Exercise sheet 5
Extracting dynamical systems from data

Due date:	2020–01–09 (3 weeks)
Tasks:	5

A dynamical system is a set of states and a combination of rules that change this state over the change of a single parameter, typically considered as "time". In this exercise, you will learn how to extract rules from observation data of a dynamical system, and to use these rules to make predictions about future observations. All of this will be done using a machine, so in a sense - the machine is "learning" the dynamics of the underlying system from the observations.

1 Extracting functions and vector fields from data

A typical description of the flow (evolution operator) of a dynamical system is infinitesimally, through a vector field. You have seen these in exercise 3. Here, you have to use machine learning to extract a vector field from data. In the first step, you have to understand how to represent functions in a machine - because after all, a vector field is just a special function that assigns each point in a space a certain vector.

The prototypical problem of machine learning is called "supervised learning" and can be stated as follows: Given some data $X = \{x^{(k)}\}_{k=1}^N \subset \mathbb{R}^n$, and function values $F = \{f\left(x^{(k)}\right)\}_{k=1}^N \subset \mathbb{R}^d$, construct a function $\hat{f} : \mathbb{R}^n \to \mathbb{R}^d$ such that the error

$$e\left(\hat{f}\right) = \|f(X) - \hat{f}(X)\|^2 = \|F - \hat{f}(X)\|^2 \tag{1}$$

is small. There are many different versions of this problem, and many sub-problems (like the auto-encoder from last lecture) can be formulated, too. In this exercise, we will take a look at

1. different representations for $\hat{f}$,
2. special $\hat{f}$ that represent vector fields,
3. how to obtain a reasonable representation for the state x, and
4. how this helps to understand systems in crowd dynamics.

1.1 Approximating linear functions

A linear function between two Euclidean spaces $\mathbb{R}^n, \mathbb{R}^d$ with $n, d \in \mathbb{N}$ is a map $f_{\text{linear}} : \mathbb{R}^n \to \mathbb{R}^d$, such that for $x \in \mathbb{R}^n$,

$$f_{\text{linear}}(x) = Ax \in \mathbb{R}^d \tag{2}$$

for some matrix $A \in \mathbb{R}^{d\times n}$.

How do we solve a supervised learning problem with a linear function? Assume we decided that our approximation function should be linear. Then, minimizing $e\left(\hat{f}\right)$ leads to

$$\min_{\hat{f}} e\left(\hat{f}\right) = \min_{\hat{f}} \|F - \hat{f}(X)\|^2 = \min_{A} \|F - XA^T\|^2, \tag{3}$$

where $F \in \mathbb{R}^{N\times d}$, $X \in \mathbb{R}^{N\times n}$, and $A \in \mathbb{R}^{d\times n}$. That means we want to find the matrix A such that the sum of the square of the individual errors is minimal. Here, another important algorithm enters: *least-squares minimization*. Problem (3) has a closed form solution minimizing the least-squares error:

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

In python, you can (and should) use `numpy.linalg.lstsq` or another library routine to compute it [1]. It is instructive to implement it yourself, but you should always use the robust and stable library routines in applications (and the tasks in this exercise).

[1] See `https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.lstsq.html`.

1.2 Approximating nonlinear functions

A nonlinear function between two Euclidean spaces $\mathbb{R}^n, \mathbb{R}^d$ with $n, d \in \mathbb{N}$ is a map $f_{\text{nonlinear}} : \mathbb{R}^n \to \mathbb{R}^d$, such that for $x \in \mathbb{R}^n$,

$$f_{\text{nonlinear}}(x) \in \mathbb{R}^d. \tag{5}$$

Of course this definition is rather arbitrary, and most of machine learning is concerned with the approximation of functions that have more structure (continuity, smoothness, boundedness, etc.).

In this exercise, we will take a look at a particular representation of a nonlinear function that is used in many numerical algorithms: a linear decomposition into nonlinear basis functions. The basic idea to write the unknown function f as a combination of known functions ϕ, such that

$$f(x) = \sum_{l=1}^{L} c_l \phi_l(x),\ \ c_l \in \mathbb{R}^d. \tag{6}$$

If L is finite, only a finite-dimensional space of functions f can be represented in this way. You may have seen this decomposition in Fourier analysis, Taylor decomposition, or in lecture four on neural networks.

