Modelling from Measurements: Homework report

Nicola Fonzi*

In this work several applications of modern machine learning techniques are applied to a wide variety of problems. Difficulties and implementations best practise, for the cases at hand, are also reported.

I. Nomenclature

DMD	=	Dynamic Mode Decomposition
$SINDY$	=	Sparse Identification of Nonlinear DYnamics
NN	=	Neural Network
SVD	=	Singular Value Decomposition

II. Introduction

Machine learning has received increasing attention from the research community in the last years. Thanks to modern techniques we are now able to find patterns in problems that were extremely challenging to study before, if possible at all. Also, beside non-interpreted models provided by neural networks, further studies issued methods to extract governing equations from data, rendering the result general and more easily applicable. In the context of this work, several of the most modern methods are explored and applied to famous problems. The goal of this report is to provide a small summary for future applications of the mentioned methods. The reference for a deeper discussion on the topics is [1].

In section 3, the most important equations used in the context of this work are reported. In section 4, a brief explanation of the framework used for the calculations is outlined. In section 5 the results are presented, together with a discussion of them. Finally, in section 6, conclusions are drown.

III. Background

In this section, the equations and methods used for the current work are reported.

A. Dynamic Mode Decomposition

DMD tries to find the optimal linear system evolving the state in time. The procedure is as follow: we take the SVD of the snapshot matrix X.

$$X = USV^* \tag{1}$$

Then we solve for the matrix A, evolving the system in time, as:

$$A = X'VS^{-1}U^* \tag{2}$$

Where X' is the snapshot matrix shifted in time of one time step, and the operation * consists in the complex conjugate transposition. Usually, the SVD is truncated retaining only the very first few modes in U.

Finally, an eigenproblem is solved for the eigenvalues in A, and the corresponding eigenvectors. The modes of A are then advanced in time analytically using the exponential solution and the full state can later be recovered.

B. Lotka-Volterra

A standard model for the predator-prey dynamics can be formulated as:

*Politecnico di Milano, Via La Masa 34, Milano. nicola.fonzi@polimi.it

$$\begin{aligned} \dot{x} &= (a - by)x \\ \dot{y} &= (cx - d)y \end{aligned} \tag{3}$$

C. SINDY

The concept of SINDY is to perform a sparse regression of the data to find the best model that describe it. We construct a matrix $\dot{X}$, containing the time derivatives of a system, organising the variables column-wise and the time snapshots row-wise. Further, we construct a database Θ so that:

$$\Theta = [\ f_1 \mid f_2 \mid \dots\ f_n] \tag{4}$$

Where f_k is a vector containing the k-th function, evaluated at the different time stamps.

The loadings ξ can be found solving, promoting sparsity, the system:

$$\dot{X} = \Theta\xi \tag{5}$$

D. Kuramoto-Sivashinsky equation

The PDE is reported here for completeness:

$$u_t = -uu_x - u_{xx} - \nu u_{xxxx} \tag{6}$$

IV. Methods

The entire range of results presented in this report has been obtained using the commercial software MATLAB. This choice increased significantly the efficiency of programming and provided a more intuitive framework. The code is available at Nicola-Fonzi GitHub personal page.

V. Results

A. Predator-Prey dynamics

In the context of this exercise we strive to obtain a dynamic model for the dynamics of lynxes and hares in North America, registered between 1845 and 1903. In order to do so, we divide the time series of data in two parts, the training set and the testing set. Indeed, the first 90% of data will be used to discover the model, while the last part will be used for a performance evaluation.

1. DMD

At first, we attempt to fit a linear dynamic model. Usually, when regression models are used, like DMD, we are in the high data limit, where a large portion of the state of a system can be observed. However, here, only two states can be analysed, namely the number of individuals of the two species. Thus, we do not expect DMD to be effective in its "standard" formulation. In practise, time embedding will be later used to extend the number of observable states. However, the number of latent variables is probably really high (the dynamics of a specie can be influenced by a countless number of factor, including the environment, predators other than the only one considered, humans), so the time embedding will not be an effective mean to increase the accuracy of the model. Finally, an extended DMD will be applied, providing qualitatively good results.

