MODELLING FROM MEASUREMENTS – REPORT

STUDENT: PAOLO CONTI. **LECTURER**: PROF. J. NATHAN KUTZ.

ABSTRACT

The increasing capacity to collect data has led to the enormous success of data-driven approaches, which are characterized by extreme flexibility, generality and non intrusive nature. At the same time, the need of a representative dynamical model of the physical system that generated the data is of paramount importance for providing interpretation to the data, for predicting future states, and, as an ultimate goal, for controlling the physical system itself. In this work we present several techniques for effectively constructing dynamical models directly from measurements.

1 Introduction and Overview

In the present work, we will consider dynamical systems of the following form

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t; \boldsymbol{\mu}), \tag{1}$$

where $\mathbf{x}$ is the state of the system, t the time, $\boldsymbol{\mu}$ the set of parameters, while the $\mathbf{f}$ represents the function that defines the dynamical evolution of the system. Alternatively, we will also consider a more general discrete-time formulation

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k), \tag{2}$$

where $\mathbf{F}$ is the so-called *map*, which takes as input the state of the system and provides as output the state at the next time-step.

Besides providing an interpretation to the collected measurements, a dynamic model allows for prediction of the states in the near future and provides insights for the control of the monitored system.

The goal of this work is to present different strategies to extract a dynamical system representation, in the form of (1) or (2), directly from a set of measurements of the physical system. In particular, we consider the following strategies:

(a) Fitting the function $\mathbf{f}$ in (1) with a linear operator, i.e. a matrix $\mathbf{A}$. We make use of *(optimized) Dynamic Mode Decomposition* (optDMD) algorithm and its variants.

(b) Considering the function $\mathbf{f}$ in (1) as a linear combination of a set of candidate (possibly) nonlinear functions. *Sparse Identification of Nonlinear Dynamics* (SINDy) algorithm and its variants are employed for this task.

(c) Training neural networks in order to approximate the map $\mathbf{F}$ in (2).

Moreover, we make use of techniques such as *Singular Value Decomposition* (SVD) or time-delay embedding to either efficiently reduce the dimensionality of the data or to increase it (e.g. to investigate the presence of latent variables); as well as to transform the states into a new coordinates which are possibly better suited to identify the underlying dynamics.

2 Theoretical Background

In this section, we briefly present the theoretical concepts behind the aforementioned strategies (a)-(b)-(c).

2.1 Singular Value Decomposition (SVD)

A matrix $\mathbf{X} \in \mathbb{C}^{n\times m}$ may be decomposed as

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^*, \tag{3}$$

where $\mathbf{U} \in \mathbb{C}^{n\times n}$ and $\mathbf{V} \in \mathbb{C}^{m\times m}$ are *unitary* matrices, namely $\mathbf{U}\mathbf{U}^* = \mathbf{U}^*\mathbf{U} = \mathbf{I}_{n\times n}$ and $\mathbf{V}\mathbf{V}^* = \mathbf{V}^*\mathbf{V} = \mathbf{I}_{m\times m}$, while $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times m}$ is a diagonal matrix which contains decreasing, non-negative entries called *singular values*. Here, * denotes complex conjugate transpose. Moreover $\mathbf{X}$ might be approximated as

$$\mathbf{X} \approx \tilde{\mathbf{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\mathbf{V}}^*, \tag{4}$$

where $\tilde{\mathbf{U}}$ (resp. $\tilde{\mathbf{V}}$) contain the first $r \ll n$ columns of $\mathbf{U}$ (resp. $\mathbf{V}$), and $\tilde{\mathbf{\Sigma}}$ contains the first $r \times r$ block of $\mathbf{\Sigma}$.

2.2 Time-delay embedding

Time-delay embedding is a technique to increase the dimensionality of the state space, by stacking together different delayed copies of the state vector $\mathbf{x}$, forming the so-called Hankel matrix. For instance, for a one-dimensional state, the k time-delay embedding Hankel matrix is

$$\mathbf{H} = \begin{bmatrix} x\left(t_1\right) & x\left(t_2\right) & \dots & x\left(t_p\right) \\ x\left(t_2\right) & x\left(t_3\right) & \dots & x\left(t_{p+1}\right) \\ \vdots & \vdots & \ddots & \vdots \\ x\left(t_k\right) & x\left(t_{k+1}\right) & \cdots & x\left(t_m\right) \end{bmatrix},$$

where $p = m - k + 1$. Then SVD is applied to $\mathbf{H}$ and it allows to gain more insight into the real rank of the system, e.g. to discover the presence of latent variables. The rank of the system is obtained by looking at the energy content of each of the singular values.

2.3 Linear dynamics – (Optimized) Dynamic Mode Decomposition (optDMD)

Linear dynamical systems are highly desirable because they dispose of a wide range of techniques for analysis, prediction and control of such systems. A linear system can be written in the form of (1), by considering a matrix $\mathbf{A}$ as $\mathbf{f}$, i.e.

$$\frac{d}{dt}\mathbf{x} = \mathbf{f}(\mathbf{x}) \approx \mathbf{A}\mathbf{x}. \tag{5}$$

The biggest advantage is that (5) admits a closed-form solution given by $\mathbf{x}(t_0 + t) = e^{\mathbf{A}t}\mathbf{x}(t_0)$, hence, the eigenvalues and eigenvectors of $\mathbf{A}$ entirely characterize the dynamics.

DMD technique allows to identify the eigenvalues and eigenvectors of the best-fit linear operator $\mathbf{A}$ directly from the data. Actually, instead of computing the spectral decomposition of $\mathbf{A}$, DMD algorithm computes the one of $\tilde{\mathbf{A}}$, being the latter the projection of $\mathbf{A}$ onto the leading spatial modes identified by applying SVD to $\mathbf{X}$. We refer to [2] for more details. Once computed the spectral expansion, the state solutions can be written as

$$\mathbf{x}(t) = \mathbf{\Phi}\exp(\mathbf{\Omega}t)\mathbf{b}, \tag{6}$$

where $\mathbf{\Omega}$ is the diagonal matrix containing the continuous-time eigenvalues, $\mathbf{\Phi}$ is the eigenvector matrix and $\mathbf{b} = \mathbf{\Phi}^{\dagger}\mathbf{x}_1$ contains the mode amplitudes. The symbol † denotes the pseudo-inverse.