Here, we will only consider special functions ϕ: so-called *radial basis functions*, defined by

$$\phi_l(x) = \exp\left(-\|x_l - x\|^2/\epsilon^2\right). \tag{7}$$

The point x_l is the center of the basis function (where it attains its maximum value 1), and the parameter ϵ is the bandwidth. You have already seen this function in lecture four in relation to diffusion maps. Here, we will not use it to represent data, but to represent functions. The center x_l of the basis function is typically just a random point in the data set, one for each basis function. If you pick as many basis functions as you have data points, you can always perfectly fit any function *on the given data*—however, generalization, interpolation and extrapolation may not be very good. You need to choose L and ϵ appropriately for every task.

How do we solve a supervised learning problem with a basis of radial functions? The benefit of decomposing f into the functions ϕ_l in a linear way is that we can use exactly the same method to approximate nonlinear functions as we used for the linear case. With the decomposition (6), minimizing $e\left(\hat{f}\right)$ leads to

$$\min_{\hat{f}} e\left(\hat{f}\right) = \min_{\hat{f}} \|F - \hat{f}(X)\|^2 = \min_{C} \|F - \phi(X)C^T\|^2, \tag{8}$$

where $C \in \mathbb{R}^{d\times L}$ contains the list of coefficients c_l, and

$$\phi(X) := (\phi_1(X), \phi_2(X), \dots, \phi_L(X))^T \in \mathbb{R}^{N\times L}$$

is a concatenation of all L basis functions evaluated on all N data points. You can solve this problem again with least-squares minimization, because the only unknown is the matrix C.

Note: the parameter ϵ for the radial basis functions can be chosen similarly to diffusion maps. Sometimes it is beneficial to choose it larger, especially if the true function f is very smooth.

Note: if you do not understand or cannot solve the estimation problem for radial basis functions yourself, you are allowed to use the python library function `scipy.interpolate.Rbf` (or any other radial basis function libary in the language of your choice)[2]. You will not get points for the code in this case, which is quite a significant hit - but at least you can get through the exercises.

Note: it is not a coincidence that I called the basis functions ϕ_l, the same symbol I used for the eigenfunctions of the Laplace-Beltrami operator in the exercise on diffusion maps. On manifolds, these eigenfunctions constitute an "ideal" basis for all smooth functions, as they can be ordered from most-smooth to least-smooth by their associated eigenvalue. Of course, this is yet another connection to Fourier decomposition.

Note: if you replace the radial basis functions ϕ_l with polynomials $(x_0 - x)^l$, you obtain an approximation that is similar to a Taylor decomposition around a point x_0.

Note: if you want to understand the connection between radial basis functions, Gaussian processes, and neural networks, you have to first read [9] and then [4, 8] (arXiv versions [3, 7]). You can choose to pursue this as a final project.

[2] See `https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.Rbf.html`.

1.3 Approximating vector fields

As you know from lecture three, a vector field is a section of the tangent bundle: $\nu : M \to TM$, such that $\nu(x) \in T_x M$. That means a vector field assigns each point x in the space M a vector in the tangent space at that point. You have already plotted vector fields in exercise three, to visualize dynamical systems through their phase portraits. If the space M is just some Euclidean space $\mathbb{R}^n$, the tangent spaces $T_x M$ are usually identified with the same Euclidean space, such that a vector field turns into a function $\nu : \mathbb{R}^n \to \mathbb{R}^n$. This is great, because after reading sections 1.1 and 1.2, you know how to approximate such functions!

If you need to approximate a vector field through supervised learning, you would need points X and vectors in a dataset V, to then approximate $\hat{\nu}(x^{(k)}) = v^{(k)} \in V$. Sometimes you will encounter data that is not directly in this form—instead, you may have points X_0 at time $t = 0$ and points X_1 at time $t = \Delta t$, a short time later. A rather naive but straightforward way to turn this into the standard supervised learning problem for the vector field is to compute

$$\hat{v}^{(k)} = \frac{x_1^{(k)} - x_0^{(k)}}{\Delta t}. \tag{9}$$

Since you now have a vector $\hat{v}^{(k)}$ for every $x_0^{(k)}$, you can approximate the vector field $\hat{\nu}(x_0^{(k)}) = \hat{v}^{(k)}$. Of course, much more advanced techniques to estimate the vectors from trajectory data exist, for example [10, 11, 5, 2].