In figure 1, the evolution in time of lynxes and hares are reported. It can be seen that the data available is limited, and coarse in time. An interpolation in time has been considered as a viable option in order to increase the number of available samples, but it was later decided that this could introduce a fictitious dynamics in the system, other than the real one. Standard DMD was then applied to the system. The approximated solution is presented in 2.

It must be noted that here only the training set is reported. This is due to the fact that the model is unable to capture the dynamics at all, thus it makes no sense to test it on a new set of data. It can be seen that two decaying exponential

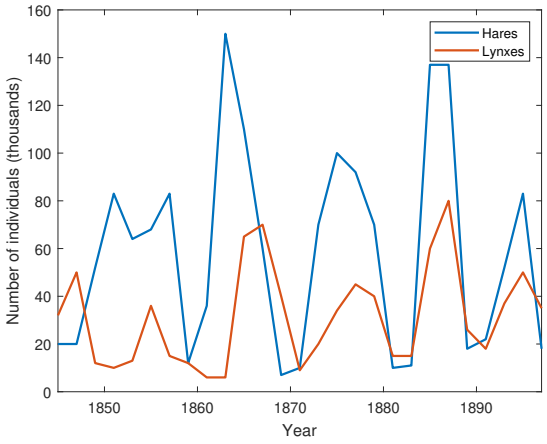

Fig. 1 Evolution in time of Hares and Lynxes in North America.

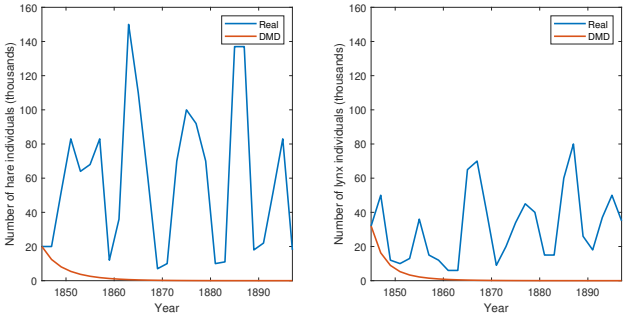

Fig. 2 Evolution in time of Hares and Lynxes in North America, both with real data and with an approximated DMD model.

solutions are found. This is to be expected as, with only two states, the DMD eigenvalues can either be a complex conjugate pair or two real values.

Further, the method has been increased in complexity. Data has been centered, this already changed the two real eigenvalues in a complex conjugate pair. Further, time embedding has been applied. It was found that with too much time embedding the system became unstable, thus the "optimal" number of embedding was taken as 5. It can be seen from the graph of the singular values of the snapshot matrix, in figure 3, that many latent variables are expected, as no value is zero.

The approximated reconstruction, again only on the training set, is reported in figure 4.

The solution is clearly better than the previous one, but still not satisfactory. Thus, exploiting the physical knowledge we have, we extend the state using three states: x_1^2, x_2^2 and $x_1 x_2$. Where x_1 is the number of hares and x_2 is the number of lynxes. This comes from the knowledge of common analytical predator-prey models, where quadratic terms often appear.

The rank of the snapshot matrix is now 22, much larger than before, and some singular values are now close to zero. However, due to the small state dimension, it was decided not to truncate the base anyway. The obtained approximating model is reported in figure 5. It must be noted that here, two sets of results are merged in the same plot, for visualisation purposes. First, until 1895, the results of the DMD model reproducing the training set is reported. At the end of the training phase the model as been pinned to the real data, similarly to what would happen in practise when trying to forecast the future, and the future states are predicted.

It can be seen that the training set is perfectly reproduced, while the future state is qualitatively correct.

