Report for exercise 5 from group K

Tasks addressed:	5
Authors:	Oliver Beck (03685783) Junle Li (03748878) Chenqi Zhou (03734992)
Last compiled:	2021–06–23
Source code:	https://github.com/Linnore/MLCMS-Ex5-GroupK

The work on tasks was divided in the following way:

Oliver Beck (03685783)	Task 1	55,0%
	Task 2	0%
	Task 3	55,0%
	Task 4	10,5%
	Task 5	10,0%
Junle Li (03748878)	Task 1	45,0%
	Task 2	45,0%
	Task 3	45,0%
	Task 4	22.5%
	Task 5	40,0%
Chenqi Zhou (03734992)	Task 1	0%
	Task 2	55,0%
	Task 3	0%
	Task 4	67,0%
	Task 5	50,0%

Abstract

In this exercise, we learned how to extract rules from observation data of a dynamical system, and to use these rules to make predictions about future observations.

Python	3.7.10
Jupyter Notebook	6.3.0

Table 1: Software versions

Report on task 1 Approximating functions

Task Description In this first task, you have to demonstrate your understanding of function approximation in two examples. Download the two datasets (A) linear function data.txt and (B) nonlinear function data.txt from Moodle. They contain 1000 one-dimensional points each, with two columns: x (first column) and f(x) (second column).

In all parts, use least-squares minimization to obtain the matrices A and C as described in sections (1.1) and (1.2). For the basis functions, pick the value of ε appropriately and discuss why you picked it like this. Plot the functions you obtain over the data sets (A) and (B), to illustrate how well they approximate the data. Why is it not a good idea to use radial basis functions for dataset (A)?

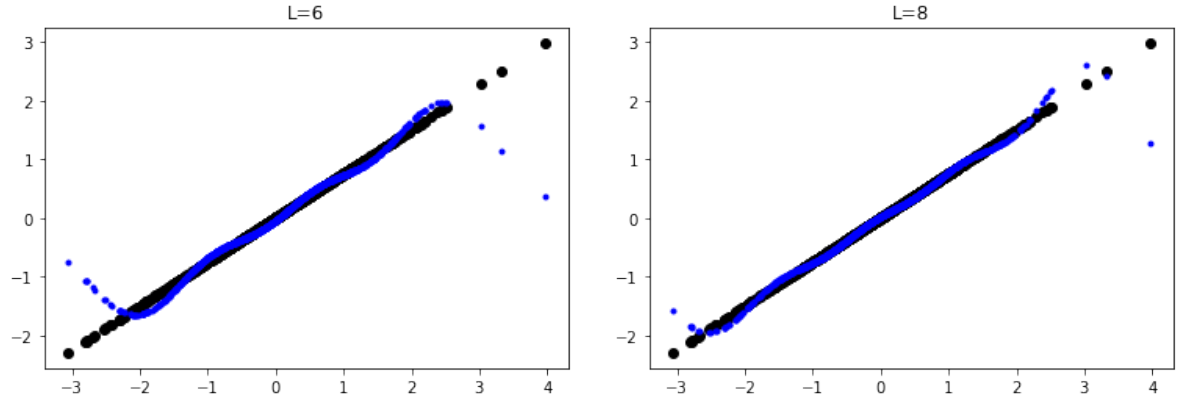
Solution

File Structure The Task 1 Jupyter notebook *task*1.*ipynp* is contained in the main directory along with the other tasks. It uses the Python classes we created in *utils* namingly *linear_approximator.py* and *radial_basis_approxiamtor.py* in order to process the data in subdirectory *data*. The notebook showcases our programming and the results on a high level, while the Python classes do the individual approximation and the detailed groundwork, both nicely documented.

When do we add a bias? If the dimension of the input X is 1, then a column of ones needs to be added to the data matrix X to introduce the bias term, since the method numpy.linalg.lstsq(a, b) needs at least 2 columns to compute the least squire solution for a@x=b. Therefore, only in this case, we add the bias even though the original data is going through the origin.

Why is it not a good idea to use radial basis function for dataset (A)? Since dataset A consists of linearly arranged data points, a linear approximation works ideally. A radial basis function only would have added unneeded complexity with less nice results.

In the figures 1 it can be seen, that for the most part of the curve the prediction fits decently well on to the data points, but not as ideal as with the linear basis function 2. For values outside the center area, with fewer data points the prediction diverges far away from the data points. If we compare figure 1(a) with figure 1(b) we see that a higher number of radial basis components leads to a larger area being covered more accurately. The performance of the prediction gets worse especially for values outside or close to the boundaries. When approximating with radial basis functions it is bound to happen that the edges of the function diverge, far from the linear data points. This makes this approach not ideal for linear functions.



(a) Approximatiion with 6 radial components (b) Approximatiion with 8 radial components

Figure 1: Approximation of a linear sample with radial components

How is the radial basis related to the Gaussian Kernel? From the exercise Notes and the Lecture our knowledge of Gaussian Kernel was refreshed. We noticed that the form of the Gaussian distribution

$$f\left(x|\mu,\sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

and the form of

$$\phi_l\left(x\right) = e^{-\frac{\|x_l - x\|^2}{\varepsilon^2}}$$

are very similar. We know that σ^2 stands for the variance and μ for the mean value. Thus, ε^2 can be seen as scaled variance. The similarity to the Gaussian distribution helps us better understand, visualize the ongoing radial basis approximation and choosing appropriate values for ε. Basically the approximating function $f(x)$ is a linear combination of Gaussian Kernels. With this knowledge, later on we can find better values for ε.

Results

First part: approximate the function in dataset (A) with a linear function. As we can see in figure 2 the approximation of the linear function and the data points that form a linear line fit each other exactly. The function has the equation $y = 0.75x$. Hence it crosses the origin and the bias is equal to 0. We can verify this, when we check the coefficient vector $(2.6E-07, 7.5E-01)$. The first entry is the bias, which is basically 0 the second entry is the first coefficient which corresponds to the gradient of the linear equation. Since the data points are very concentrated, for clearer visualization we only plotted every 20th data point.

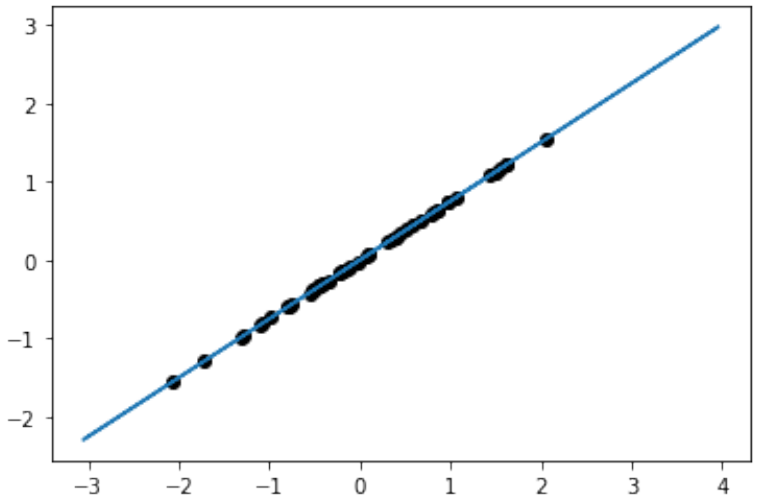


Figure 2: Approximation of the function in dataset (A) with a linear function.

Second part: approximate the function in dataset (B) with a linear function. Similar to the first part we obtain a linear function for the approximation. It has the equation $y = 0.0287x + 0.111$ and by nature doesn't fit the plotted points that lie on a higher degree function in figure 3. Again with the coefficient vector $(0.111, 0.0287)$ we obtain this function.

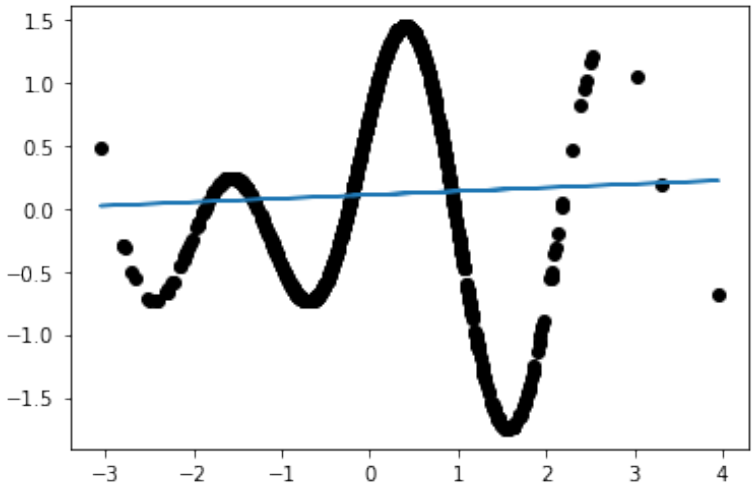


Figure 3: Approximation of the function in dataset (B) with a linear function.