2 Time-delay embedding

In many cases (and in all cases where data from the real world is involved), we cannot capture the full state of the system. Only observations of it can be measured, using many different sensors—cameras, temperature sensors, pressure sensors, etc. In 1981, Takens and Ruelle developed the mathematical foundations to obtain state space information from observations [12, 14]. They heavily rely on the theorem of Whitney [15], stating how many smooth functions are needed to embed smooth manifolds.

Let $k \geq d \in \mathbb{N}$, and $\mathcal{M} \subset \mathbb{R}^k$ be a d-dimensional, compact, smooth, connected, oriented, Riemannian manifold. Together with the results from Packard et al. [6] and Aeyels [1], the definitions and theorems of Takens and Ruelle [14] describe the concept of observability of state spaces (here, the manifold $\mathcal{M}$) of nonlinear dynamical systems. A dynamical system is defined through its state space and a diffeomorphism $\psi : \mathcal{M} \to \mathcal{M}$.

Theorem 1. ***Generic delay embeddings.*** *For pairs (ψ, y), $\psi : \mathcal{M} \to \mathcal{M}$ a smooth diffeomorphism and $y : \mathcal{M} \to \mathbb{R}$ a smooth function, it is a generic property that the map $E_{(\psi,y)} : \mathcal{M} \to \mathbb{R}^{2d+1}$, defined by*

$$E_{(\psi,y)}(x) = \left(y(x), y(\psi(x)), \ldots, y(\underbrace{\psi \circ \cdots \circ \psi}_{2d \text{ times}}(x)) \right)$$

is an embedding of $\mathcal{M}$; here, "smooth" means at least C^2.

Genericity in this context is defined as "an open and dense set of pairs (ψ, y)" in the C^2 function space. Open and dense sets can have measure zero, so Sauer et al. [13] later refined this result significantly by introducing the concept of prevalence (a "probability one" analog in infinite dimensional spaces).

For this exercise, Takens' theorem will be essential in task 3, where you encounter the dilemma of having too few observations per time step to predict the future. The theorem essentially states that you can just use time-delayed (or time-advanced) versions of the observable as a new "state", and it will work just as well as if you had access to the original state x.

Note: the number of points per exercise is a rough estimate of how much time you should spend on each task.

Task 1/5: Approximating functions **Points: 20/100**

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).

1. **First part**: approximate the function in dataset (A) with a linear function.
2. **Second part**: approximate the function in dataset (B) with a linear function.
3. **Third part**: approximate the function in dataset (B) with a combination of radial functions.

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)?

Note: you should take great care with the least squares code and the radial basis functions in this task, as you need to re-use them in the following tasks. The code should be nicely documented, modular, and concise, as always.

Task 2/5: Approximating linear vector fields **Points: 20/100**

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. In **part one** of the task, 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 from section (1.3) to estimate the vectors $v^{(k)}$ at all points $x_0^{(k)}$, and then approximate the matrix $A \in \mathbb{R}^{2\times 2}$ with a supervised learning problem: the vector field is linear, so you can expect that for all k,

$$\nu(x_0^{(k)}) = v^{(k)} = Ax_0^{(k)}. \tag{10}$$

For **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, for a time $\Delta t = 0.1$. 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 $\sum_{k=1}^{1000} \left\| \hat{x}_1^{(k)} - x_1^{(k)} \right\|^2$.

In **part three**, choose the initial point $(10, 10)$, far outside the initial data. Again solve the linear system with your matrix approximation, for $T = 100$ seconds, and visualize the trajectory as well as the phase portrait in a domain $[-10, 10]^2$.

Task 3/5: Approximating nonlinear vector fields **Points: 20/100**

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 x_0 and x_1 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

$$x_1^{(k)} = \psi\left(\Delta t, x_0^{(k)}\right),\ k = 1, \dots, N, \tag{11}$$

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.