In the present work we employ optDMD algorithm [1], which avoids the spectral decomposition and it directly solves the exponential fitting problem

$$\underset{\hat{\mathbf{\Omega}},\hat{\mathbf{\Phi}},\hat{\mathbf{b}}}{\arg\min}\, ||\mathbf{X} - \hat{\mathbf{\Phi}}\exp\Big(\hat{\mathbf{\Omega}}t\Big)\hat{\mathbf{b}}||_{\mathrm{F}},$$

which is more stable with respect to classical DMD.

2.4 Nonlinear dynamics – Sparse Identification of Nonlinear Dynamics (SINDy)

The SINDy algorithm [3] identifies nonlinear dynamical systems directly from data, relying on the assumption that typically just few active terms appear in the dynamics $\mathbf{f}$ in equation (1). A library of candidate linear and nonlinear terms is chosen and the few active terms are identified by SINDy through sparse regression. Specifically, we aim to approximate $\mathbf{f}$ as follows

$$\dot{\mathbf{X}} = \mathbf{f}(\mathbf{X}) \approx \mathbf{\Theta}(\mathbf{X})\mathbf{\Xi} \tag{7}$$

where $\mathbf{X} = [\mathbf{x}_1 \,|\, ... \,|\, \mathbf{x}_m]$ is the snapshot matrix containing the measurement of the n states $\mathbf{x}_t \in \mathbb{C}^n$ for all the time instants $t = 1, ..., m$, while the matrix $\dot{\mathbf{X}} = [\dot{\mathbf{x}}_1 \,|\, ... \,|\, \dot{\mathbf{x}}_m]$ contains the time derivatives of the states. Here, $\mathbf{\Theta}(\mathbf{X})$ is the library of candidate functions that might describe the dynamics of the data, e.g. , $\mathbf{\Theta}(\mathbf{X}) = \big[\mathbf{1} \,|\, \mathbf{X} \,|\, \mathbf{X^2} \,|\, ... \,|\, \sin(\mathbf{X}) \,|\, ...\big]$, while $\mathbf{\Xi}$ is the unknown matrix of coefficients that may be estimated by sparse regression

$$\underset{\hat{\mathbf{\Xi}}}{\arg\min}\, ||\dot{\mathbf{X}} - \mathbf{\Theta}(\mathbf{X})\hat{\mathbf{\Xi}}||_2^2 + R(\hat{\mathbf{\Xi}}),$$

where R is regularizer chosen to promote sparsity in $\mathbf{\Xi}$.

2.5 Neural Networks (NNs) for the estimation of the map F

Neural networks (NNs) are state-of-the-art techniques for approximating highly nonlinear functions. By making use of large datasets, a huge number of NN parameters is trained and this results in extremely flexible NN models.

In the current work, we consider NNs to approximate the dynamical map $\mathbf{F}$ in (2). Namely we aim to create a NN model $\mathbf{f}_{\texttt{NN}}$ such that

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k) \approx \mathbf{f}_{\texttt{NN}}(\mathbf{x}_k; \mathbf{W}), \tag{8}$$

where $\mathbf{W}$ are the NN weights.

3 Algorithm Implementation and Development

In this section we present the algorithms developed in the current work for the implementation of the techniques presented in the previous section and further improvement strategies.

3.1 Bagging optimized DMD (bopDMD) and Ensemble SINDy (E-SINDy)

For the implementation of the optDMD and SINDy algorithms we rely respectively on the `optDMD` [1] and `pySINDy` [4] packages. Moreover, we combine optDMD and SINDy with the the statistical method of bootstrap aggregating which provides greater stability and robustness when dealing with noise-compromised data. The resulting algorithms are denoted as BOP-DMD (bagging optimized DMD) [5] and E-SINDy (Ensemble SINDy) [4] and they are are outlined in Alg. 1 and 2, respectively.

Algorithm 1 BOP-DMD

Require: $\mathbf{X}, p, K$.
  for $k = 1, ..., K$
    Select p of m snapshots ($p < m$), obtaining $\mathbf{X}_k$.
    Compute $\boldsymbol{\Omega}_k, \boldsymbol{\Phi}_k, \mathbf{b}_k$ by optDMD.
    if not diverging
      Compute the trajectories $\hat{\mathbf{X}}_k$.
  Compute the means $\boldsymbol{\Omega} = \langle \boldsymbol{\Omega}_k \rangle, \boldsymbol{\Phi} = \langle \boldsymbol{\Phi}_k \rangle, \mathbf{b} = \langle \mathbf{b}_k \rangle$.
  Compute mean and std of traj. $\boldsymbol{\mu} = \langle \hat{\mathbf{X}}_k \rangle, \boldsymbol{\sigma} = \sqrt{\langle \hat{\mathbf{X}}_k^2 \rangle}$.
  return $\boldsymbol{\Omega}, \boldsymbol{\Phi}, \mathbf{b}, \boldsymbol{\mu}, \boldsymbol{\sigma}$.

Algorithm 2 E-SINDy

Require: $\mathbf{X}, p, K$.
  for $k = 1, ..., K$
    Select p of m snapshots ($p < m$), obtaining $\mathbf{X}_k$.
    Compute $\boldsymbol{\Xi}_k$ by solving $\dot{\mathbf{X}}_k = \boldsymbol{\Theta}(\mathbf{X}_k)\boldsymbol{\Xi}_k$ with SINDy.
    if not diverging
      Compute the trajectories $\hat{\mathbf{X}}_k$.
  Compute the mean of coefficients $\boldsymbol{\Xi} = \langle \boldsymbol{\Xi}_k \rangle$.
  Compute mean and std of traj. $\boldsymbol{\mu} = \langle \hat{\mathbf{X}}_k \rangle, \boldsymbol{\sigma} = \sqrt{\langle \hat{\mathbf{X}}_k^2 \rangle}$.
  return $\boldsymbol{\Xi}, \boldsymbol{\mu}, \boldsymbol{\sigma}$.