Third part: approximate the function in dataset (B) with a combination of radial functions. We obtain an approximation function that fits the input data points very well 4. For this we use the parameters $L = 10$ and $\varepsilon = \sqrt{2}$. We will explain them in more detail in the corresponding part.

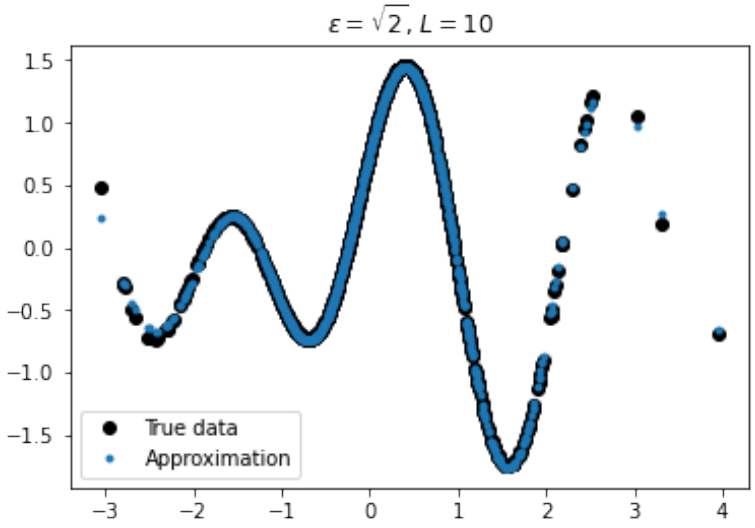


Figure 4: Approximation of the function in dataset (A) with a a combination of radial functions.

Choosing parameter ε What we can observe from the plotted figures is that an epsilon, which is too small mostly leads to an approximation that is 0, with a few points being the exception when the true data reaches a local extreme. The higher the bandwidth ε becomes the more points are non-zero and move towards the true data and a good approximation. Higher density of sample points (in the center) leads to a higher accuracy of the approximation. Once the value of ε has reached a good approximation the accuracy drops again. The higher the ε gets the lower the dimension of the approximation becomes.

What we can conclude from the observations is that the bandwidth ε, as the name already tells gives a kind of bandwidth, a scope, a neighbourhood, in which the input of the single data points is being used to merge into the output approximation. If this bandwidth is to small the approximation is only based on very local values, fluctuating heavily. If it is ideal it seems to have a reasonable scope of the neighbourhood. We found that a good value for ε was generally $\sqrt{2}$, which plays into the formulation of the standard Gaussian distribution. If the bandwidth is too large, it considers the next local minima for the maxima that is to be approximated locally. If the bandwidth is very large it considers all the values and therefore degenerates into some kind of average distribution.

Choosing parameter L For both cases bad/good parameter ε we observe that the accuracy doesn't decrease when L increases. Also the more data points, the better the results.

In case of the bad ε in 7 we notice that the accuracy increases continuously with increasing L up until a certain point after which the graph of the approximation stays identical and unchanged. In case of the good ε we observe increasing accuracy until the approximated values match the true data.

We think that the stagnating accuracy of 6is due to the bandwidth ε being a little to high, which hinders the approximation to reach the peaks of the extrema, since the peaks of the approximation always will be averaged down by the neighbourhood. This issue remains no matter how high the number L of basis functions. After a certain number of L the approximation is very good 6. A gaussian kernel term, which is similar to the radial basis, can cover a single extrema, addition for maxima, subtraction for minima fairly well. So if the number L of additions exceeds the number of extrema the basic outline of the function can be approximated by the term. In our example this happens for L$\geq$8.

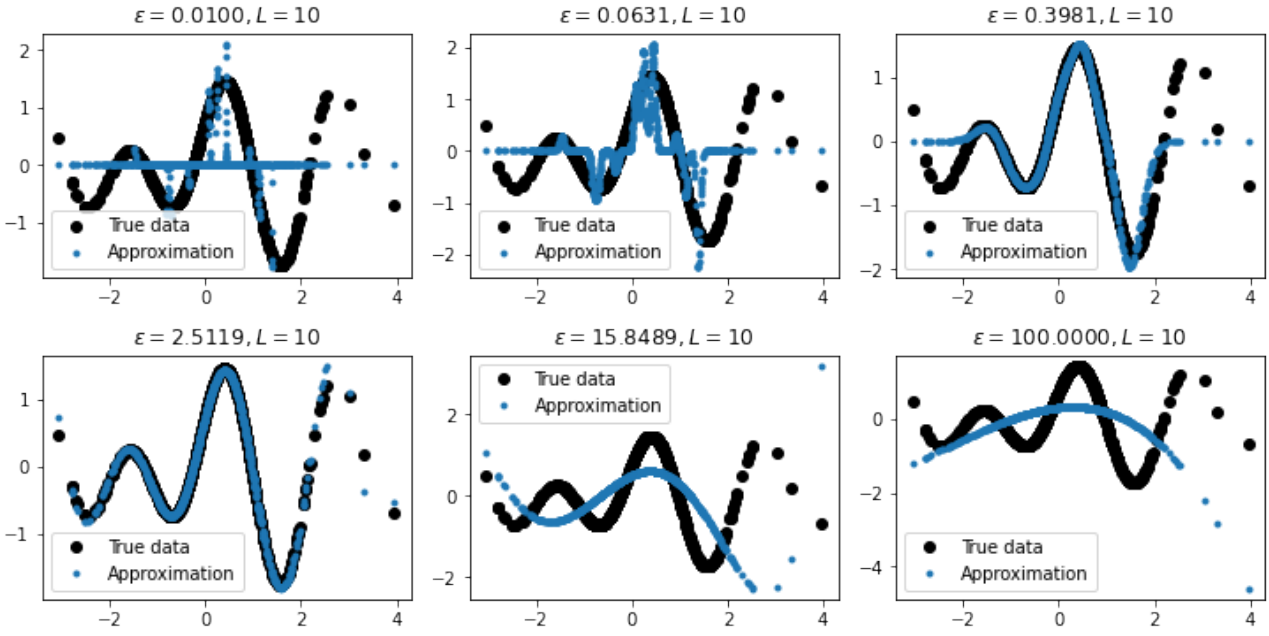


Figure 5: Approximation of the function in dataset (A) with a a combination of radial functions.