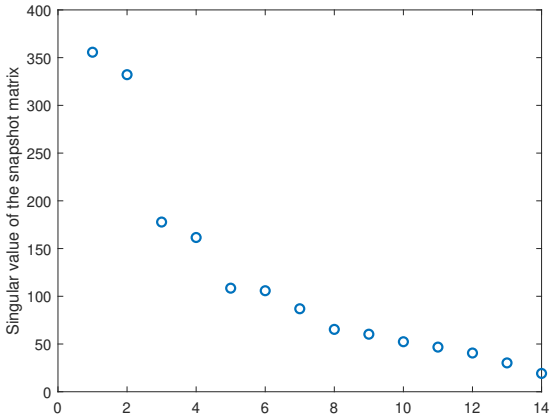

Fig. 3 Singular values of the snapshot matrix, with time embedding, containing the evolution of hares and lynxes in time.

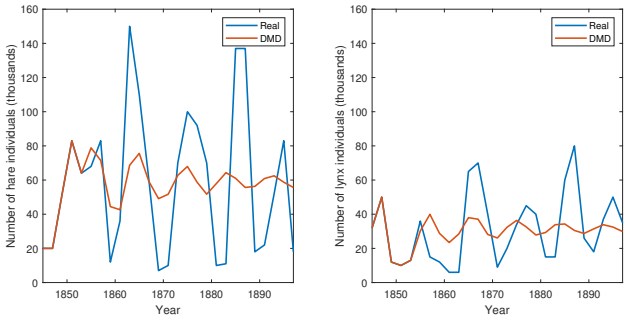

Fig. 4 Evolution in time of Hares and Lynxes in North America, both with real data and with an approximated DMD model with time embedding.

2. Lotka-Volterra

After completing a linear model, we try to fit a common and well known predator-prey model to the data at hand. The model is given by the Lotka-Volterra set of equations, which are not reported here but can easily be found in the literature.

In order to fit the equations, a linear problem is set. We define $\dot{X}_a$ as the column vector containing all the derivatives in time of, respectively, the hare population if $a = 1$ or the lynx one if $a = 2$. Then, we define the matrices Θ_1 and Θ_2 as:

$$\Theta_1 = \begin{bmatrix} X_1 & X_1 X_2 \end{bmatrix} \tag{7}$$

$$\Theta_2 = \begin{bmatrix} X_2 & X_1 X_2 \end{bmatrix} \tag{8}$$

Where the column vector X_a contains the number of individuals of a population defined by the subscript a as before. The coefficients of the Lotka-Volterra equations are then obtained solving:

$$\dot{X}_a = \Theta_a \xi_a \tag{9}$$

The systems are overdetermined, thus they are solved in a least square sense. It is of great importance the correct calculation of the derivatives. It was both considered to use the raw data and to use a spline in the neighborhood of the data point. The results of the different methods are compared in the following table.

The two results show similar coefficients, so it was decided that the raw data was providing enough accuracy for the derivatives.

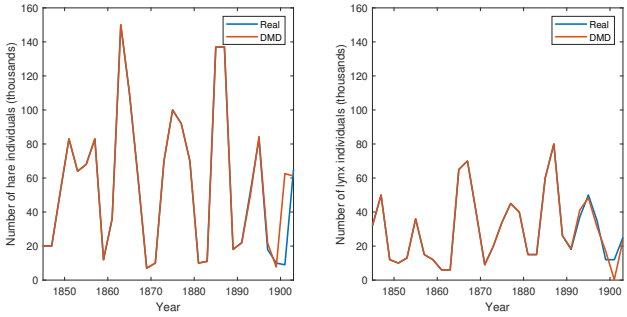

Fig. 5 Evolution in time of Hares and Lynxes in North America, both with real data and with an approximated DMD model with time embedding and state expansion.