3.2 Estimation of F with neural networks.

Here, we present how to train the NN model $\mathbf{f}_{\texttt{NN}}$ to approximate the dynamical map $\mathbf{F}$, as in (8), and how to test $\mathbf{f}_{\texttt{NN}}$ and use it compute the trajectories.

Training A NN architecture is built, then trained in a purely black-box way, i.e. by providing $\mathbf{x}_k$ as input and $\mathbf{x}_{k+1}$ as output, for all the time instants $k = 1, ..., m-1$. A mean squared error loss which measures the mismatch between the NN output $\mathbf{f}_{\texttt{NN}}(\mathbf{x}_k; \mathbf{W})$ and $\mathbf{x}_{k+1}$ is minimized by a gradient-based optimization technique and the NN weights $\mathbf{W}$ are then learned by backpropagation.

Testing and trajectories estimation Once we trained our model, we can use it to advance the states from t to $t + \Delta t$ and, consequently, to compute the trajectories. Specifically, we can follow two strategies:

(I) For all the time steps $t = 1, ..., m-1$ we estimate $\mathbf{x}_{t+1}$ with $f_{\texttt{NN}}(\mathbf{x}_t)$. Thus, for each time-step we provide as input the real state data.

(II) We employ the NN predicted solution at previous step as input for the prediction of the solution at the next step. Hence, except for the first step, for which we provide the correct initial condition $\mathbf{x}_1$ to compute $f_{\texttt{NN}}(\mathbf{x}_1) \approx \mathbf{x}_2$, for all other states, the input value is recursively computed as NN prediction at the previous time-step, e.g., $f_{\texttt{NN}}(f_{\texttt{NN}}(\mathbf{x}_1)) \approx \mathbf{x}_3$.

Although the approach (I) is more straightforward, the more likely and frequent context is the one identified by the (II) scenario. In fact, when we aim to calculate trajectories of a solution, the data we have at disposal is typically just the initial condition.

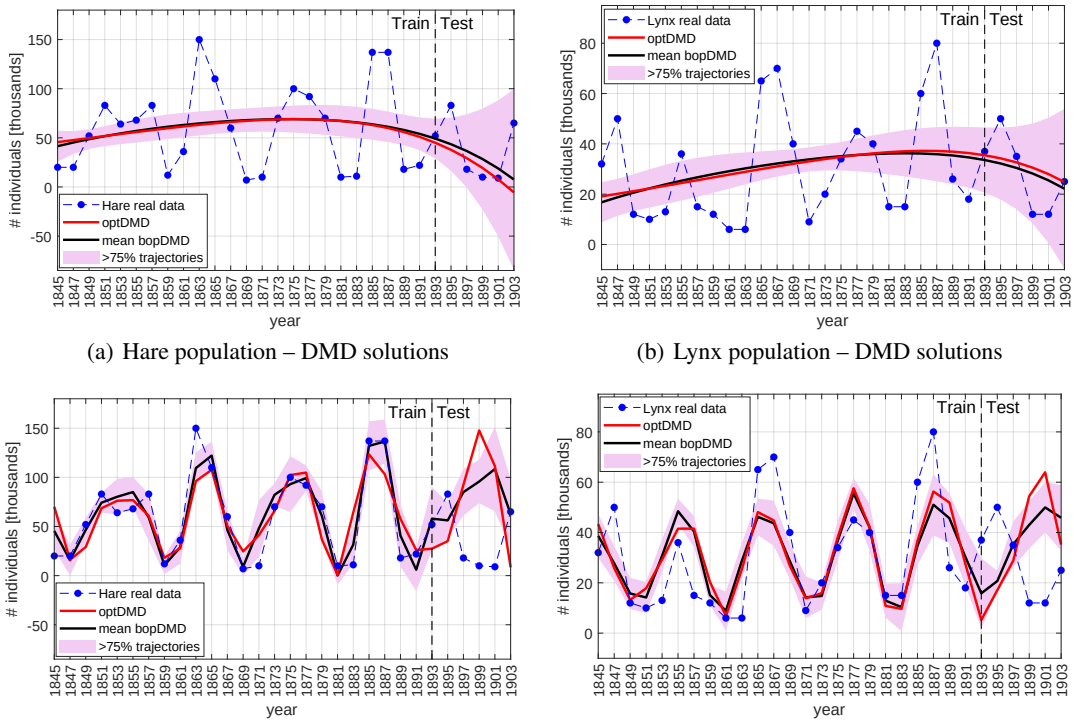
4 Computational Results

4.1 Dyamics of Lynx-Hare populations

In this section we aim to find a dynamical description for the evolution of the Canadian lynx and snowshoe hare populations, employing some of the proposed methods. We have at disposal a dataset about the number of individuals of the two species from 1845 to 1903. In addition, we are interested in testing the capability of the identified models to forecast future population states. To this end, we divide the dataset into *training set* (data up to year 1893) and *testing set* (data after year 1893). The latter is used as reference to compare the accuracy of the future states predicted by the models. In the following we present the obtained results.

4.1.1 DMD and time-delay DMD models

As first attempt, we employ DMD models – specifically, optDMD and bopDMD (see Sect. 3.1) – on the raw dataset to identify the population dynamics and to predict the population states. In Fig. 1(a)-1(b) we illustrate the obtained results. The DMD with two time series can only provide us with two eigenvalues, which results in an estimated solution with at most one single oscillating frequency – that is indeed what we obtained. In order to have more frequencies in the DMD solution we perform time-delay embedding (see Sect. 2.2) in order to increase the dimensionality of the dataset and we select the first 5 variables. Analogously, we apply DMD models on these new variables. Results are presented in Fig. 1(c)-1(d). The dramatic improvement in the prediction quality using time-delay embedding is strongly suggesting the presence of latent variables.