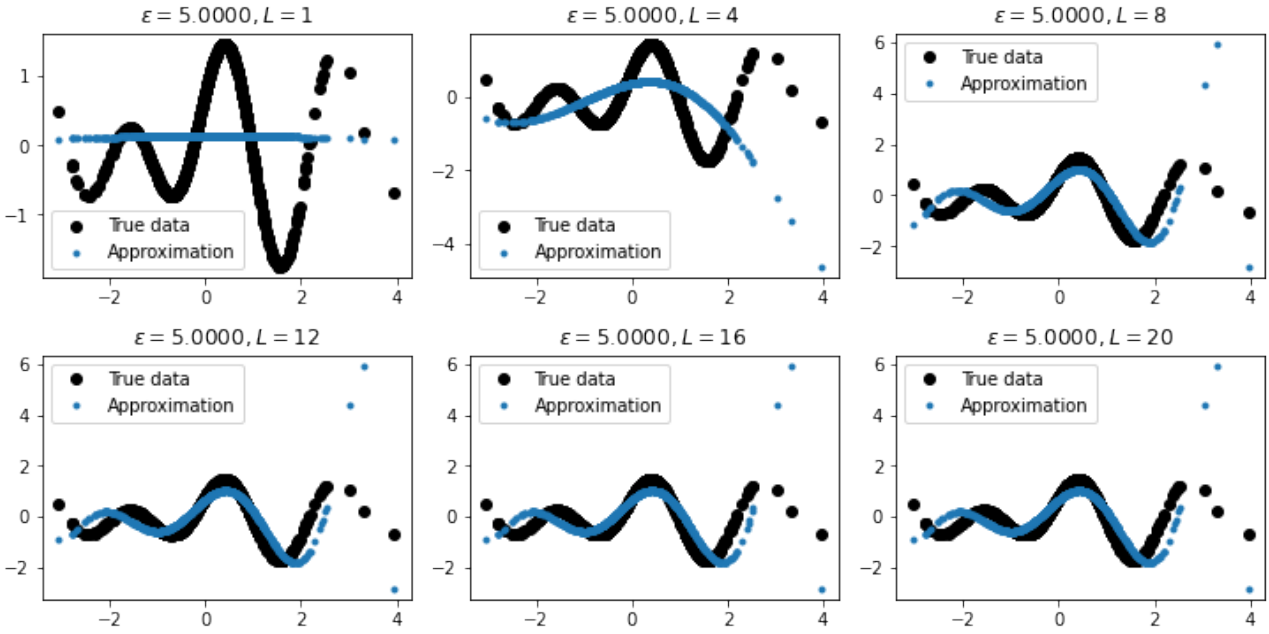


Figure 6: L with bad parameter ε

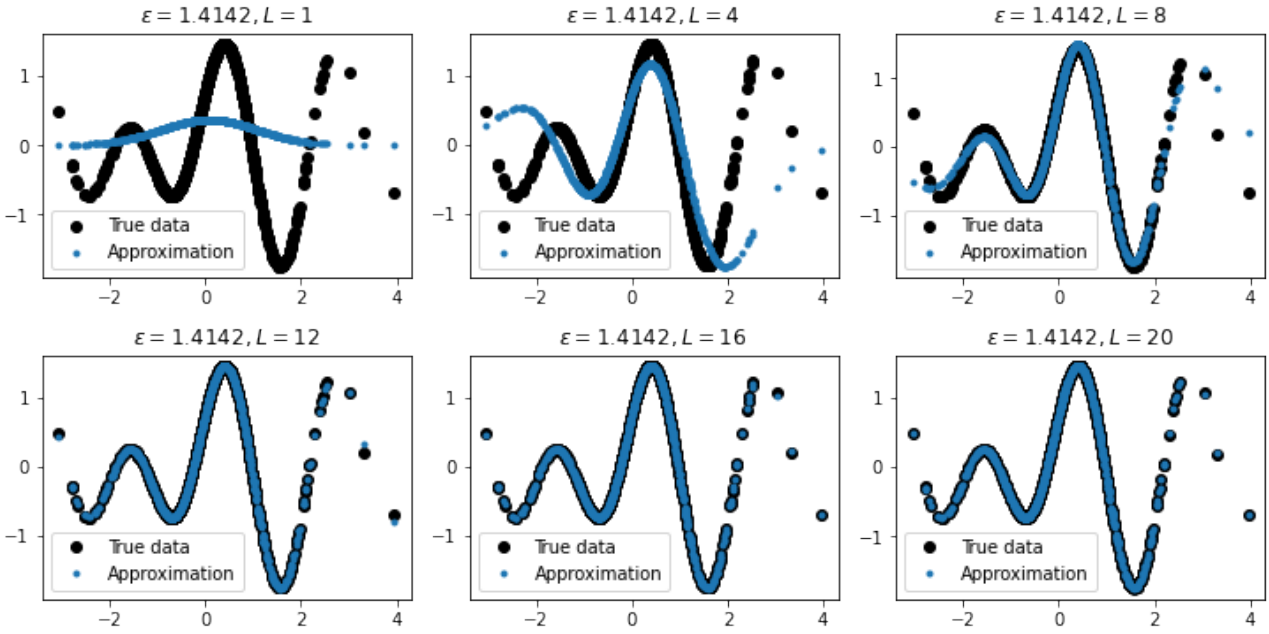


Figure 7: L with good parameter ε

Report on task TASK 2, Approximating linear vector fields

Task Description Download the datasets *linear vectorfield data x0.txt* and *linear vectorfield data x1.txt* from Moodle. They each contain 1000 rows and two columns, for 1000 data points x_0 and x_1 in two dimensions.

Part one You have to estimate the linear vector field that was used to generate the points x_1 from the points x_0. Use the finite-difference formula

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

to estimate the vectors $v^{(k)}$ at all points $x_0^{(k)}$, with a time step Δt that will minimize the error in part 2. Then approximate the matrix $A \in R^{2\times 2}$ with a supervised learning problem: the vector field is linear, so you can expect that for all k,

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

Part two Once you have an estimated matrix $\hat{A} \approx A$, solve the linear system $\dot{x} = \hat{A}x$ with all $x_0^{(k)}$ as initial points, up to a time $T_{end} = 0.1$. You will get estimates for the points $x_1^{(k)}$. Compute the mean squared error to all the known points $x_1^{(k)}$, i.e. compute $\frac{1}{N}\sum_{k=1}^{N}\left|\left|\hat{x}_1^{(k)} - x_1^k\right|\right|$ with N = 1000.

Part three Choose the initial point (10, 10), far outside the initial data. Again solve the linear system with your matrix approximation, for $T_{end} = 100$, and visualize the trajectory as well as the phase portrait in a domain $[-10, 10]^2$.

Solution In part one of the task, we get two datasets, which contain 1000 data points x_0 and x_1 in two dimensions. We can use the finite-difference formula as above to calculate the vector field $\hat{v}^{(k)}$, which describes the movement of X_0 in time Δt. As for how to determine the time step Δt, because the vector field is a linear function (scaled by A) of the point x_0, the time step Δt does not actually matter for predicting x_1 (it would be canceled). In order to prove this and explore the influence of different values of Δt on the error in Part 2, we

take different values of Δt, from 0.001 to 0.1, and calculate Mean Squared Error. The result is that the errors are all the same. So we take $\Delta t = 0.1$ for the convenience of calculation.

For part two, we apply the approximated vector field and solve the linear system up to a time $T_{end} = 0.1$ to get estimates for the points x_1. We calculate the mean squared error (MSE) between prediction for point x_1 and dataset x_1.

In part three, we use approximated vector field and again solve the linear system. We use *Euler's Algorithm* in exercise 3 to simulate the trajectory from the initial point (10, 10) with time step $\Delta t = 0.1$ and $T_{end} = 100$. Then we plot the phase portrait in a domain $[-10, 10]^2$ using *streamplot*, same as in exercise 3. Finally, in order to verify the accuracy of the trajectory in part two, we used *streamplot* to plot the trajectory again.

Results We approximate the linear vector field that was used to generate the points x_1 from the points x_0 via linear regression. The estimated matrix $\hat{A} \in R^{2\times 2}$ is

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

The mean squared error (MSE) of an estimator measures the average of the squares of the errors, that is, the average squared difference between the estimated values $\hat{x_1}$ and the actual value x_1. The mean squared error is 5.266092665224433e-17, which is very small and represents a good approximation.

We use *Euler's Algorithm* to simulate and visualize the trajectory starts from (10, 10) with time step $\Delta t = 0.1$ and $T_{end} = 100$. Then we visualize the phase portrait in a domain $[-10, 10]^2$. The trajectory starts from point (10, 10) and ends at (0, 0). Refer to Figure *Topological classification of hyperbolic equilibria on the plane* in exercise 3, our system belongs to the focus category in topological classes of hyperbolic equilibria. The system has one focus fixed point at (0,0).