1. 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

$$\frac{d}{dt}\psi(t, x) \approx \hat{f}_{\text{linear}}(x) = Ax. \tag{12}$$

   Once you obtained A, start at each individual initial point $x_0^{(k)}$ and solve (12) up to a small time Δt, so that you obtain an approximate end point $\hat{x}_1^{(k)}$ as close as possible to the known end point $x_1^{(k)}$. What is the mean squared error $\sum_{k=1}^{N} \left\| \hat{x}_1^{(k)} - x_1^{(k)} \right\|^2$ between all the approximated and known end points for your chosen Δt?

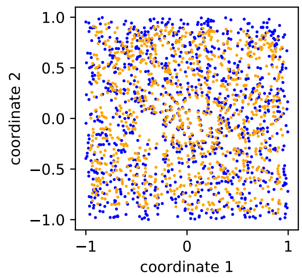

Figure 1: Datasets x_0 (blue) and x_1 (orange), with 2000 points scattered in two dimensions.

2. Now, try to approximate the vector field using radial basis functions, such that

$$\frac{d}{dt}\psi(t,x) \approx \hat{f}_{\text{rbf}}(x) = C\phi(x). \tag{13}$$

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

3. 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?

Task 4/5: Time-delay embedding **Points: 20/100**

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 $X \in \mathbb{R}^{1000\times 2}$. 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 Takens theorem stated above, how many coordinates do you need to plot to be sure that the periodic manifold is embedded correctly?

Part two of this task involves approximating chaotic dynamics from a single time series. You already plotted the Lorenz attractor in exercise three (take a look if you do not remember). 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. Why do you think that is?

Bonus (5 points): 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.

Part three of this task involves the crowd dynamics example from the third exercise. All of you generated the x coordinate of a single pedestrian in the third exercise over time, for an obstacle position where pedestrians could move freely and did not get stuck. Plotting $x(t)$ against $x(t+\Delta t)$ resulted in a quite noisy version of a limit cycle. With Takens theorem, you can now explain why it still gave reasonable results. There is another great feature of the theorem you have to test in this part: adding a large number of delays and then reducing the dimension again will remove noisy components of the data. To test this, concatenate 200 delays of the x

coordinate of the individual pedestrian into one data point, such that $p(t) = [x(t), x(t+\Delta t), \ldots, x(t+199\Delta t)]^T \in \mathbb{R}^{200}$. Create more data points by sliding t along the time series, i.e. create $p(0)$, $p(0.4)$, $p(0.8)$, etc. This should result in a dataset $P \in \mathbb{R}^{1300\times 200}$, if the x time series had 1500 points. Analyze this data set by plotting the first two principal components of it against each other. What do you observe?

Task 5/5: Learning crowd dynamics **Points: 20/100**

In this task, you have to learn a dynamical system that predicts the utilization of the MI building in Garching. The dataset `MI_timesteps.txt` contains utilization data for several measurement areas on the campus, over the course of seven weekdays. Students enter the area from the U-Bahn in the morning, move to the MI-building, walk to one of the two mensas (student and Max Planck) during lunch, go back to studying and then move back to the U-Bahn in the evening. You can ignore a burn-in period of 1000 time steps at the beginning of the file.

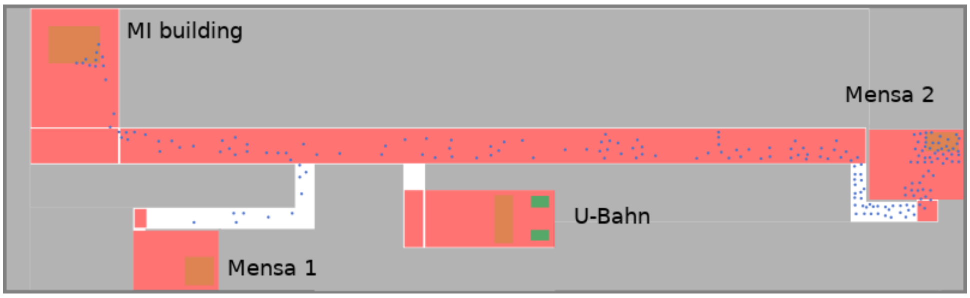

Figure 2: TUM Garching campus abstraction, with MI building, U-Bahn and two mensas. The red areas are the counting devices producing the observations.