(a) Hare population – DMD solutions

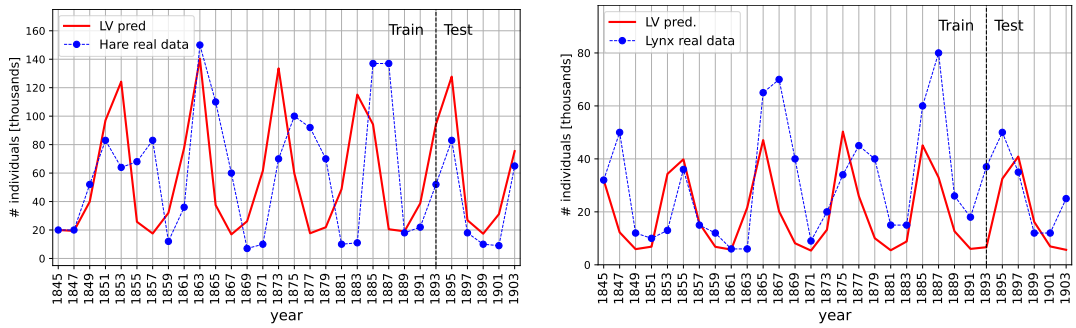
(b) Lynx population – DMD solutions

(c) Hare population – time-delay DMD solutions

(d) Lynx population – time-delay DMD solutions

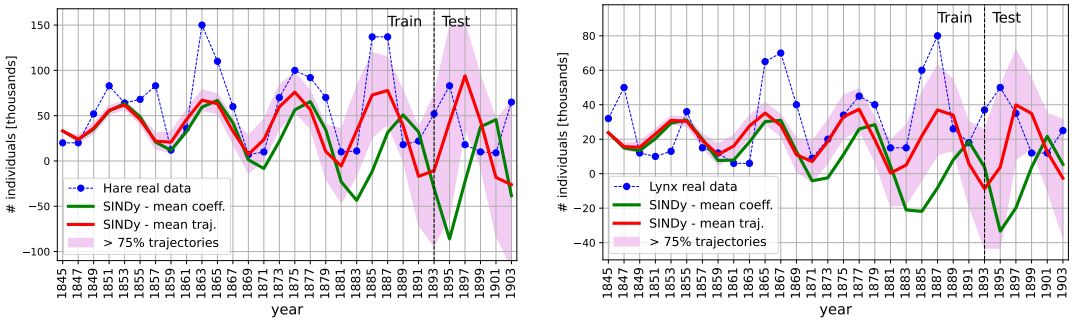
Figure 1: Predicted solutions employing optDMD and bopDMD which are fitted on the training set, while tested on the overall time-window. The continuous black line, labeled as *mean bopDMD*, represents the average over all trajectories computed by bopDMD. The bopDMD algorithm naturally provides uncertainty estimates, thus we exploit them to highlight (in pink) the region in which more than 75% of trajectories fall[1]. We note an amplification of the uncertainty and a worse accuracy outside the training region. Overall, bopDMD with time-embedding provides the most accurate dynamical description of the data.

[1]By Chebyshev's inequality the probability that a trajectory lie outside the region $\mathcal{D}_k = (\boldsymbol{\mu} - k\boldsymbol{\sigma}, \boldsymbol{\mu} + k\boldsymbol{\sigma})$ does not exceed $1/k^2$, where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean value and the standard deviation calculated over all trajectories obtained with bopDMD. By taking $k = 2$ we have that a trajectory lies outside $\mathcal{D}_2$ with probability smaller than 0.25, thus statistically more than 75% fall in $\mathcal{D}_2$.



(a) Hare population – LV solution

(b) Lynx population – LV solution



(c) Hare population – E-SINDy solutions

(d) Lynx population – E-SINDy solutions

Figure 2: Predicted solutions with E-SINDy employing a LV (top) and a fourth-degree polynomial (bottom) models. For the latter, the green and red line represents the trajectory obtained by integrating the system whose coefficients are the mean over all E-SINDy coefficients and the mean over all trajectories (namely $\boldsymbol{\Xi}$ and $\boldsymbol{\mu}$ in Alg. 2 of Sect. 3.1), respectively.

4.1.2 Nonlinear modelling with SINDy

Next, we move from fitting a linear model to find a nonlinear dynamical representation employing E-SINDy (Sect. 3.1). First, a Lotka-Volterra (LV) predator-prey model of the form $\dot{x} = (b - py)x$ and $\dot{y} = (rx - d)y$ is considered, where x and y represent the numbers of individuals of the hare and lynx populations, respectively. We train the SINDy model $[\dot{\mathbf{x}} \,|\, \dot{\mathbf{y}}] = \boldsymbol{\Theta}(\mathbf{x}, \mathbf{y})\boldsymbol{\Xi}$ with a library of functions $\boldsymbol{\Theta}(\mathbf{x}, \mathbf{y}) = [\mathbf{x} \,|\, \mathbf{y} \,|\, \mathbf{xy}]$ in order to estimate the unknown coefficients b, p, r and d, which are some entries of $\boldsymbol{\Xi}$ (while we constrain to zero the remaining entries). The estimated values are $b = 36.01, p = 1.79, r = 0.69, d = 39.98$ and the corresponding solutions are illustrated in Fig. 2(a)-2(b).

Next, motivated by the results of the previous section which suggest the presence of latent variables, we try to fit a SINDy model on the time-delay embedded coordinates. We consider only the first 3 variables to have a parsimonious number of equations. A polynomial library up to the fourth order is considered. Results are depicted in Fig. 2(c)- 2(d).