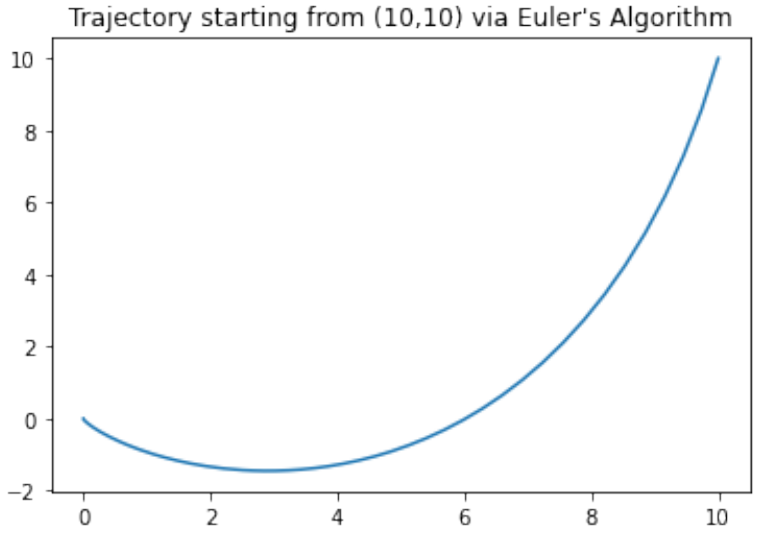


Figure 8: Trajectory starting from (10,10) via Euler's Algorithm

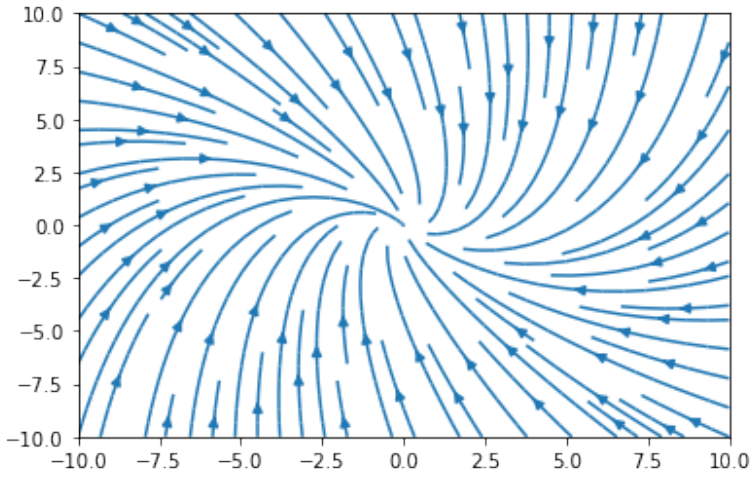
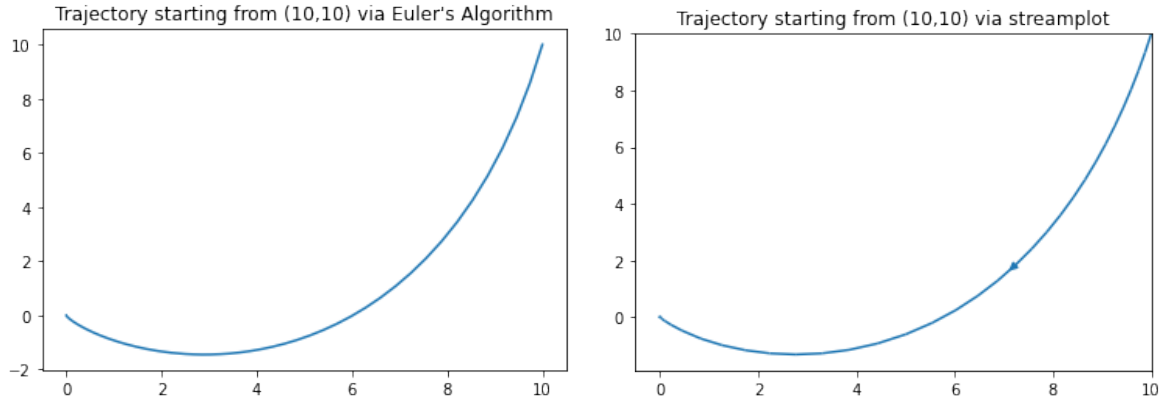


Figure 9: Phase portraits of system

Finally, we compared the trajectories plotted with *Euler's Algorithm* and *streamplot* respectively, and the results are very similar.



(a) Trajectory starting from (10,10) via Euler's Algorithm (b) Trajectory starting from (10,10) via streamplot

Figure 10: Comparison of the two trajectories

Report on task TASK 3, Approximating nonlinear vector fields

Task Description Download the datasets $nonlinear\_vectorfield\_data x0.txt$ and $nonlinear\_vectorfield\_data x1.txt$ from Moodle. They each contain N = 2000 rows and two columns, for N data points x0 and x1 in two dimensions. The first dataset contains the initial points over the domain $[-4.5, 4.5]^2$ while the second dataset contains the same points advanced with an unknown (to you) evolution operator $\psi : T \times \mathbb{R}^2 \to \mathbb{R}^2$ such that

$$x_1^k = \psi\left(\Delta t, x_0^{(k)}\right), k = 1, \ldots, N,$$

with a small (and also unknown) $\Delta t > 0$, similar to the task on linear vector fields. Your task is to study the underlying dynamics of this process.

Subtask 1

Description As in the previous task, try to estimate the vector field describing ψ with a linear operator $A \in \mathbb{R}^{2\times 2}$, such that

$$\frac{d}{dt}\psi\left(t,x\right) \approx f_{linear}\left(x\right) \approx Ax.$$

Once you obtained A, start at each individual initial point $x_0^{(k)}$ and solve up to a small time Δt, so that you obtain an approximate end point $\widehat{x}_1^{(k)}$ 1 as close as possible to the known end point $x_1^{(k)}$ 1 . What is the mean squared error between all the approximated and known end points for your chosen Δt?

Solution Since we are using linear approximation the Δt doesn't influence our approximation in the linear case of x_1, because it is just a scalar value. We set it to 0.1. Other than that we calculate it similar to task 2.

Result The mean squared error is 0.018635040587481 or rounded 1,8 %. This mean squared error is large compared to the values we have seen in the past. This shows us that the approximation with a linear operator isn't effective for this task, estimating the vector field.

Subtask 2

Description Now, try to approximate the vector field using radial basis functions (with the number of centers between 100 and 1000–what is a good number?), such that

$$\frac{d}{dt}\psi\left(t,x\right) \approx f_{rbf}\left(x\right) \approx C_{\phi}\left(x\right)$$

Perform a mean squared error analysis. How do the errors differ to the linear approximation? What do you conclude, is the vector field linear or nonlinear? Why?

Solution We discovered that the mean squared error from part 1 was larger than desired. To find an L that satisfies us we can loop over different L values to see, after which value the mean squared value is low enough.

Result We think an error of $1e-8$ is low enough. This is achieved by L being 300. Bigger L means higher computing cost, so we choose a smaller L that satisfies our desired accuracy. Since we experience greatly varying accuracy for linear and non-linear approximation, we conclude that the vector field is linear. We conclude this foremost, because the mean squared error of the linear approximation is alot higher than in the nonlinear approximation ($1,8e-2$ vs. $1e-8$).

Subtask 3

Description Once you have made your choice, use the approximated vector field to solve the system for a larger time, with all initial points x_0. Where do you end up, i.e. where are the steady states of the system? Are there multiple steady states? Can the system be topologically equivalent to a linear system?

Solution We plot the trajectories of the system at different time steps once with a linear approximator and once with a nonlinear approximator. Time step is chosen to be $\Delta t = 0.1s$

Result For the nonlinear case in figure 11 we can see that the trajectories change within the first second. They stay stable after the first second. The phase portrait contains 5 local stable nodes (arranged like the number five on a die). It also contains 4 local unstable saddle points (arranged in a diamond shape).