Derivatives with interpolated data	Derivatives with raw data
$\dot{x}_1 = 0.2066x_1 - 0.0051x_1x_2$	$\dot{x}_1 = 0.1517x_1 - 0.0038x_1x_2$
$\dot{x}_2 = -0.2176x_2 + 0.0027x_1x_2$	$\dot{x}_2 = -0.1578x_2 + 0.0020x_1x_2$

Further, the obtained model was used testing its ability to reproduce the training data. The resuls are presented in figure 6.

It can be seen that the really simple model is not able to accurately capture the large variablity of the dynamics. Further, the population also reach negative values, which is clearly not physical. However, as it can be expected, it correctly represents the peak in the prey population slightly before the peak in the predator population.

3. SINDY

We now relax the constraint on the assumed equation that was used in the previous section, utilising a larger range of possible nonlinear terms. Indeed, we will use SINDY algorithm to discover the nonlinear equations governing the dynamics from scratch.

The approach is very similar to what was done before. We construct a matrix $\dot{X}$, stacking the the column vectors introduced before: $\dot{X} = [\dot{X}_1\dot{X}_2]$. Further, we construct a database Θ of all the functions we want to introduce in the analysis. Different terms, especially quadratic, have been introduced in the database. The loadings can be found solving, promoting sparsity, the system:

$$\dot{X} = \Theta\xi \tag{10}$$

Unfortunately, no good model for the system dynamics could be found. In figure 7 and 8, the coefficients for two possible system of equations are reported.

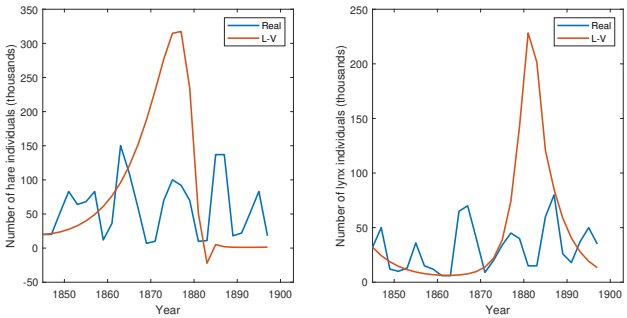

Fig. 6 Evolution in time of Hares and Lynxes in North America, both with real data and with an approximated Latka-Volterra system of equations.

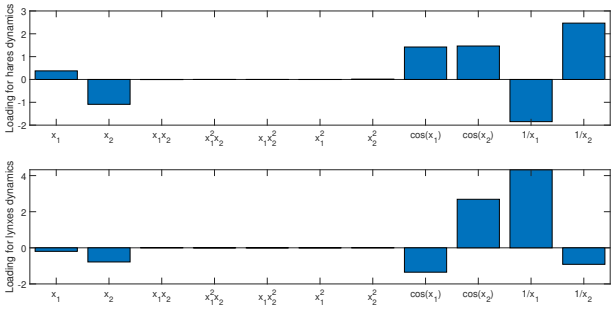

Fig. 7 Loadings value for the equations evolving lynx and hare populations in time.

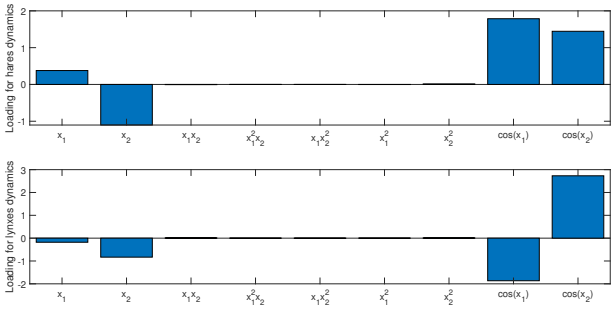

Fig. 8 Loadings value for the equations evolving lynx and hare populations in time.