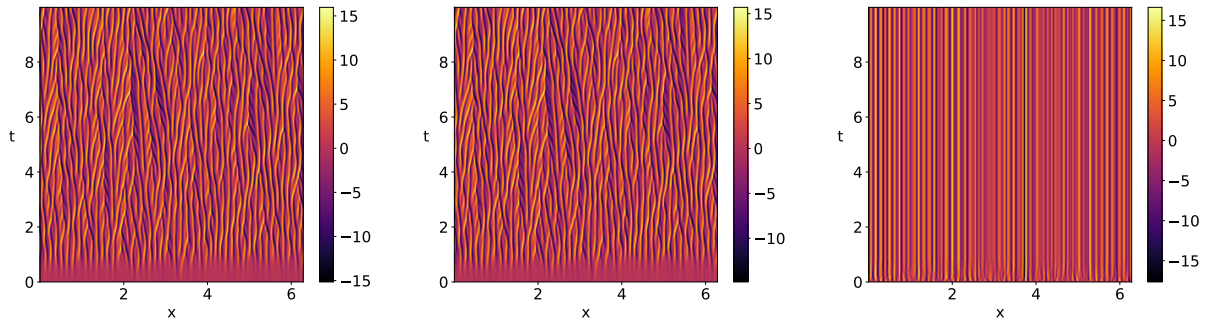
Although the fourth-order SINDy model seems to more accurately trace the correct oscillation frequencies of the evolution of the two populations, as time increases the uncertainty becomes larger and larger and, especially, negative values are estimated for the numerosity of the populations, that have no physical meaning. The LV model, although simpler, seems the most accurate, along with optDMD which, however, is less reliable since it presents a large error in the testing time-window.

4.2 Kuramoto–Sivashinsky equation

In the following exercise we consider the 1D Kuramoto–Sivashinsky (KS) equation

$$u_t = -uu_x - u_{xx} - u_{xxxx}, \tag{9}$$

which is a fourth-order nonlinear partial differential equation that presents a chaotic behavior. The solution of this equation presents a peculiar – as well as fascinating – trend, in fact, as time passes, stripes appear and merge, but they never disappear or split (see, e.g., Fig. 3(a)). Regarding this example, our goal is to test the performance of neural networks in estimating the temporal evolution of this type of dynamical system.



(a) Numerical solution 1 (b) NN output – approach (I) (c) NN output – approach (II).

Figure 3: KS numerical solution 1 (left) and predicted solutions obtained with approaches (I) (center) and (II) (right), starting from the same initial condition and employing the same NN model.

Dataset A numerical approximation of the solution to (9) is generated employing the provided ODE time-stepper `KuraSiva.ipynb` (see Appendix A). We consider $n = 2048$ spatial degrees of freedom (equispaced in $[0, 2\pi]$) and a normal sampled initial condition. KS equation (9) is integrated starting from $t_0 = 0$ up to a final time $T = 10$ with a time step $\Delta t = 0.01$ (for a total number $m = 1001$ of time-steps) and the approximated values of the solution are stored in a matrix $\mathbf{U} \in \mathbb{R}^{n \times m}$. In Fig. 3(a) we illustrate the numerical solutions obtained from a randomly sampled initial condition (denoted as numerical solution 1).

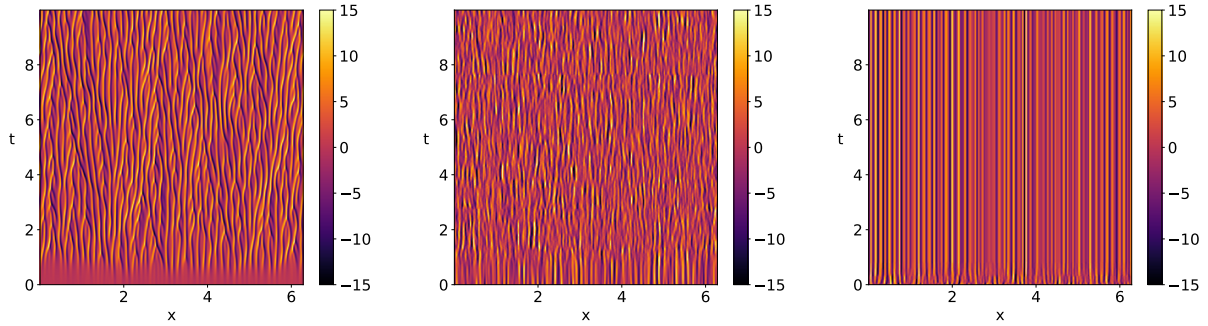
4.2.1 Advancement of the solution from t to $t + \Delta t$

As a first analysis, we investigate if a NN model is able to advance the solution of equation (9) from t to $t + \Delta t$. Namely we aim to create a NN model $f_{\texttt{NN}}$ such that

$$f_{\texttt{NN}}(u(\mathbf{x}, t)) \approx u(\mathbf{x}, t + \Delta t).$$

To this end, we train a NN model and we predict the trajectories following strategies (I) and (II) presented in Sect.3.2.

In Fig. 3, in addition to the numerical solution, we illustrate the NN outputs obtained from using strategies (I) and (II). We notice that by providing the numerical solution as input for testing ((I) strategy), the NN model achieves great accuracy – indeed, the predicted solution of the NN (3(b)) is extremely close to the numerical one (3(a)). While, by recursively relying on the previously predicted solution ((II) strategy), we have a quick and irretrievable deterioration in the quality of prediction (3(c)). This suggests that the NN model, although extremely accurate when fed with accurate inputs, it is not very robust and rather sensitive to even small noise on the input data. Hence it does not allow accurate tracking of the solution trajectory. To get a better understanding about the predictive ability of trajectories, in the following we test our model by considering data coming from different initial conditions.



(a) Numerical solution 2 (b) NN output – approach (I) (c) NN output – approach (II)

Figure 4: KS numerical solution 2 (left) and predicted solutions obtained with approaches (I) (center) and (II) (right), starting from a newly sampled initial condition and employing the pre-trained NN model.

