Report for exercise 6 from group H

Tasks addressed:	5
Authors:	Taiba Basit (03734212) Zeenat Farheen (03734213) Fabian Nhan (03687620)
Last compiled:	2021–07–14
Source code:	https://github.com/Combo1/MLCMS/tree/main/Final_Project

The work on tasks was divided in the following way:

Taiba Basit (03734212)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%
Zeenat Farheen (03734213)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%
Fabian Nhan (03687620)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%

Report on task Task 1, Description of the Koopman operator

A majority of methods from dynamical systems analysis rely on Poincare's geometric picture that focuses on "dynamics of states". Much of the framework used today is built around notions from differential geometry, trajectories and invariant manifolds . While these method have been used successfully, it has drawbacks in handling high-dimensional ,ill-described, and uncertain systems, which are more common nowadays with big data measurements. Moreover, understanding the governing equations in a non-linear dynamical systems is difficult as they can be topologically complex and sometimes do not have a closed form.

Koopman operator is one such solution that shifts the focus of studying dynamical systems from "dynamics of states" to "dynamics of observables." Observables are basically measurements in the state space. With the Koopman Operator approach, we study the underlying dynamics of the system by looking at evolution of these measurements, rather than directly at state space evolution.

The Koopman operator is an infinite-dimensional, linear operator that can capture full nonlinear dynamics. We utilize it to analyze the evolution of functions on the state space, by expanding them into a basis of eigenfunctions of the Koopman operator, to study the dynamics of high-dimensional dynamical systems.

Some of the advantages of using Koopman operator is:

1. It is completely data driven i.e. we can construct a good approximation of the system without having the knowledge of the underlying system
2. It can handle high dimensional data unlike the differential geometry methods
3. As we work in the linear space, we can easily apply already well-understood linear techniques like spectral analysis to analyze the dynamics of the system.
4. Additionally the Koopman operator is suitable for visualizing high-dimensional dynamical systems, since we can track objects attached to our initial conditions.

However we have a trade-off of complexity. Some of the disadvantages of using the koopman operator are:

1. As we are analyzing the evolution of functions on state space, there is no immediate connection to our physical intuition, which makes it difficult to comprehend the meaning of the system.
2. Additionally, this approach is inherently infinite dimensional, even when the underlying state space is finite dimensional.

The Koopman operator presents a picture for "dynamics of observables". It makes measurements along trajectories, such that we need fewer initial conditions on our dynamical system, but longer run-time to study its behavior. The Koopman operator can be defined in a time-discrete setting and continuous-time dynamical system. In the time-discrete setting is defined as:

$$U_T \colon F \longrightarrow F, [U_T f](p) = f(T(p)),$$

with f being a function in the function space F. U_T is a composition of f and T, while f is an observable and T the iterated map.

In the continuous-time dynamical system, there is not a Koopman operator, but a semigroup of operators U^t given by a generator U [1].

$$[U^t f](p) = f(\phi^t(p)),$$

According to [2] the eigenvalues and eigenfunctions of the Koopman operator capture long term dynamics of observables and serve as low-dimensional approximation of the infinite-dimensional operator. The Koopman mode however are vectors to reconstruct the state of the system as a linear combination of Koopman eigenfunctions. So the Koopman operator defines a new dynamical system which conserves the evolution of observables of the original system [3].

The figure 1 represents concisely how we move from the state space to the measurement space. The top path shows the state space evolution by evolution operator F. The bottom path shows the evolution in measurement space by the Koopman operator K. The connection between the states and observables is through the full-state

observable $g(x) = x$ and (μ_k, ϕ_k, v_k), the set of "tuples" of Koopman eigenvalues, eigenfunctions, and modes required to reconstruct the full state.

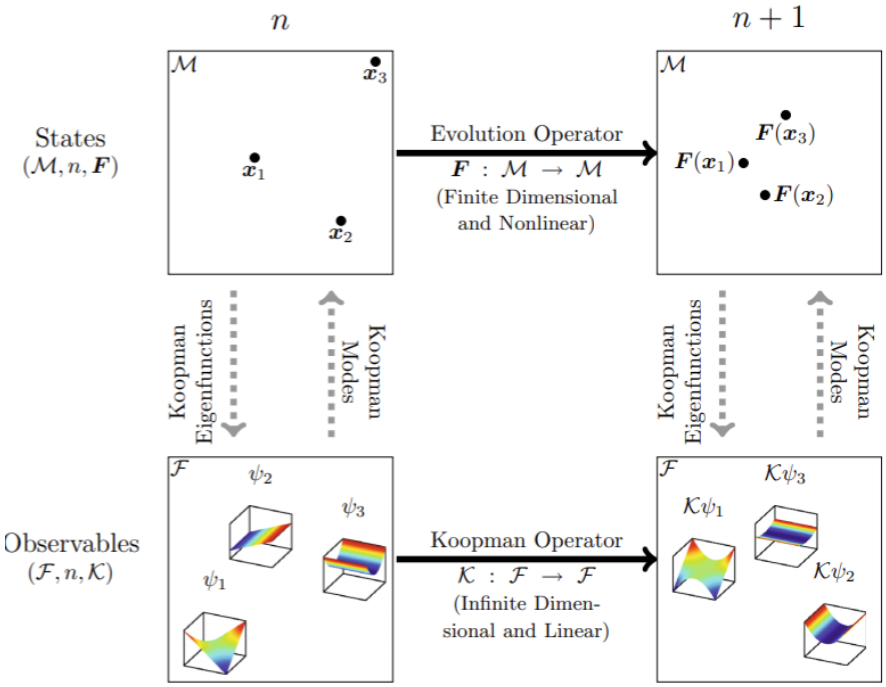

Figure 1: Koopman Operator

Report on task Task 2, First implementation of the EDMD algorithm

As discussed previously, the Koopman Operator is an infinite dimensional linear operator. However, working in infinite dimensions is not feasible and we thus try to get an approximation of the Koopman Operator in finite dimension. This approximation is then a matrix and calculating eigenvalues and eigenfunctions of this matrix is then possible. We are interested in finding the eigenfunctions of the Koopman operator because once we have the eigenfunctions, we can represent our chosen set of observables in terms of these eigenfunctions and observe linear dynamics in the Koopman invariant subspace.