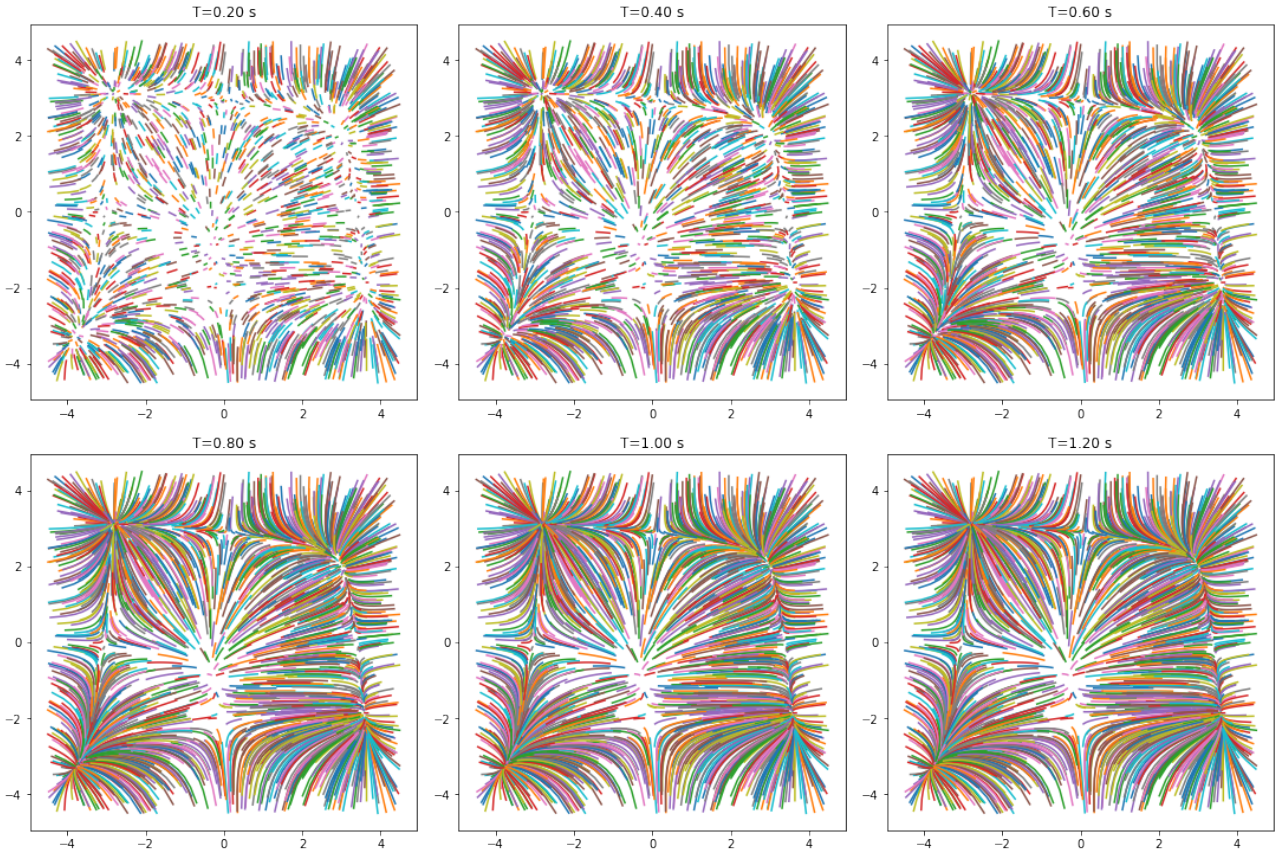


Figure 11: Phase portrait using nonlinear operator

For the linear case in figure 12 we also observe that the phase portraits don't change after the time step one second. There is a single hyperbolic equilibria in the center of the plot at the root, a stable node.

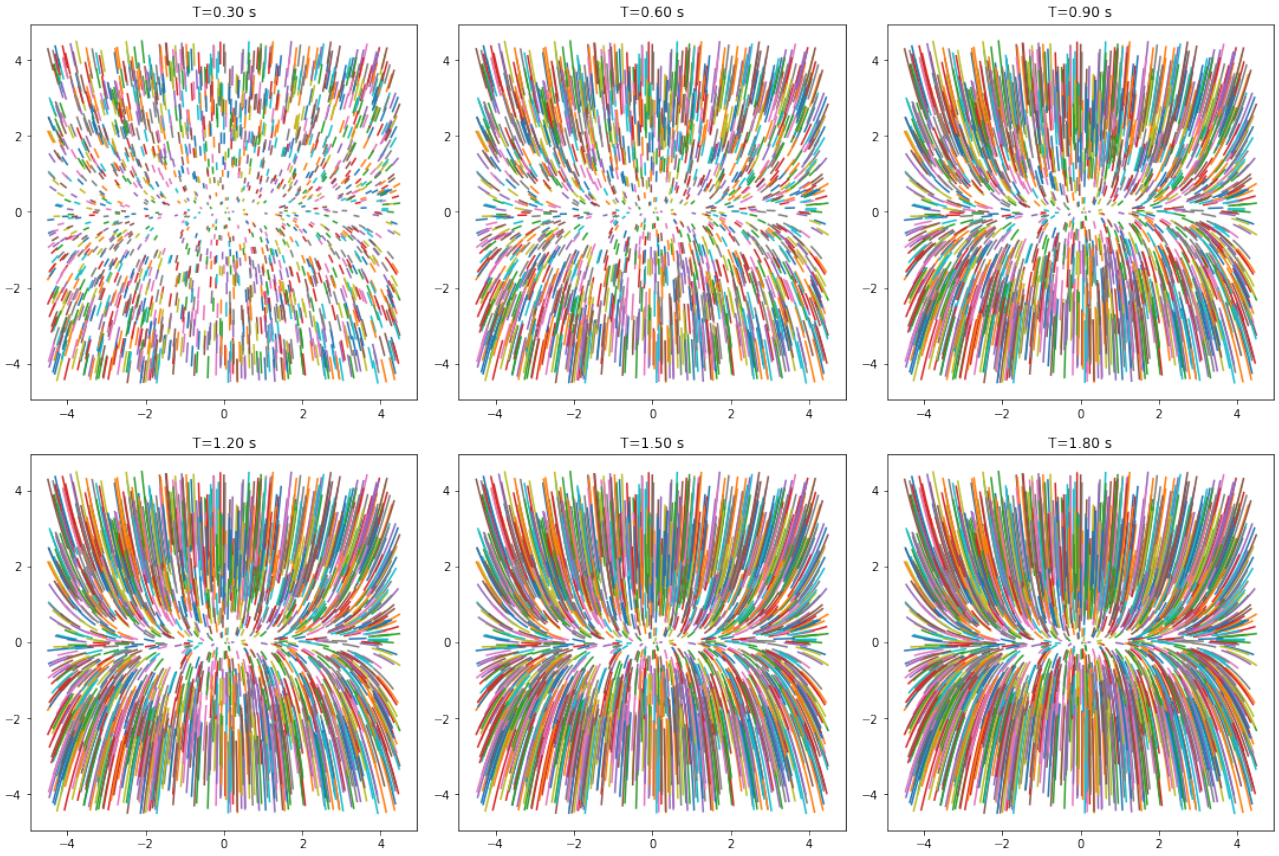


Figure 12: Phase portrait using linear operator

From exercise 3 we know that a topological equivalence to a linear system only can have a single hyperbolic equilibria, just as we saw in figure 12. But when we produce the phase portrait with a non-linear approximator we clearly see several stable nodes and unstable saddle points at the same time, which means the system cannot be topologically equivalent to a linear system. In general the topography of a linear system can represent the topography of a nonlinear system only at local parts, e.g at the neighborhood-manifold around a hyperbolic equilibrium.

Report on task TASK 4, Time-delay embedding

Task Description Part one Part one of this task involves embedding a periodic signal into a state space where each point carries enough information to advance in time. Download the dataset *takens 1.txt*, which contains the data matrix $X \in R^{1000\times 2}$. The two columns are the two coordinates of a closed, one-dimensional manifold. Note: the manifold is one-dimensional, because it can be mapped locally to a one-dimensional Euclidean space. You cannot embed the complete manifold in a one-dimensional space, because it is periodic. Now, plot the first coordinate against the line number in the dataset (the "time"), and then choose a delay Δn of rows and plot the coordinate against its delayed version, similar to $x(t)$ and $x(t + \Delta t)$ in exercise three. According to the Takens theorem, calculate how many coordinates need to be plotted to ensure that the periodic manifold is correctly embedded.

Task Description Part two Part two of this task involves approximating chaotic dynamics from a single time series. You already plotted the Lorenz attractor in exercise three (and use the parameters $\sigma = 10, \rho = 28, \beta = 8/3$ to be in the chaotic regime, from starting point (10, 10, 10)). In this exercise, you have to test Takens theorem even for this fractal set. If the coordinates in your Lorenz attractor are called x, y, z, imagine you can only measure the x-coordinate and do not know about y and z. Takens theorem tells you how you can still get a reasonable idea about the shape of the attractor: 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, for a suitable choice of $\Delta t > 0$. Describe the result in comparison to the attractor in x, y, z coordinates. Now, do the same for the z coordinate, where an embedding should fail. And try to explain why.

Task Description Bonus After you estimated the new state space for the Lorenz attractor with the x-coordinate, approximate the vector field $\hat{\nu}$ on it using radial basis functions. Then, solve the differential equation

$$\frac{d}{dt}(x_1, x_2, x_3) = \hat{\nu}(x_1, x_2, x_3)$$

with your approximation using a standard solver (e.g. *solve_ivp*) and compare the trajectories to the training data.

Solution In part one of the task, we download *dataset takens 1.txt*, which contains 1000 data points X with two columns, which are the two coordinates of a closed, one-dimensional manifold. The line number in the dataset actually represents the time. According to the Takens theorem (here $d = 1$), 3 coordinates $(y(t), y(t+1), y(t+2))$ need to be plotted to ensure that the periodic manifold is correctly embedded with probability 1.