4.2.2 Comparison of the evolution trajectories for the NN model with different initial conditions

Using the previously trained NN model, we test the quality of the prediction for a new initial condition, denoted as numerical solution 2.

In Fig. 4, we depict the newly computed numerical solution and the corresponding predictions with approaches (I) and (II). Differently from the previous section, we note a degradation of the prediction accuracy also for the approach (I). This indicates that the NN model, in addition to being extremely sensitive to perturbations on the input, it is strongly dependent on the initialization, thus it is unable to generalize and it is more likely to be overfitting the training set rather than actually learning the dynamical model underlying the data.

4.2.3 NN forecast of future states

As a further test, we test the ability of the network to forecast future states. To this end, we re-train the NN model with training data consisting of numerical solution 1, but only up to time $\tilde{T} = 5$. Once the NN is trained, following the approach (I), we compute the trajectories up to the final time $T = 10$, thus predicting ahead of the training time $\tilde{T}$, starting from the same initial condition as the one used during training.

Results are illustrated in Fig. 5. We note how the quality of the prediction degenerates once the training time-window is exceeded. This is further evidence that the NN model considered is not robust and it is overfitting the training dataset.

4.3 Reaction-diffusion problem

Next we consider a lambda-omega reaction-diffusion system which is governed by the following equations

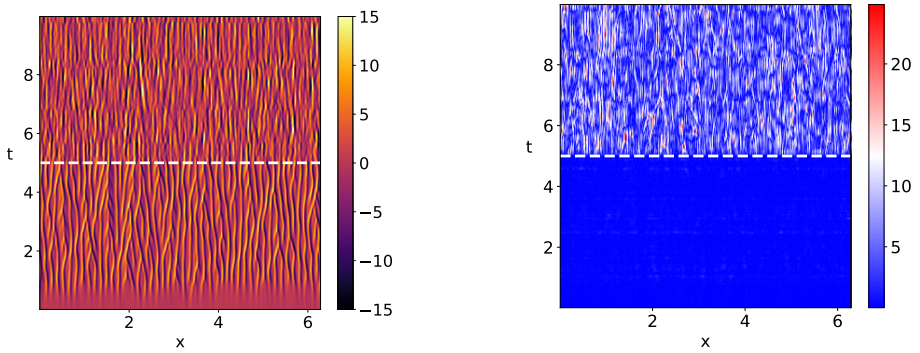
$$\begin{aligned} \dot{u} &= \left(1 - \left(u^2 + v^2\right)\right) u + \beta \left(u^2 + v^2\right) v + d_1 \left(u_{xx} + u_{yy}\right), \qquad t \in (0, T),\ x, y \in [-L, L], \\ \dot{v} &= -\beta \left(u^2 + v^2\right) u + \left(1 - \left(u^2 + v^2\right)\right) v + d_2 \left(v_{xx} + v_{yy}\right), \qquad t \in (0, T),\ x, y \in [-L, L], \end{aligned} \tag{10}$$

where $d_1 = 0.1, d_2 = 0.1, \beta = 1$ and $L = 10$.

Dataset A numerical approximation of the solution to (10) is generated employing the provided numerical solver `ReactionDiffusion.ipynb` (see Appendix A). Solutions are generated over an equispaced spatial grid on $\Omega = [-L, L]^2$, consisting of $n = 512$ points in each direction. The values of u, v are computed up to a final time $T = 10$ with a time step $\Delta t = 0.05$ (for a total number $m = 201$ of time-steps) and stored respectively in $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n \times n \times m}$.

4.3.1 Advancement of the solution from t to $t + \Delta t$ in low-dimensional SVD subspace

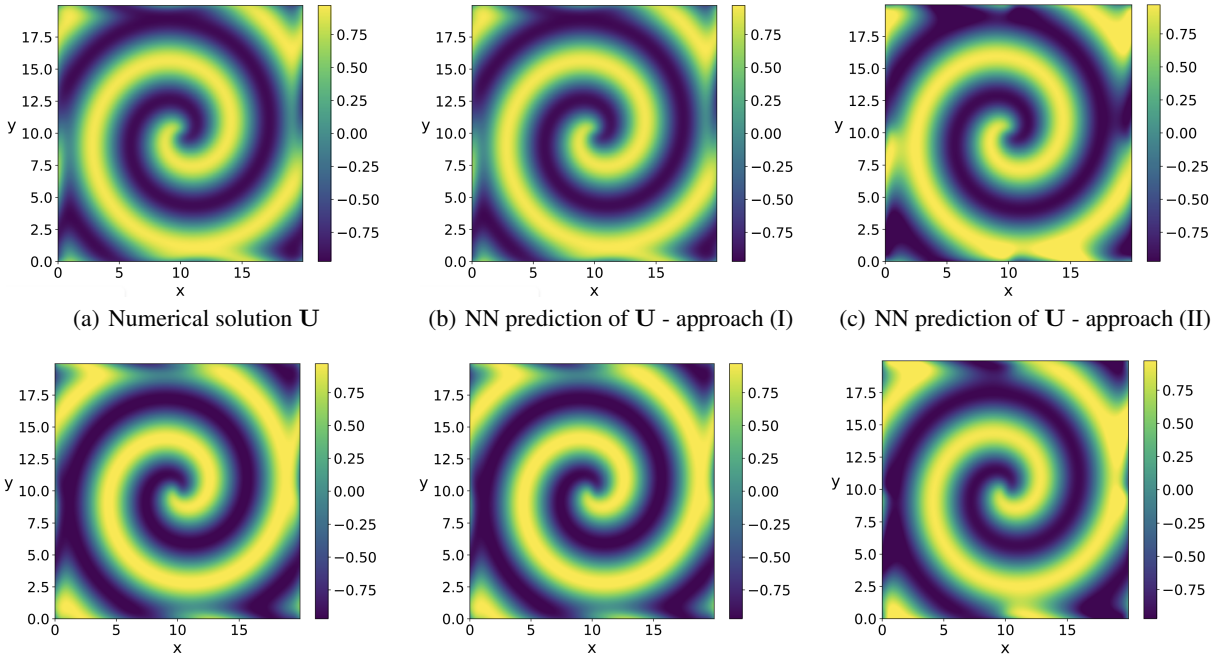
Similarly to the previous example, we investigate if a NN model is able to advance the solution of the equation (10) from t to $t + \Delta t$. In this case, instead of working with the original high-dimensional data, we reduce the dimensionality of the problem by applying SVD (see Sect. .2.1) to $\mathbf{U}$, $\mathbf{V}$, separately. Then, we feed the NN model with new variables