One method that helps us to approximate the Koopman Operator and therefore the Koopman eigenvalue, eigenfunction and koopman modes is Extended Dynamic Mode Decomposition (EDMD).

This procedure requires a data set of snapshot pairs $\{(x_m, y_m)\}$, which are snapshots of the system state and a dictionary of observables D = $\{\phi_1\phi_2...\phi_M\}$ [2].The optimal choice of dictionary elements remains an open question but for now we assume that D is "rich enough" to accurately approximate a few of the leading Koopman eigenfunctions.

We also need to provide a B, a vector of weights which is required to reconstruct the full-state observable using the elements of dictionary D. Once we have all the prerequisites, the algorithm looks as follows:

1. Approximating the Koopman Operator and its Eigenfunctions:
   To calculate the Koopman operator we minimize :

$$J = \frac{1}{2}\sum_{m=1}^{M} |(\Psi(y_m) - \Psi(x_m)Ka|^2$$

2. Then we perform Singular Value Decomposition on this matrix K to obtain the set of eigenvalues μ , eigenvectors $\xi$and left eigenvectors w.

3. We get the j-th eigenfunction of the Koopman operator by multiplying j-th eigenvector ξ_j with the vector valued function of the dictionary of observables Ψ.

$$\phi_j(x) = \Psi(x)\xi_j,$$

4. Computing the Koopman Modes with the left eigenvectors and vector B:

$$v_i \triangleq (w_i * B)^T,$$

The code for implementation is present in file edmd_algo_test.ipynb and our implementation of the EDMD algorithm is present in file edmd_algorithm.py. We followed the algorithm presented in paper[2] to implement the EDMD algorithm as follows:

1. Generate data snapshot pairs in the form of matrix X and Y where X is the initial state of system and Y is the time evolved state.
2. Build a dictionary of choice e.g.polynomial with degree 3.
   dictionary = PolynomialFeatures(degree)
3. Calculate the transform of X and by applying the dictionary to generate psi_X and psi_Y.
   psi_X = dictionary.fit_transform(X)
   psi_Y = dictionary.fit_transform(Y)
4. Calculate matrix B as defined in eq(16) in Williams et.al. using scipy.linalg.lstsq method
   B = scipy.linalg.lstsq(psi_X, X, cond=None)[0]
5. Calculate Koopman operator K as defined in eq(12) in [2] using scipy.linalg.lstsq
   K = np.linalg.lstsq(psi_X,psi_Y,rcond=None)[0]
6. Calculate the eigenvectors and eigenvalues of K using SVD decomposition
   u,s,vh = np.linalg.svd(K)
7. Calculate Koopman modes as defined in eq(20)
   V = (u@B).T
8. Calculate Koopman eigenfunctions as defined in eq(18)
   phi = psi_X @ vh
9. Transform eigenvalues into a diaginal matrix
   s_diag = np.diag(s)

To test our implementation of EDMD algorithm, we try to detect the underlying non-linear dynamics of Hopf ODE systems defined by the equations below with $\mu = 1$

$$\dot{y_0} = -y_1 + y_0(\mu - y_0^2 - y_1^2)$$
$$\dot{y_1} = y_0 + y_1(\mu - y_0^2 - y_1^2)$$

Figure 2 shows the plots of initial and final position of the system after solving the system with **solve_ivp** for a time duration of 0.04 in 21 timesteps with 64 initial points. Here X represents the initial state of the system and Y represents the final time evolved state of the system. We then fit the snapshot pairs (x_m, y_m) on our implementation of EDMD algorithm. We fit the model using a Polynomial dictionary of degree 3.

After obtaining the approximations for Koopman eigenfunctions, Koopman modes and Koopman eigenvalues, we predict the next state of the system by applying Koopman operator on the measurement $\Phi(X)$ to obtain Y_{pred}. We can reconstruct the state space from the measure space using the below equation:

$$F(x) = (Kg)(x) = \sum_{k=1}^{N_k} v_k(K\phi_k)(x) = \sum_{k=1}^{N_k} \mu_k v_k \phi_k$$

Figure 3 shows the predicted state of the system after applying EDMD algorithm.

We observe that the reconstructed system is similar to the original system in Figure 2. There are some visible differences in the shape of the limit cycle but overall the implementation seems to work. We also plotted the streamplot of the orgininal system and the reconstructed system as shown in figure 4 and 5. We see that the two plots seem similar.