For part two, we use the Lorenz attractor as a reference. According to the Takens theorem, for the Lorenz attractor (two-dimensional manifold), 5 coordinates need to be plotted to ensure that the periodic manifold is correctly embedded with probability 1.

Results First we plot the first coordinate against the line number in the dataset.

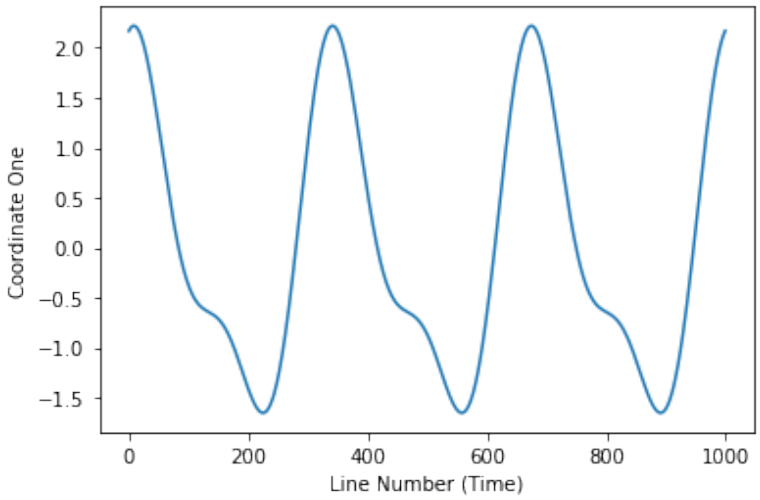
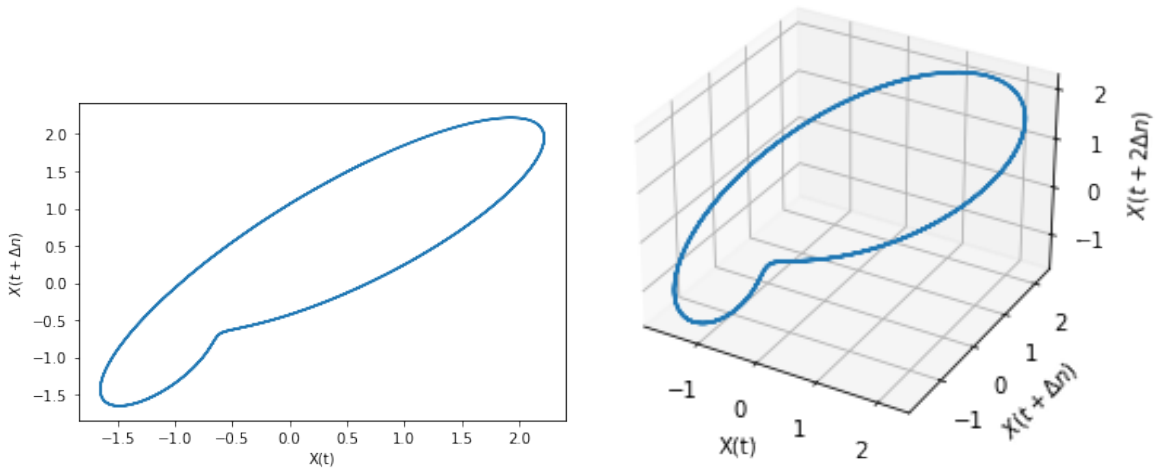


Figure 13: First coordinate against line number

Then we choose a delay Δn of rows and plot the coordinate against its delayed version with 2 coordinates $x(t)$ and $x(t + \Delta n)$ (delay $\Delta n = 20$). The result shows that two coordinates cannot ensure that the periodic manifold is correctly embedded with probability 1.

We plot the coordinate against its delayed version with 3 coordinates $x(t), x(t + \Delta n)$ and $x(t + 2\Delta n)$ (delay $\Delta n = 20$). The results show that 3 coordinates, according to the Takens theorem, ensure that the periodic manifold is correctly embedded and reflect the true model. By comparing the two figures, it can be verified that the two coordinates has correctly reflected the true model.



(a) Two coordinates with $x(t)$ and $x(t + \Delta n)$ (b) Three coordinates with $x(t), x(t + \Delta n)$ and $x(t + 2\Delta n)$

Figure 14: Comparison of 2d and 3d

We plot the Lorenz attractor as in exercise three using the parameters $\sigma = 10, \rho = 28, \beta = 8/3$ to be in the chaotic regime, from starting point (10, 10, 10)).

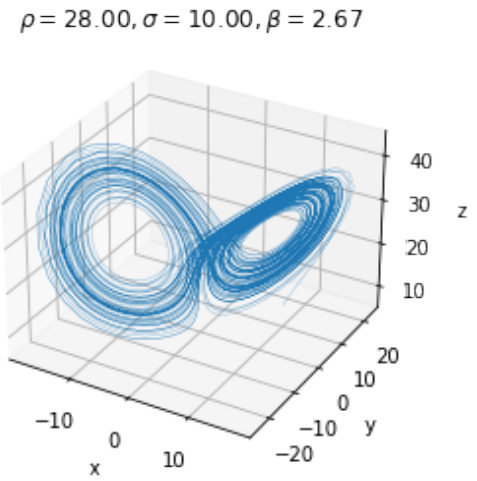
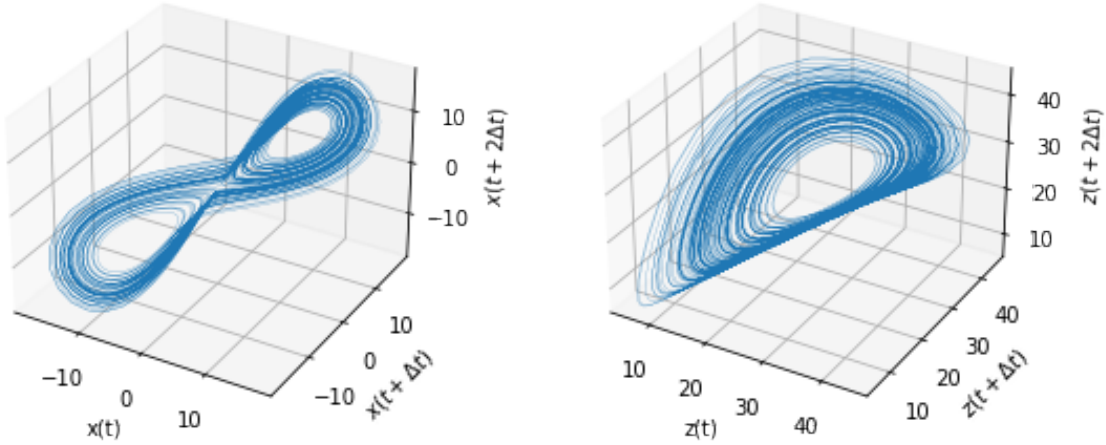


Figure 15: The Lorenz Attractor(3d)

Then we 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 delay $\Delta t = 3$. According to the 3-dimensional diagram of Lorenz attractor, it can be seen that the three coordinates of x correctly reflect the true model. It is possible that the periodic manifold is correctly embedded but it cannot be guaranteed.

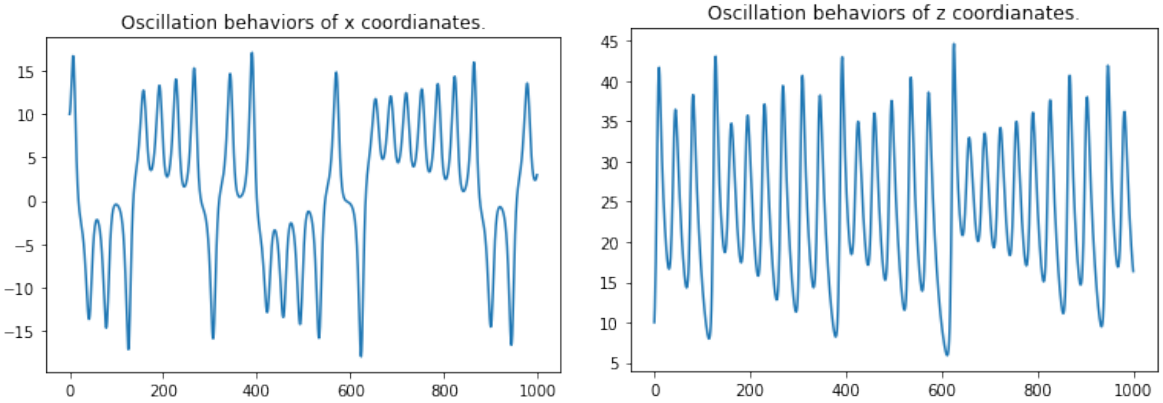
We visualize $z_1 = z(t)$ against $z_2 = z(t + \Delta t)$ and $z_3 = z(t + 2\Delta t)$ in a three-dimensional plot with delay $\Delta t = 3$. According to the 3-dimensional diagram of Lorenz attractor, it can be seen that the periodic manifold is not correctly embedded in three coordinates of z.