(a) NN forecast – approach (I)

(b) Absolute error between the NN output and the numerical solution 1

Figure 5: KS predicted solutions by NN model (left) and absolute error with respect to numerical solution (right). The training range consists in the time-window below the white line at $\tilde{T} = 5$. We note a macroscopic degradation in the quality of the prediction (left) as well as a dramatic increase in absolute error (right) for $t > \tilde{T}$.



(a) Numerical solution $\mathbf{U}$ (b) NN prediction of $\mathbf{U}$ - approach (I) (c) NN prediction of $\mathbf{U}$ - approach (II)

(d) Numerical solution $\mathbf{V}$ (e) NN prediction of $\mathbf{V}$ - approach (I) (f) NN prediction of $\mathbf{V}$ - approach (II)

Figure 6: Numerical approximation of spatially distributed solutions u and v of the reaction-diffusion problem (left) and predicted solutions obtained with approaches (I) (center) and (II) (right) at time instant $t = 6$.

$\mathbf{u}_{\text{SVD}}, \mathbf{v}_{\text{SVD}}$ consisting in the the projections of $\mathbf{U}$ (resp. of $\mathbf{V}$) onto the first 64 spatial modes, for a total of $n_{\text{SVD}} = 128$ spatial degrees of freedom. Therefore we aim to create a NN model $f_{\texttt{NN}}$ such that

$$f_{\texttt{NN}}(\mathbf{z}(t)) \approx \mathbf{z}(t + \Delta t),$$

where $\mathbf{z}(t) = [\mathbf{u}_{\text{SVD}}(t) \,|\, \mathbf{v}_{\text{SVD}}(t)]$. We train the NN model relying on (I) and (II) strategies (Sect. 3.2). Then we project the NN output back to the high-dimensional physical space through the spatial SVD-modes.

In Fig. 6 we illustrate the comparison between the NN predictions and the numerical solutions $\mathbf{U}$ and $\mathbf{V}$ at the time instant $t = 6$. Both approaches (I) and (II) allow to accurately reconstruct the physics of the problem, even though approach (II) is less precise than (I) in the regions close to the boundaries of the spatial domain.

The fact that the prediction accuracy of NN-based models is better for this example, which has a not chaotical behaviour and a simpler dynamics than the one of the previous problem, it suggests that performance of the presented strategy strongly depends on the complexity of the system.

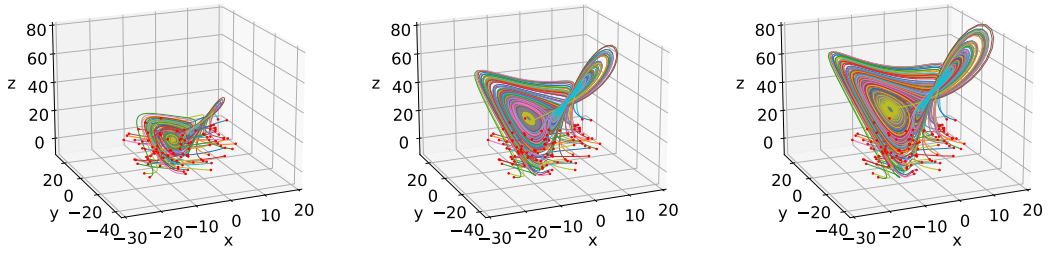
4.4 Chaotic Lorenz system

In this section we consider the parametrized Lorenz system consisting in the following equations

$$\begin{cases} \dot{x} = \sigma(y - x), & t \in (0, T), \\ \dot{y} = x(\rho - z) - y, & t \in (0, T), \\ \dot{z} = xy - \beta z, & t \in (0, T), \end{cases} \tag{11}$$

where $T = 8$, $\sigma = 10$, $\beta = \frac{8}{3}$, while ρ is a parameter. Lorenz equations (11) give rise to rich and chaotic dynamics that evolve on a two lobes attractor. From the Fig. 7, we notice that as the parameter ρ increases, the position of the attractors changes and the size of the lobes increases.

Dataset A numerical approximation of the solution to (11) is generated employing `odeint` numerical integrator on Python. We consider $n_{\text{IC}} = 200$ normal sampled initial conditions and three values of the parameter $\rho = 10, 28$ and 35, for the training set. For each parameter instance, Lorenz equations (11) are integrated starting from $t_0 = 0$ up to a final time $T = 8$ with a time step $\Delta t = 0.01$.