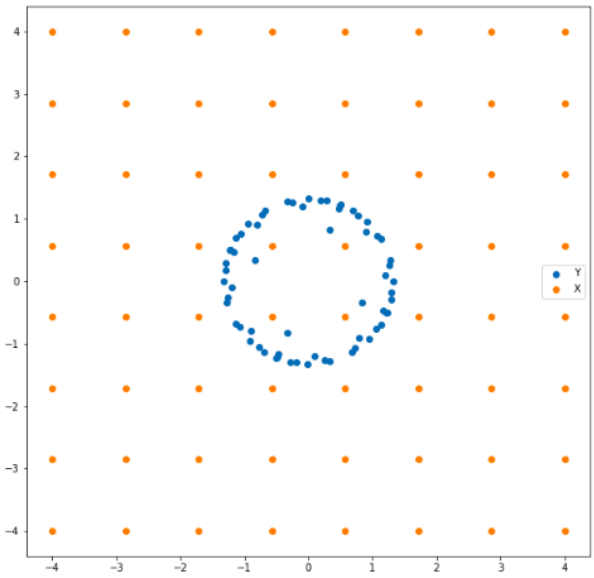

Figure 2: Initial and final state of the Hopf ODE system solved for 21 timesteps

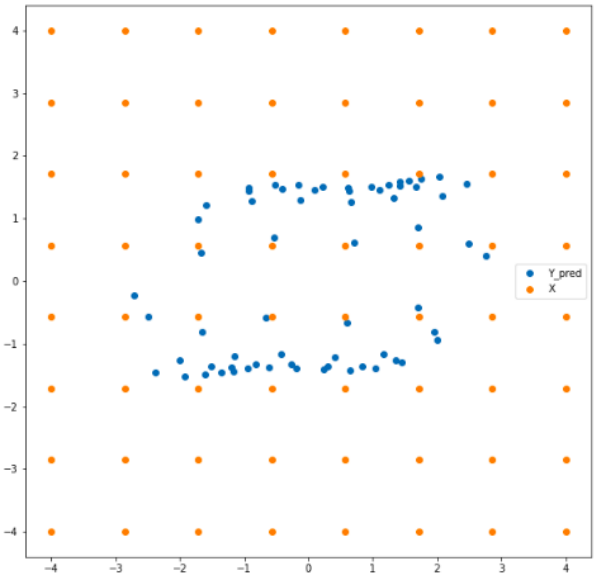

Figure 3: Initial and final state of the Hopf ODE system after EDMD algorithm

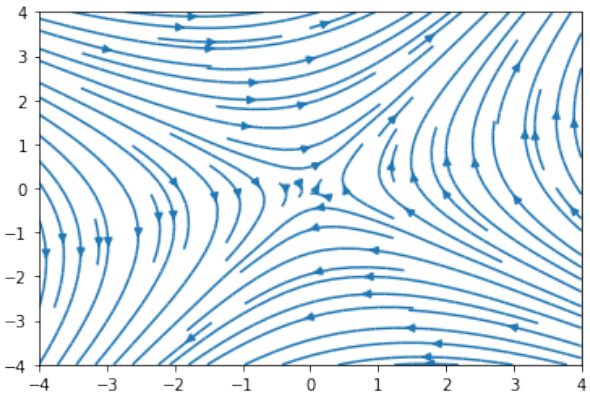

Figure 4: Streamplot of Hopf ODE system using original data

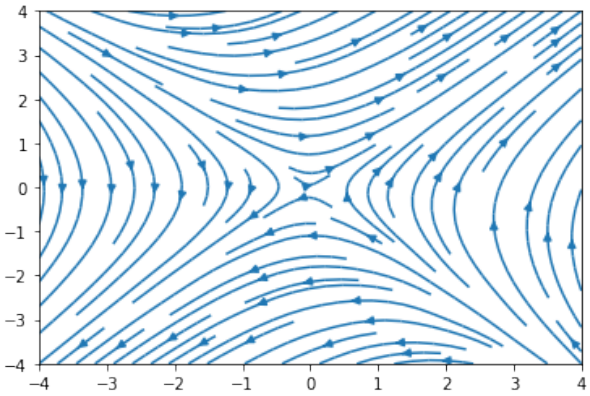

Figure 5: Streamplot of Hopf ODE system using reconstructed data

Report on task Task 3, Tests on an example in the paper by Williams et al

Testing on Our Implementation

In this task, we try to reproduce the results of Linear Time Invariant system example as presented in paper by Williams et. al. [2]. The system under consideration is an LTI system defined as

$$x(n+1) = \begin{bmatrix} 0.9 & -0.1 \\ 0.0 & 0.8 \end{bmatrix} x(n) = Jx(n)$$

Because the underlying dynamics are linear, we expect that the Koopman approach contains the eigendecomposition of J. We can analytically derive the Koopman eigenfunctions and eigenvalues as

$$\phi_{ij}(x,y) = \left(\frac{x-y}{\sqrt{2}}\right)^i y^j,$$
$$\lambda_{ij} = (0.9)^i (0.8)^j$$

for $i, j \in Z$.

1. **Data Preparation** We created matrix of 100 initial points X of shape (100,2) and matrix of time evolved state Y of shape (100,2) Figure 6 shows the initial and final points of the system

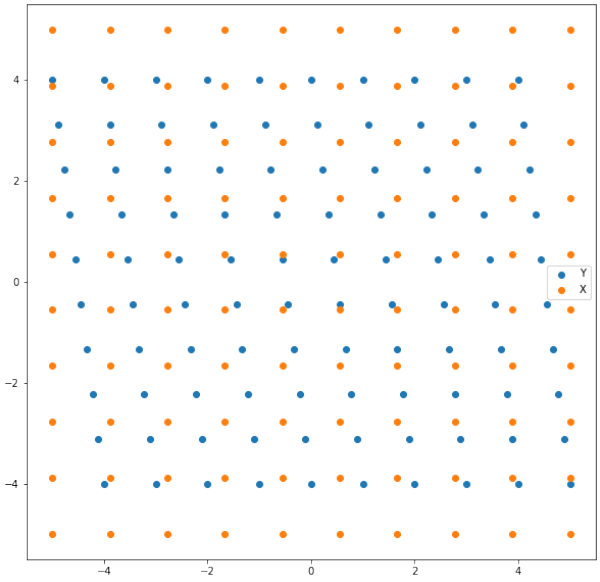

Figure 6: Plot of initial and final states of the LTI system

2. **Analytical Equation** Next we calculate the Koopman eigenfunctions using analytical expressions as shown above and interpolate them using datagrid function of scipy library. Figure 7 shows the pseudocolor plots of first 8 eigenfunctions ordered by their corresponding eigenvalues. Here the eigenfunctions are normalized in the range (-1,1). The plots obtained were exactly same as the ones presented in the paper.