(a) The 3d x-coordinate (b) The 3d z-coordinate

Figure 16: Comparison of x and z coordinates

In order to explain the embedding failure using the z coordinates, we plot the oscillation behavior of the x coordinates and the oscillation behavior of the z coordinates separately.



(a) Oscillation behaviors of x coordinates (b) Oscillation behaviors of z coordinates

Figure 17: Comparison of x and z coordinates

Observe that both of x and z coordinates have oscillation behaviors. However, the oscillation behavior of x coordinates looks much complicated than the one of z coordinates. Moreover, the oscillation behavior of z coordinates looks very much like the one of a circle in 3D space. As the local extreme values vary along oscillation of z coordinates, this results in a ring in 3D space. This analysis agrees with the time-delay plot of z coordinates in figure 16(b).

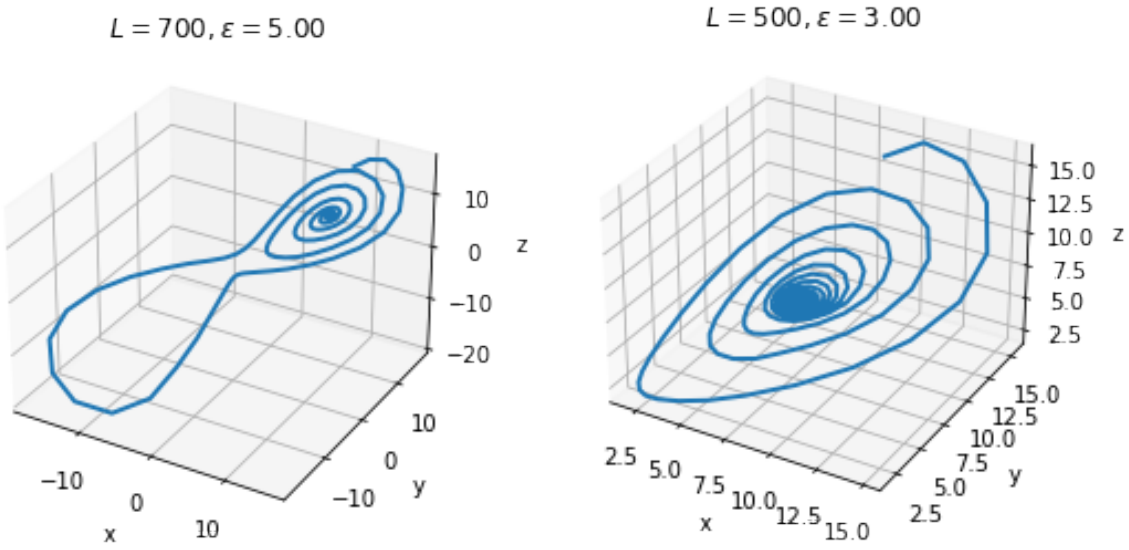
Another reason why an embedding using z coordinates fails is that, the manifold of Lorenz attractor is locally in 2D space. According to Takens Theorem, we need 4 delays of z (in total 5 coordinates) to construct a correct delay embedding with probability 1. Therefore, using 2 delays of z coordinates does not guaranteed an embedding of the true system by theory, which happens in our case.

Bonus We estimate the new state space (daleyed dataset x with delay $\Delta n = 1$) for the Lorenz attractor with the x-coordinate. We approximate the vector field $\hat{\nu}(x_1, x_2, x_3)$ using radial basis functions and calculate the mean squared error (MSE) between prediction and new dataset.

We solve the differential equation with approximated vector field using a standard solver *solve_ivp*. The accuracy of the radial-basis approximator in this case seems not to be very high. Basically it can capture the "spiral-like" feature of at least one attractors in the vector field. However, the trajectory would fall into the

attractors instead of oscillating around the attractors.

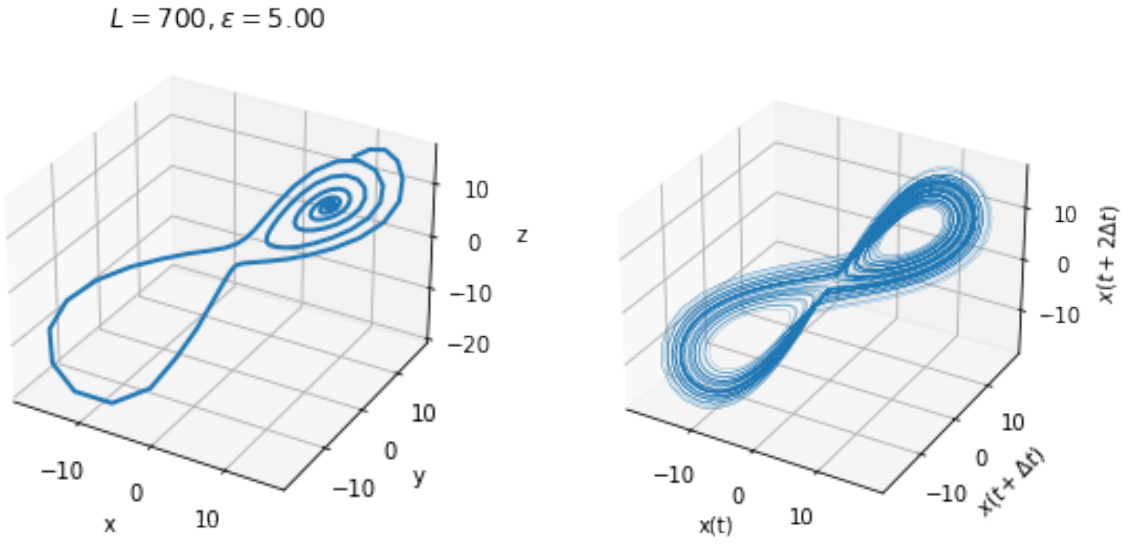
We try different values of hyperparameter L (number of Gaussian kernels) and ε (Bandwidth), which effects the approximation of the vector field, and we plot the corresponding trajectories. The trajectory with parameter $L = 700, \varepsilon = 5.0$ shows better approximation with MSE=3.1543395956569 66e-06, while MSE of $L = 500, \varepsilon = 3.0$ is 0.00012659543298635287. Figure 18(a) shows a representative trajectory in our approximated vector field, which captures the spiral-like field around the two attractors, and the trajectory absorbed by one of them. After many configuration of the hyperparameters L and ε, this is already the best result we obtained. Figure 18(b) shows only one "spiral-like" feature.



(a) The trajectory with $L = 700, \varepsilon = 5.0$

(b) The trajectory with $L = 700, \varepsilon = 3.0$

Figure 18: Comparison of two trajectories



(a) The trajectory with $L = 700, \varepsilon = 5.0$

(b) Time delay embedding by x coordinate

Figure 19: Comparison of the trajectory with the training data

Report on task TASK 5, Learning crowd dynamics

Description

In this task, we are given a dataset containing information of the crowd dynamics of the utilization of the MI building in Garching. Our goal is to analyze this real-life dynamical system.

Solutions and Results

Part One. In this sub-task, we reconstructed a dataset containing rolling version of second to forth columns of the original dataset. After normalization through *sklearn.preprocessing.Normalizer* on the dataset, we applied PCA on the normalized dataset by using *sklearn.decomposition.PCA*. Since the state space of this dynamical system is considered as one-dimensional, according to Takens Theorem, 3 coordinates consist of time-delay observation should give us a correct embedding of the true model with probability one. Therefore, 3 principal components are selected to construct such embedding. Figure 20 shows our result.