(a) $\rho = 10$ (b) $\rho = 28$ (c) $\rho = 35$

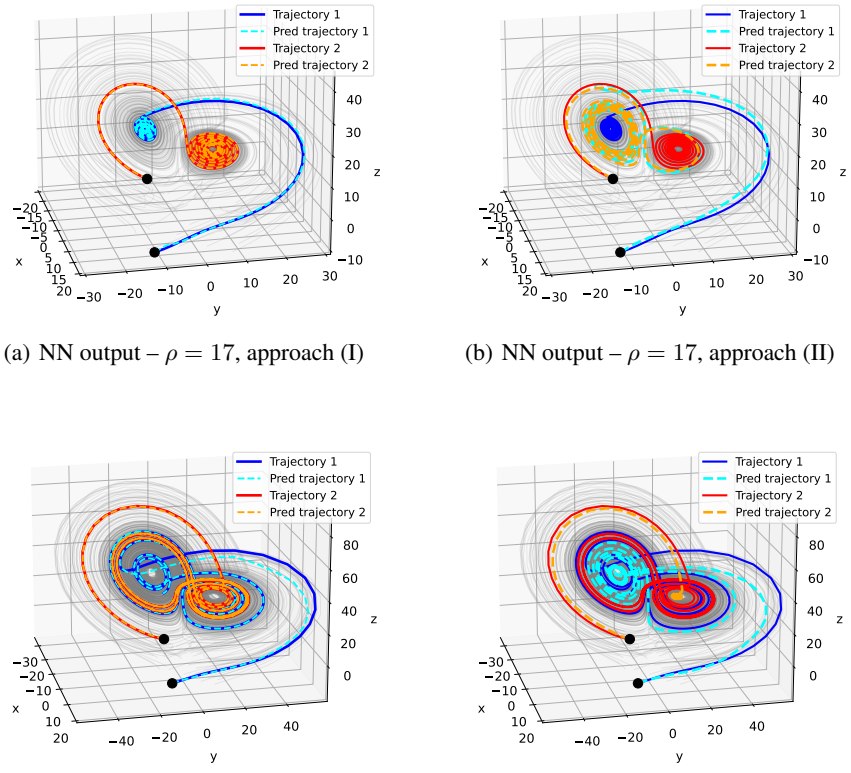
Figure 7: Solution trajectories computed by numerical integration of system (11) for 100 randomly sampled initial conditions (red dots), considering different values for the parameter ρ.

4.4.1 Advancement of the solution from t to $t + \Delta t$ with parametric dependence

Also in this example, we investigate if a NN model is able to advance the solution of the equation (11) from t to $t + \Delta t$ as well as to account for the dependence on the parameter ρ. Therefore we aim to create a NN model $f_{\texttt{NN}}$ such that

$$f_{\texttt{NN}}(\mathbf{x}(t), \rho) \approx \mathbf{x}(t + \Delta t),$$

where $\mathbf{x}(t) = [x(t), y(t), z(t)]^T \in \mathbb{R}^3$. Differently from the previous examples, in addition to $\mathbf{x}(t)$, the parameter ρ is also provided as an additional explicit input to the neural network. A further difference is that now we consider a large number of initial conditions, $n_{\text{IC}} = 200$, and no longer just one.



(a) NN output – $\rho = 17$, approach (I) (b) NN output – $\rho = 17$, approach (II)

(c) NN output – $\rho = 40$, approach (I) (d) NN output – $\rho = 40$, approach (II)

Figure 8: Numerical and NN predicted trajectories for two testing initial conditions (black dots) and parameter instances $\rho = 17$ (top) and $\rho = 40$ (bottom). For each case, in gray we plot the two lobes attractor on which the trajectories lie.

In Fig. 8, we report the results obtained by predicting the solution trajectories with approaches (I) and (II) (see Sect. 3.2). The trajectories are computed starting from two newly sampled initial conditions (not seen by the NN during the training) and for two test parameters instances $\rho = 17$ and 40. We note that approach (I) allows to trace with excellent accuracy the trajectories for initial conditions and for parameters not employed in the training. Regarding the trajectories predicted by approach (II), although they do not correspond to the numerical ones, they still lie on the two lobes attractor (depicted in gray in Fig. 8) relative to the considered parameter instance. This is actually a remarkable result. In fact approach (II) propagates the prediction error recursively over time by construction. For a chaotic system, such as the current example, a small perturbation can completely change the evolution of the system. Therefore, because of the chaotic character of the system under consideration, our major interest does not consist in accurately tracing the trajectories but rather in identifying the position of the attractors and their structure, which is precisely what we have obtained.

5 Conclusions

In this work, different strategies to construct models for complex dynamical systems have been presented.

As we have noted throughout the report, neural network models might be extremely accurate for fitting the dynamical map which defines the evolution of the system states. However, their performance depends on the complexity of the dynamics, they are extremely sensitive to noise and they have poor generalization ability – both with respect to initial conditions and time –, as we have highlighted, e.g. , in the application to the Kuramoto-Sivashinsky equation.

On the other hand, we have noticed that one possible improvement consists in training the NN model with a large number of initial conditions. In fact, in the case of the Lorentz equations for which we have chaotic dynamics, we have obtained that the trajectories identified by the NN model lie on the manifold characterizing the dynamics, even by varying the parameter of the problem.

In the case of dynamical systems for which we have just few measurements, possibly affected by noise, the NN-based methods are not suitable and we have made use of techniques such as (optimized) DMD and SINDy with bagging as they are more robust. Thanks to these techniques, in the example regarding the hare-lynx populations we have derived significant insights about the dynamics of the system and we have even forecast future states. Although the future predictions are not particularly accurate, the uncertainty estimates provided by these approaches have allowed us to define a confidence level on these estimates.

Finally, the identification of the presence of latent variables and an appropriate change of coordinate and dimensionality (either reduction with SVD or augmentation with time-delay embedding) have turned out to be crucial in the identification of the dynamical models.

References

[1] T. Askham and J. N. Kutz. Variable projection methods for an optimized dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 17(1):380–416, 2018.

[2] S. L. Brunton and J. N. Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.

[3] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[4] U. Fasel, J. N. Kutz, B. W. Brunton, and S. L. Brunton. Ensemble-sindy: Robust sparse model discovery in the low-data, high-noise limit, with active learning and control. *Proceedings of the Royal Society A*, 478(2260):20210904, 2022.

[5] D. Sashidhar and J. N. Kutz. Bagging, optimized dynamic mode decomposition (bop-dmd) for robust, stable forecasting with spatial and temporal uncertainty-quantification. *arXiv preprint arXiv:2107.10878*, 2021.

Appendix A: Code

All the code is uploaded at the Githug page `https://github.com/ContiPaolo/ModellingFromMeasurements`.