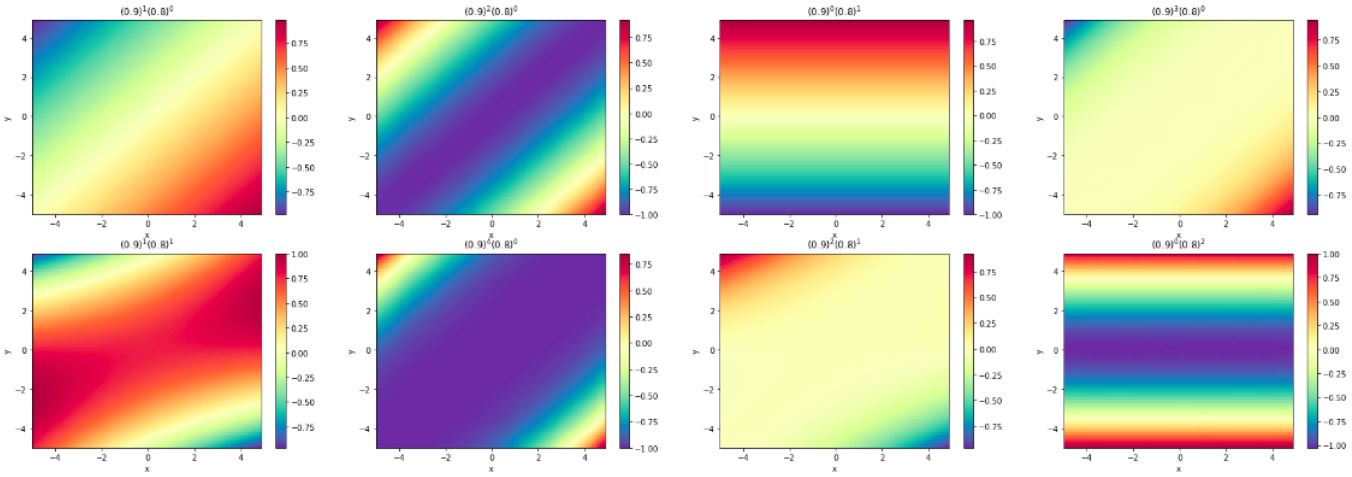

Figure 7: Pseudocolor plots of first 8 eigenfunctions computed analytically along with their corresponding eigenvalues

3. **Koopman Implementation** Next, we ran our implementation of EDMD algorithm on the generated data snapshot pairs X and Y. We chose a polynomial dictionary of degree 4. The code for this is present in LTI_polynomial.ipynb In the paper [2], the authors choose a dictionary of Hermite polynomials of degree 4 to approximate the system. We also implemented the EDMD algorithm on the data with a dictionary of Hermite polynomials with degree 4 as shown in figure 8. We used scipy.special.hermite method to generate the required Hermite polynomials. The code for this is present in file LTI_hermite.ipynb

$$\{H_0(x)H_0(y),\ H_1(x)H_0(y),\ H_2(x)H_0(y),\ H_3(x)H_0(y),\ H_4(x)H_0(y),$$
$$H_0(x)H_1(y),\ H_1(x)H_1(y),\ H_2(x)H_1(y),\ H_3(x)H_1(y),\ H_4(x)H_1(y),$$
$$H_0(x)H_2(y),\ H_1(x)H_2(y),\ H_2(x)H_2(y),\ H_3(x)H_2(y),\ H_4(x)H_2(y),$$
$$H_0(x)H_3(y),\ H_1(x)H_3(y),\ H_2(x)H_3(y),\ H_3(x)H_3(y),\ H_4(x)H_3(y),$$
$$H_0(x)H_4(y),\ H_1(x)H_4(y),\ H_2(x)H_4(y),\ H_3(x)H_4(y),\ H_4(x)H_4(y)\},$$

Figure 8: The Hermite Dictionary

4. **Plotting of the results** The pseudocolor plots obtained for the first 8 non-trivial eigenfunctions with a polynomial dictionary is shown in Figure 9. We see that the results obtained are not similar to the expected results. However, the eigenvalues are still similar to the analytically calculated eigenvalues of Koopman operator. Assuming that the incorrect plots were because we did not choose the right dictionary as suggested by the authors of [2], we changed the dictionary to Hermite polynomials of degree 4.

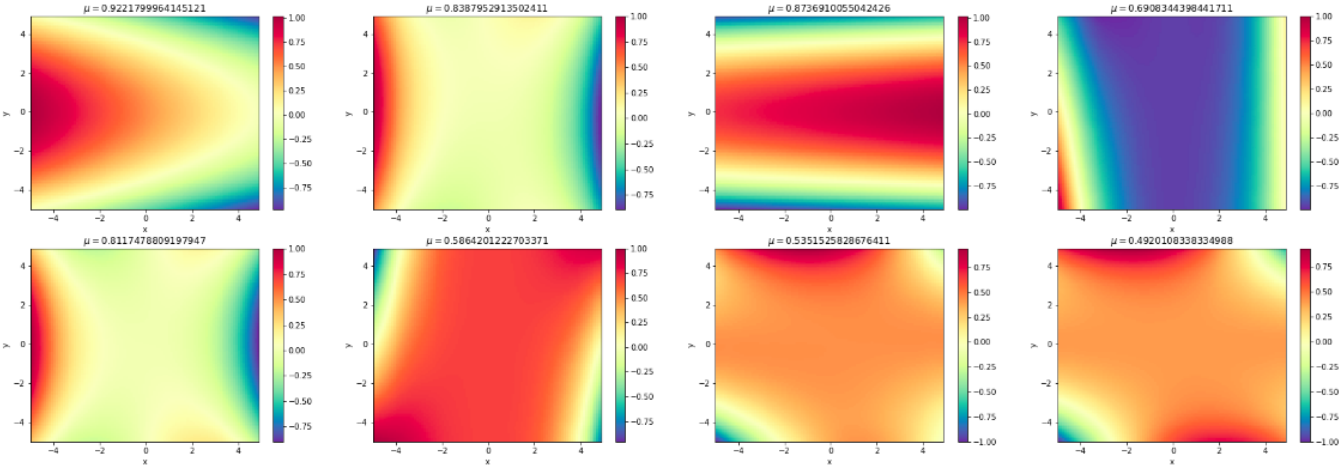

Figure 9: Pseudocolor plots of first 8 eigenfunctions computed using polynomial dictionary along with their corresponding eigenvalues

Figure 10 shows the pseudocolor plots for eigenfunctions with Hermite polynomials of degree 4. Again, the plots obtained are very different from the expected result. The eigenvalues also do not match the expected results. This can be due to the fact that we implemented the EDMD algorithm incorrectly. We tried to detect the error by debugging the code but were unable to do so. We also tried to change the distribution of our initial data points X to a normal distribution, but we did not see any improvement in our results.