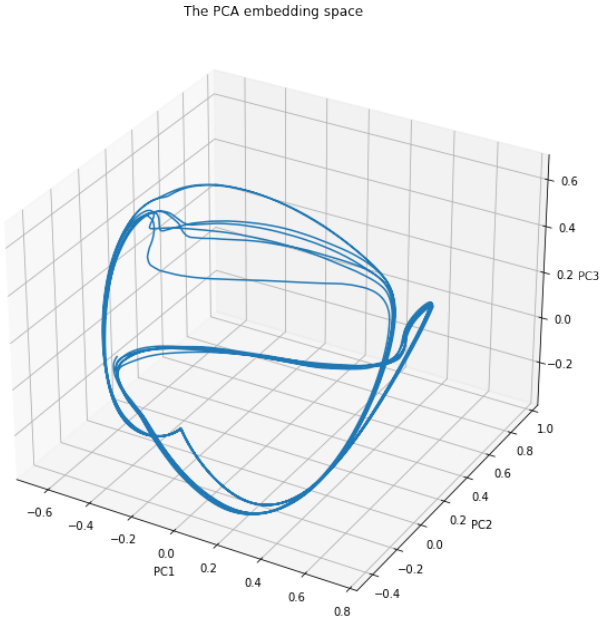
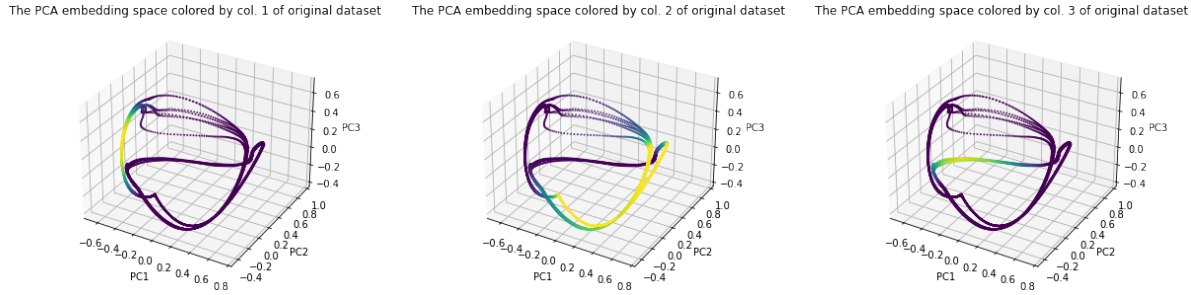
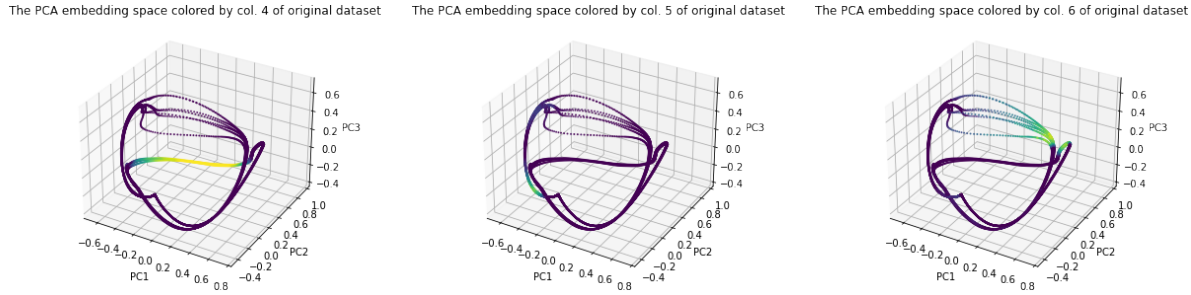


Figure 20: The embedding space constructed by 3 principal components

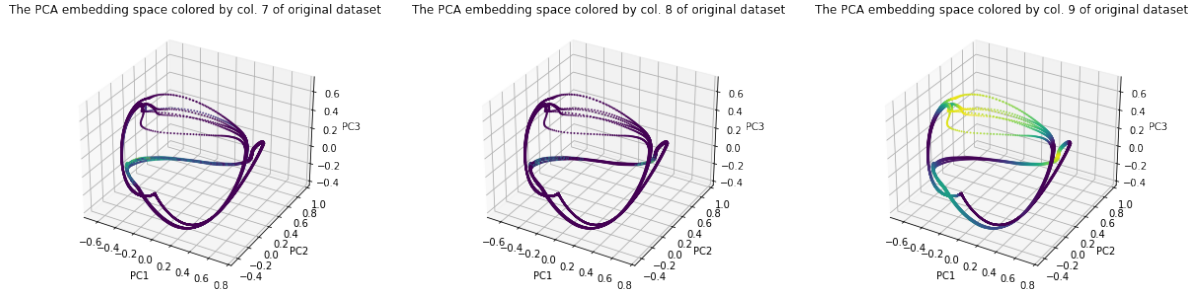
Part Two. Figure 21 shows the required plotting.



(a) Colored by the first measurement (b) Colored by the second measurement (c) Colored by the third measurement



(d) Colored by the forth measurement (e) Colored by the fifth measurement (f) Colored by the sixth measurement



(g) Colored by the seventh measurement (h) Colored by the eighth measurement (i) Colored by the ninth measurement

Figure 21:

Part Three. In this sub-task, we evaluate the velocity of the system on arc length. To do so, the first step is to identify the period of the data. Figure 22 shows the oscillation of the first principal components, from which we can have a general guess about the period–around 2000. Then we wrote codes to verify the time lag between the minimum in [0, 2000] and the minimum in [200, 4000]. It turned out that the period is 1985. The value of the period can also be computed by using the second and third principal components. After we identified the period, we looped over 5 periods of the data to computer the distance of successive data points in the embedding space. As the time lag of successive data points is only 1 given by the original dataset, we can therefore compute the corresponding velocity at each value of the arc length. As mentioned above, each velocity we computed is actually averaged using the velocity of the same position over 5 periods, which would give us better results in theory. Figure 23 shows our result.

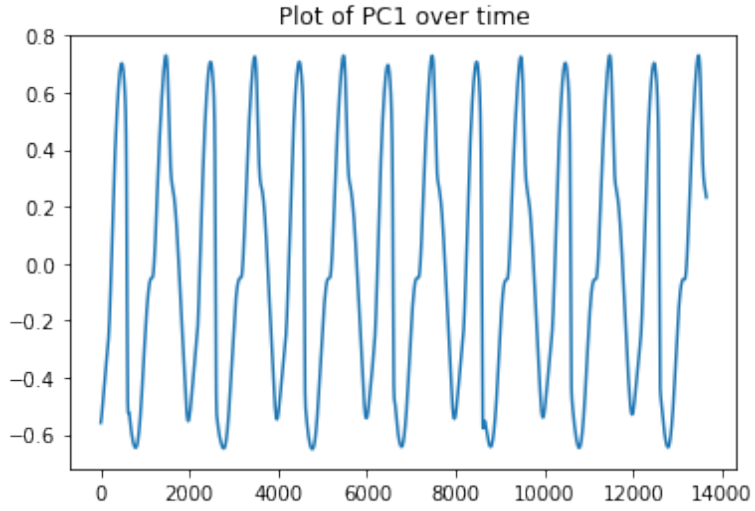


Figure 22: Oscillation behaviors of the first principal component

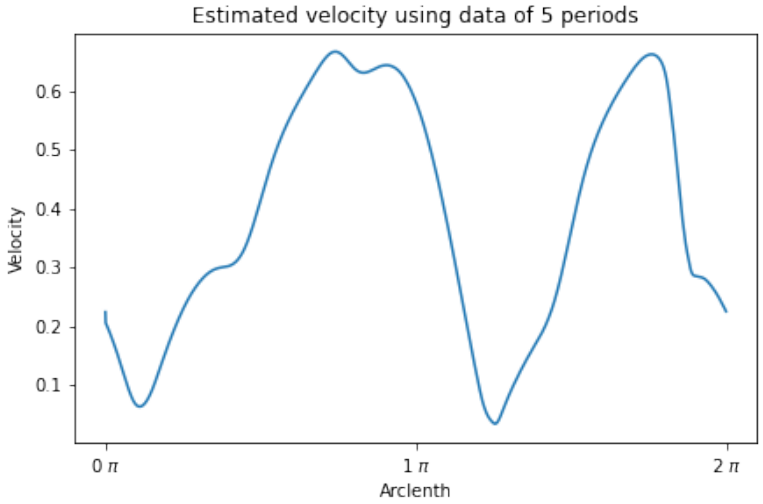


Figure 23: Velocity on Arc Length