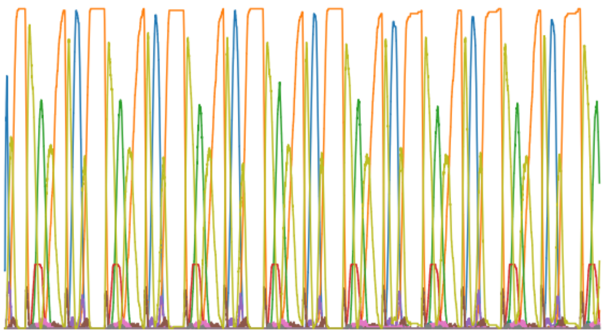

Figure 3: Dataset in the timestep file. Over the course of seven days, several utilization measurements (colors) are taken in several areas on the campus.

This task involves the following parts:

1. Create a reasonable state space of the given system. It is periodic and has no parametric dependence, so it probably will be a one-dimensional, closed loop. How many dimensions will you need to embed it according to Takens theorem? Create a delay embedding with 350 delays of the first three measurement areas (columns 2,3,4 in the file), and use as many principal components as necessary according to Takens.

2. In the embedding space you created in the first part, there are now many different points. Color the points by the first coordinate of the delay embedding, i.e. by the measurement at the first time of the delays, for all nine measurement areas (you have to create nine plots).

3. Learn the dynamics on the periodic curve you embedded in the principal components. To do this, consider the time step that is also available in the file, and determine how fast the system advances over the PCA space at every point in the space. You know which point is the predecessor of which other point, since all of them come from a single time series of measurements. Ideally, you compute the arclength of the curve in the PCA space and then approximate the change of arclength over time: a vector field on the arclength!

4. Predict the utilization of the MI building (first measurement area, first column in the file after the time steps) for the next 14 days with your system, and plot the results over time. Compare the results with the given data.

References

[1] Dirk Aeyels. Generic observability of differentiable systems. *SIAM Journal on Control and Optimization*, 19(5):595–603, sep 1981.

[2] Felix Dietrich, Thomas N. Thiem, and Ioannis G. Kevrekidis. On the koopman operator of algorithms. *arXiv*, 2019.

[3] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv*, 2017.

[4] Jaehoon Lee, Jascha Sohl-dickstein, Jeffrey Pennington, Roman Novak, Sam Schoenholz, and Yasaman Bahri. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018.

[5] Alexandre Mauroy and Jorge Goncalves. Koopman-based lifting techniques for nonlinear systems identification. *arXiv*, 2017.

[6] N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw. Geometry from a time series. *Physical Review Letters*, 45(9):712–716, 1980.

[7] Guofei Pang, Liu Yang, and George Em Karniadakis. Neural-net-induced gaussian process regression for function approximation and pde solution. *arXiv*, 2018.

[8] Guofei Pang, Liu Yang, and George Em Karniadakis. Neural-net-induced gaussian process regression for function approximation and PDE solution. *Journal of Computational Physics*, 384:270–288, May 2019.

[9] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation And Machine Learning)*. The MIT Press, 2005.

[10] R. Rico-Martinez, J.S. Anderson, and I.G. Kevrekidis. Continuous-time nonlinear signal processing: a neural network based approach for gray box identification. In *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*. IEEE, 1994.

[11] R. Rico-Martínez and I.G. Kevrekidis. *Nonlinear system identification using neural networks: dynamics and instabilities*, chapter 16, pages 409–442. Elsevier Science, 1995.

[12] David Ruelle and Floris Takens. On the nature of turbulence. *Commun. Math. Phys.*, 20(3):167–192, Sep 1971.

[13] Tim Sauer, James A. Yorke, and Martin Casdagli. Embedology. *Journal of Statistical Physics*, 65(3):579–616, 1991.

[14] Floris Takens. Detecting strange attractors in turbulence. *Lecture Notes in Mathematics*, pages 366–381, 1981.

[15] Hassler Whitney. Differentiable manifolds. *The Annals of Mathematics*, 37(3):645, jul 1936.