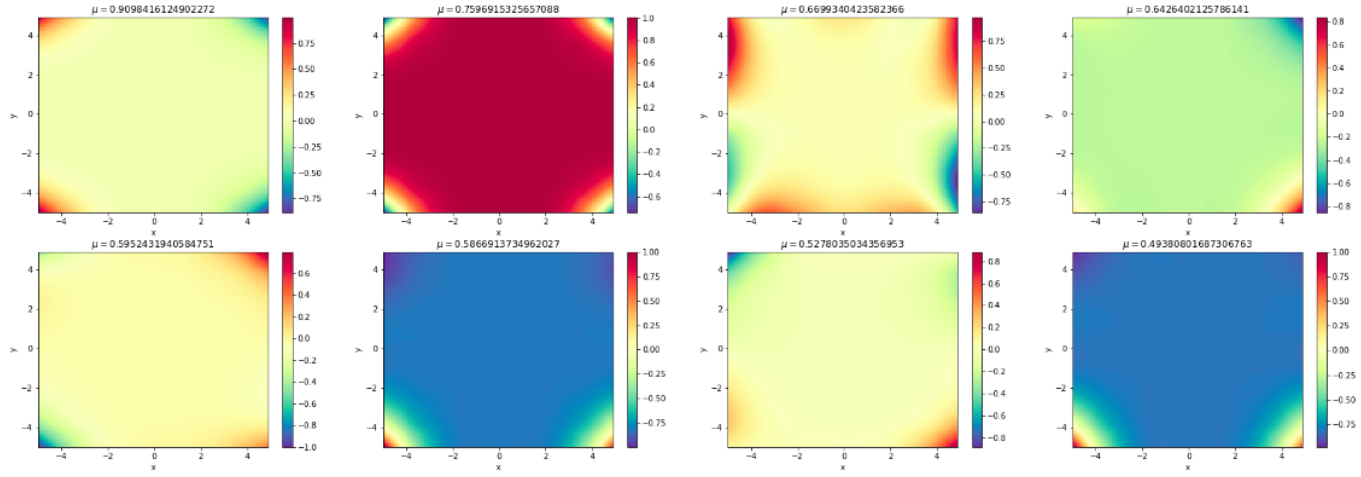

Figure 10: Pseudocolor plots of first 8 eigenfunctions computed using Hermite dictionary along with their corresponding eigenvalues

Testing on Datafold

As our task was to reproduce the example in paper [2], we finally resorted to implementing EDMD implementation of Datafold library. We followed the same steps as before. The code for this is present in LTI_datafold.ipynb

1. **Data Preparation** Instead of preparing matrices of initial and final points, we instead generated a TSCDataFrame object with 100 timeseries data each run for 5 timesteps. The dataframe contains the state of the system at every timestep.

2. **Koopman Implementation** We implemented the EDMD model on the generated TSCDataFrame. We used a polynomial dictionary of degree 4 as this was already supported by the library. Figure 11 shows the measurement space of the data. We plot the predicted values of the next state of the system based on the input of initial state of the system in figure 13. We can see that the lines move in the same direction as expected by the original plot in Figure 6.

	feature	x1	y1	x1^2	x1 y1	y1^2	x1^3	x1^2 y1	x1 y1^2	y1^3	x1^4	x1^3 y1	x1^2 y1^2	x1 y1^3	y1^4
ID	time														
0	0.0	-5.000	-5.000	25.000000	25.000000	25.000000	-125.000000	-125.000000	-125.000000	-125.000000	625.000000	625.000000	625.000000	625.000000	625.000000
	0.1	-4.000	-4.000	16.000000	16.000000	16.000000	-64.000000	-64.000000	-64.000000	-64.000000	256.000000	256.000000	256.000000	256.000000	256.000000
	0.2	-3.200	-3.200	10.240000	10.240000	10.240000	-32.768000	-32.768000	-32.768000	-32.768000	104.857600	104.857600	104.857600	104.857600	104.857600
	0.3	-2.560	-2.560	6.553600	6.553600	6.553600	-16.777216	-16.777216	-16.777216	-16.777216	42.949673	42.949673	42.949673	42.949673	42.949673
	0.4	-2.048	-2.048	4.194304	4.194304	4.194304	-8.589935	-8.589935	-8.589935	-8.589935	17.592186	17.592186	17.592186	17.592186	17.592186
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99	0.0	5.000	5.000	25.000000	25.000000	25.000000	125.000000	125.000000	125.000000	125.000000	625.000000	625.000000	625.000000	625.000000	625.000000
	0.1	4.000	4.000	16.000000	16.000000	16.000000	64.000000	64.000000	64.000000	64.000000	256.000000	256.000000	256.000000	256.000000	256.000000
	0.2	3.200	3.200	10.240000	10.240000	10.240000	32.768000	32.768000	32.768000	32.768000	104.857600	104.857600	104.857600	104.857600	104.857600
	0.3	2.560	2.560	6.553600	6.553600	6.553600	16.777216	16.777216	16.777216	16.777216	42.949673	42.949673	42.949673	42.949673	42.949673
	0.4	2.048	2.048	4.194304	4.194304	4.194304	8.589935	8.589935	8.589935	8.589935	17.592186	17.592186	17.592186	17.592186	17.592186

Figure 11: Ploynomial dictionary of degree 4

	feature	x1	y1
ID	time		
0	0.0	-5.000	-5.000
	0.1	-4.000	-4.000
	0.2	-3.200	-3.200
	0.3	-2.560	-2.560
	0.4	-2.048	-2.048
...	...	...	...
99	0.0	5.000	5.000
	0.1	4.000	4.000
	0.2	3.200	3.200
	0.3	2.560	2.560
	0.4	2.048	2.048

Figure 12: TSC Dataframe for the LTI Example

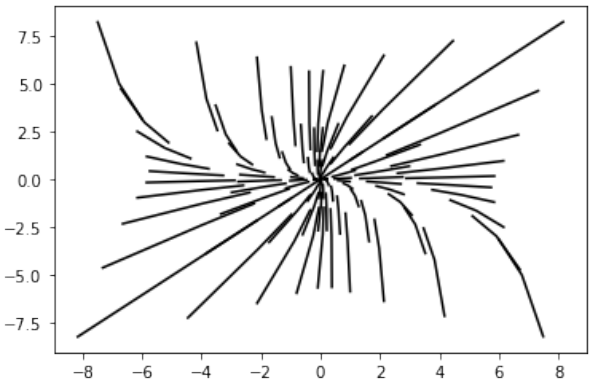

Figure 13: Plot of next state of system

3. **Plotting of the results** To plot the eigenfunctions of the Koopman operator, we first converted the TSCDataFrame to Pandas dataframe and extracted the first 8 non-trivial eigenfunctions. The plots are shown in figure 14. We can see that the plots obtained are exactly the same as the plots for eigenfunctions calculated analytically in figure 7. Infact, the eigenvalues are also accurate to the analytically computed eigenvalues. This level of accuracy between the Koopman approximated results and the analytically calculated values is because EDMD allows us to expand the basis ,unlike DMD, which helps to capture more of the Koopman eigenfunctions. However, these richer set of eigenfunctions have Koopman modes which are zero as shown in fig 15.

