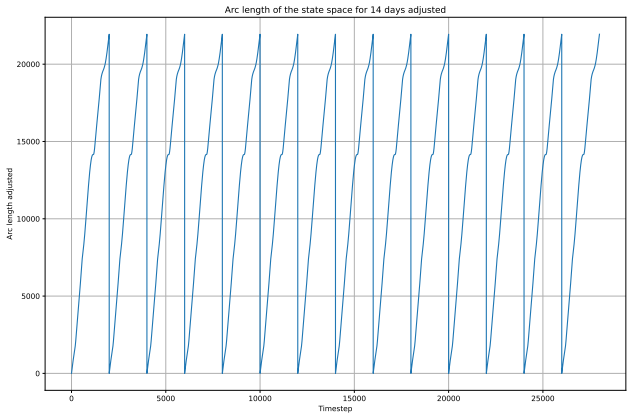


(a) Prediction of the arc length of the state space over the next 14 days, obtained by numerically integrating over the velocity vector field on the arc.

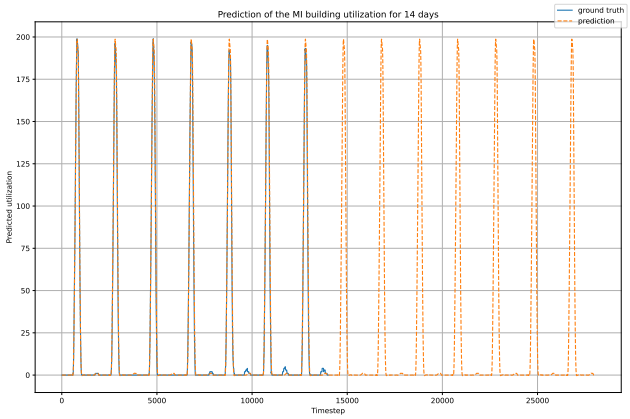


(b) Prediction of the utilization of the MI building over the next 14 days using 23a (left) as an input to the learnt mapping between state space and MI building utilization.

Figure 23: Unsuccessful prediction of the utilization of the MI building over 14 days due to bad extrapolation using RBF regression.



(a) Prediction of the arc length of the state space over the next 14 days, starting from zero after full periods due to the periodic nature of the state space.



(b) Prediction of the utilization of the MI building over the next 14 days using 24a (left) as an input to the learnt mapping between state space and MI building utilization.

Figure 24: successful prediction of the utilization of the MI building over 14 days.

References

[1] Floris Takens. Detecting strange attractors in turbulence. *Lecture notes in Mathematics*, 1981.