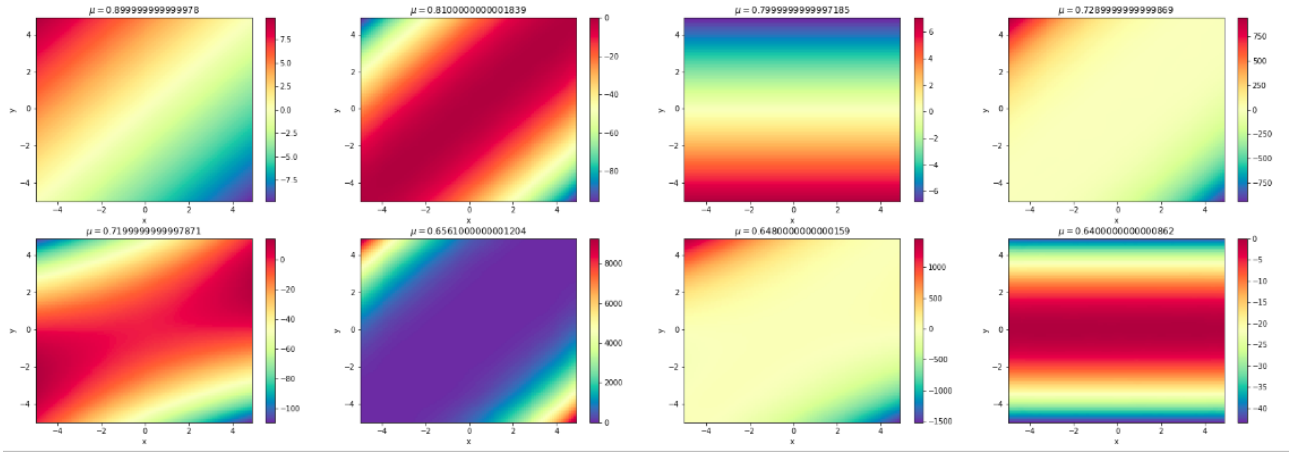

Figure 14: Pseudocolor plots of first 8 eigenfunctions computed using polynomial dictionary in Datafold along with their corresponding eigenvalues

feature	evec0	evec1	evec2	evec3	evec4	evec5	evec6	evec7	evec8	evec9	evec10	evec11	evec12	evec13
x1	-1.000000e+00	-1.396826e-11	-0.707107	-8.350361e-15	-8.929176e-13	9.319589e-15	-3.844393e-15	-1.471522e-13	4.879681e-15	-4.457201e-15	2.905332e-15	6.497884e-15	1.848681e-15	-7.903648e-16
y1	-9.724448e-14	-1.240729e-11	-0.707107	-2.082144e-14	-1.794187e-12	2.881141e-14	-5.998369e-15	-4.734316e-13	2.224424e-14	-1.331713e-14	1.762491e-14	3.355192e-14	1.371280e-14	-7.726391e-15

Figure 15: Koopman modes

Report on task Task 4, Tests on a simple example in crowd dynamics

We began to build the evacuation scenario in Vadere. We considered a 40 meter long and 10 meter wide room, with three obstacles, which leave two gaps for the pedestrians to walk through to the target on the other side of the room. We want to reconstruct this dynamical system with the Koopman operator thus getting a approximator for the center of mass of our pedestrian cloud. We took a look into the Vadere post visualization file, which contains pedestrianId, simTime, endTime-PID1, startX-PID1, startY-PID1, endX-PID1, endY-PID1, targetId-PID2. To train the EDMD algorithm we took the columns: pedestrianId, simTime, endX-PID1, endY-PID1.

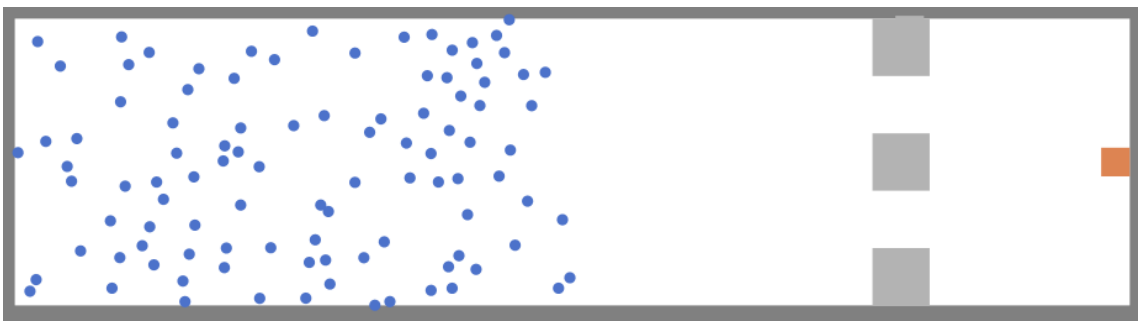

Figure 16: Simple evacuation scenario in Vadere

In jupyter notebook, we started by loading the data with numpy and visualized the pedestrian trajectory.

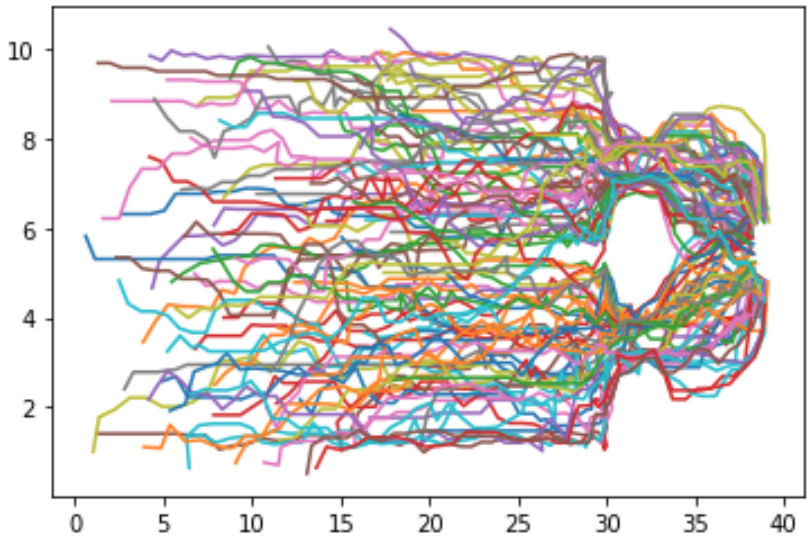

Figure 17: Pedestrian trajectory

The content for this task can be found in the Preprocessing.py file and Task4-Final_submission.ipynb files. For the preprocessing phase, we ordered the pedestrian data, generated unified time stamps, padded the data, ordered again and generated the unified time stamps again. We needed to order the data, because the datafold implementation requires us this format, the same applies to the unified time stamps and the padding. We padded the data set with the last time stamp recorded of the pedestrian, because the missing time stamps occurred because the pedestrian arrived at the target, such that we just assumed he stayed still after reaching the goal. This is also the reason why we applied ordering and generation of time stamps twice, because we first needed to get the element with the highest time stamp, so that we can copy it.

Important to note is, for the code to work, you need to insert a header line. Otherwise the TSCDataFrame.from_csv() method will recognize the first timestep of the first pedestrian as features. This was unfortunately not solvable, because when provided with column names, the argument was provided twice apparently.

At this point we utilized the datafold implementation of the EDMD algorithm. Required for this was the TSCDataFrame, which fits the model. In the feature space we sampled the features, which we called x1 and x2 (x and y coordinate of the pedestrian) to see how each feature influences their end position. Then ultimately we used the predict method to predict the input data.

	feature	x1	x2	x1^2	x1 x2	x2^2	x1^3	x1^2 x2	x1 x2^2	x2^3
ID	time									
1	0.000000	16.726455	9.851365	279.774286	164.778410	97.049392	4679.631915	2756.158609	1623.292263	956.068987
	0.889514	17.275862	9.954393	298.455421	171.970732	99.089949	5156.074780	2970.942692	1711.864326	986.380343
	1.779028	17.899441	9.836961	320.389993	176.076096	96.765793	5734.801825	3151.663718	1732.053608	951.881282
	2.668542	18.574779	9.940189	345.022425	184.636816	98.807356	6408.715384	3429.588095	1835.324835	982.163792
	3.558056	19.248536	9.890215	370.506139	190.372155	97.816348	7131.700777	3664.385283	1882.821493	967.424684
...	...	...	...	...	...	...	...	...	...	...
100	62.265983	38.870050	5.787588	1510.880799	224.963828	33.496173	58728.012424	8744.355274	1301.997905	193.862039
	63.155497	38.870050	5.787588	1510.880799	224.963828	33.496173	58728.012424	8744.355274	1301.997905	193.862039
	64.045011	38.870050	5.787588	1510.880799	224.963828	33.496173	58728.012424	8744.355274	1301.997905	193.862039
	64.934525	38.870050	5.787588	1510.880799	224.963828	33.496173	58728.012424	8744.355274	1301.997905	193.862039
	65.824039	38.870050	5.787588	1510.880799	224.963828	33.496173	58728.012424	8744.355274	1301.997905	193.862039

Figure 18: Feature space analysis

Afterwards we loaded the data with pandas into the TSCDataFrame object and applied the EDMD algorithm with a 3rd order degree polynomial dictionary. This 3rd order degree polynomial dictionary has the advantage compared to the regular DMD model, that we don't directly learn from the state space, such that we can more accurately reconstruct our data. To make a good dictionary choice, expert knowledge is required from the principle equations of the underlying system, but compared to RBF polynomials have the disadvantage that the amount of features needed is rapidely increasing as the input dimension increases. To analyze the dictionary we use the transform method, which applies the dictionary transformation without the prediction. As features we get all the possible combinations of x1 and x2, while staying a third order polynomial [4].

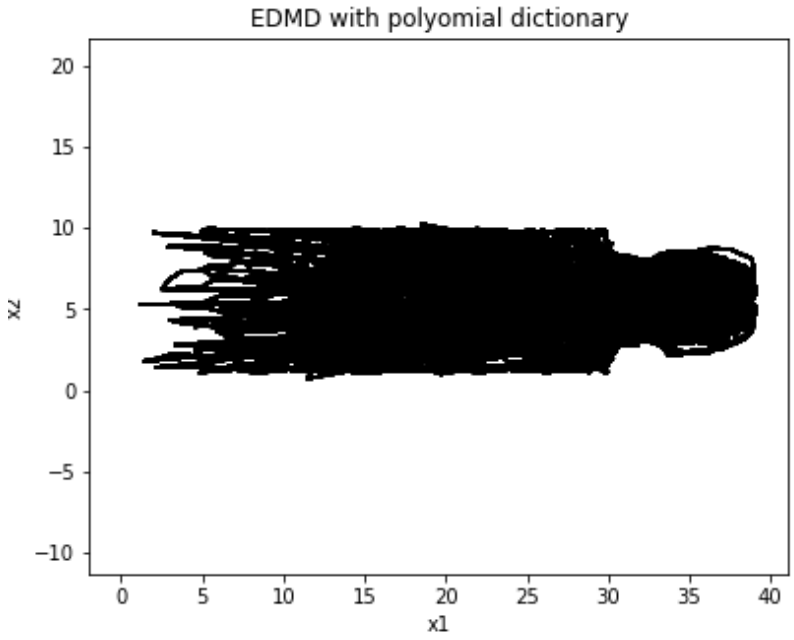

Figure 19: EDMD with 3rd order Polynomial dictionary

Lastly, we applied the EDMD algorithm with a radial basis function dictionary. For this purpose we calculated coefficients for the RBF of each pedestrian. This has the same effect as the polynomials, linearization of the phase space. We chose a Gaussian Kernel with epsilon = 0.17 and set the centre of the functions to the initial condition states, mostly because of the provided tutorial `https://datafold-dev.gitlab.io/datafold/tutorial_06_basic_edmd_limitcycle.html`.

	feature	x1	x2	rbf0	rbf1	rbf2	rbf3	rbf4	rbf5	rbf6	rbf7	...
ID	time											
1	0.000000	16.726455	9.851365	1.000000e+00	3.293434e-44	2.794865e-91	5.493077e-210	8.273701e-143	2.468980e-151	2.520817e-07	2.035935e-112	...
	0.901014	17.275862	9.954393	3.989134e-01	7.167773e-48	1.493739e-96	2.285820e-228	4.842265e-158	4.280839e-167	5.813654e-11	4.036651e-126	...
	1.802028	17.899441	9.836961	1.746831e-02	1.601281e-49	1.552787e-98	2.144799e-247	5.142403e-174	1.427018e-185	1.183010e-15	5.024771e-142	...
	2.703041	18.574779	9.940189	4.228165e-05	8.281300e-56	1.189279e-106	7.687809e-272	1.211721e-194	4.864618e-207	3.242908e-22	6.128313e-161	...
	3.604055	19.248536	9.890215	7.465660e-09	5.367418e-61	1.089981e-112	1.389048e-295	6.460188e-215	2.157218e-229	1.107942e-29	1.367382e-180	...
...	...	...	...	...	...	...	...	...	...	...	...	...
100	63.070966	38.870050	5.787588	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	...
	63.971980	38.870050	5.787588	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	...
	64.872994	38.870050	5.787588	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	...
	65.774008	38.870050	5.787588	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	...
	66.675021	38.870050	5.787588	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	...

Figure 20: Feature space analysis

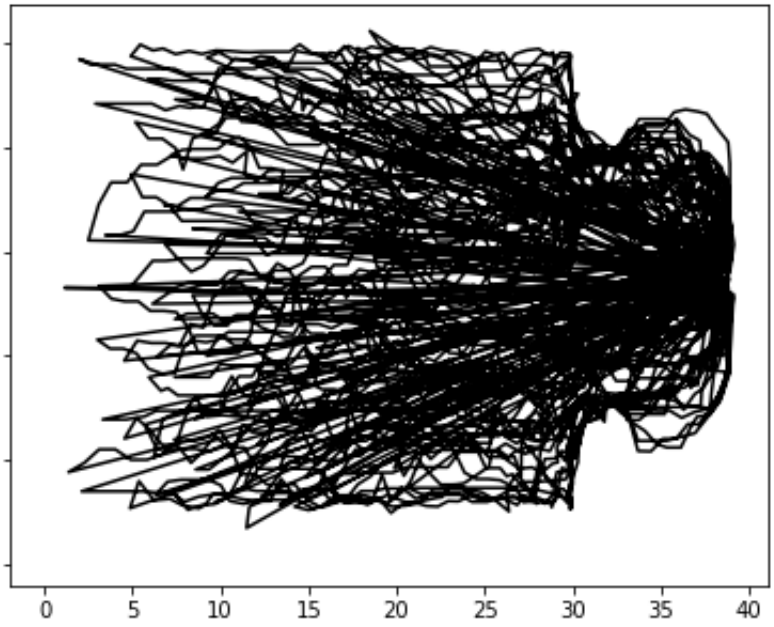

Figure 21: EDMD with Radial basis function dictionary

Report on task Task 5, Discussion of the results

1. In task 1, we presented the reason for studying Koopman operator and it's use in understanding the underlying dynamical system in the measurement space. We also discuss and compare Koopman Operator method with the traditional Poincare Geometric interpretation of studying dynamical systems.

2. In task 2, we present how we implemented the EDMD algorithm and discuss the results obtained on the Hopf ODE system. We chose to test our implementation by comparing the results obtained to the original system. It was interesting to see how the Koopman Operator could approximate the underlying dynamical system by only working in the measurement space. As the results looked promising, we proceeded to test our implementation on the example provided in the paper [2].

3. This was presented in task 3. The main goal of doing this example was to demonstrate that there is good quantitative agreement between the analytically obtained Koopman modes, eigenvalues, and eigenfunctions and the approximations produced by EDMD. However, as our own implementation did not work as expected, we were still successfull in reproducing the results using EDMD implementation of datafold library. Upon testing with different dictionary we realized that the choice of dictionary matters in EDMD implementation as domain on which the underlying dynamical system is defined is not necessarily known. In task 3, we chose a different set of dictionary than suggested by the authors of [2]. However, we were able to get pretty good results with Polynomial dictionary of order 4. task 4 in the polynomial dictionary result. We can see that the pedestrians walked pretty straightforward into the target. It is interesting to note that even thought we changed to chokepoint to two gaps instead of one. The Koopman operator seems to approximate the pedestrians going through the wall/obstacle, because the pedestrians split themselves and the operator is taking the center of mass into account.

4. For the radial basis function dictionary we can see a much more complex pattern, which looks a lot more realistic compared to the polynomial example to our pedestrian trajectory plot.

References

[1] Budišić, Marko, Ryan Mohr, and Igor Mezić. "Applied koopmanism." Chaos: An Interdisciplinary Journal of Nonlinear Science 22, no. 4 (2012): 047510.

[2] Williams, Matthew O., Ioannis G. Kevrekidis, and Clarence W. Rowley. "A data–driven approximation of the koopman operator: Extending dynamic mode decomposition." Journal of Nonlinear Science 25, no. 6 (2015): 1307-1346.

[3] Williams, Matthew O., Clarence W. Rowley, Igor Mezić, and Ioannis G. Kevrekidis. "Data fusion via intrinsic dynamic variables: An application of data-driven Koopman spectral analysis." EPL (Europhysics Letters) 109, no. 4 (2015): 40007.

[4] Lehmberg et al., (2020). datafold: data-driven models for point clouds and time series on manifolds. Journal of Open Source Software, 5(51), 2283,