The first model has issues as the population of lynxes tend to zero in the integration in time, thus the inversely proportional term goes to infinity. Removing the terms $1/x$ resolves the issue of the infinite value, but does not improve the prediction accuracy. The author believes that, due to the large number of latent variables of the system, standard SINDY may have issues finding a good model. A possible solution, that will be explored in the future, may be to exploit a time embedding of the system dynamics, extracting the first r columns of the singular value modes as new state. Then, the latent variables should be included in those modes and a better model could be found.

All the results have been obtained with a range of lasso lambda parameters $\lambda = [0.001 \ : \ 0.1]$. The results changed with lambda, but always showed the same pattern as the one presented in above figure (obtained with $\lambda = 0.001$). Indeed, all the models considered the proportional terms, the inversely proportional ones and the cosine terms as most important.

Interestingly enough, the term $x_1 x_2$ always disappears. This is not expected and supports the idea that more variables are needed.

B. Kuramoto-Sivashinsky equation

Here, neural networks are introduced in order to capture the evolution in time of the Kuramoto-Sivashinsky equation. Different architectures have been considered, starting from a simple network consisting only of fully connected layers, reaching also the introduction of more complicated layers like the long short-term memory layer. Further, different transfer functions have been considered.

At first, only one simulation has been used as a training set. The first 4 seconds have been used for the training, and the last 2 seconds for the testing. The validation set is automatically extracted by the code from the training set. In the following, in figure 9, results related to a network consisting of 3 hidden layers, with 1000 neurons, and linear activation function are presented. The network has been trained for 1000 epochs, using a stochastic gradient optimiser, with a batch size of 100 (the total number of samples is 400).

It can be seen that the training set is correctly reproduced. However, the result does not generalise and, as soon as the input to the network is "new", the model diverges. At first, this was seen as a symptom of overfitting. However, introducing more neurons, stopping the training before, introducing more layers, did not solve the issue. Probably, the strongly chaotic behaviour of the system provides a set of inputs to the network, in the testing phase, which is out of the

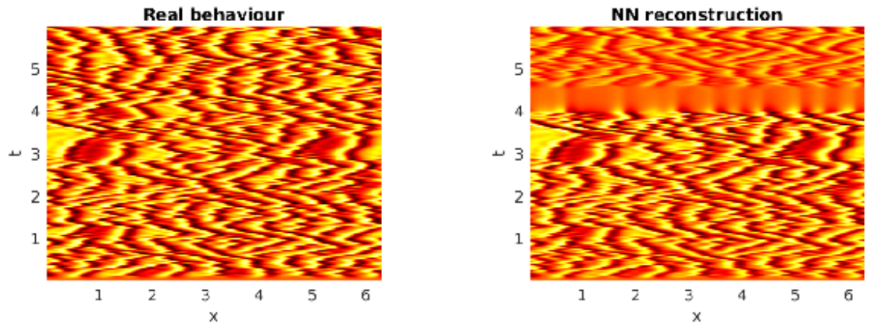

Fig. 9 Evolution in time of the Kuramoto-Sivashinsky problem. First 4 seconds: training set, last 2 seconds, testing set. Neural network trained with one time history.

range of inputs used for training. Thus, extrapolation is requested to the network. In order to solve the issue, more simulations have been added to the training samples. Indeed, 1000 simulations of 4 seconds have been used, hoping to create a dataset large enough to contain also future generated inputs.

In figure 10 the result of this process is presented. Here, again, we train the network with the parameters described before and we compare the prediction capabilities for the training and testing phase.

Unfortunately, the data set is not sufficient to include the really complex behaviour and the result is not satisfactory. The solution to this problem is still under examination.

C. Reaction-Diffusion

In this section, the Lambda-Omega reaction diffusion equations, between two scalar variables, are analysed. The behaviour of the system is first obtained with a standard integrator. The obtained database is later used to train a neural network. One time history only is used, and the time history is divided in 80% of data for training, and 20% for testing. The large state space is projected onto a lower dimensional space. This is done constructing a snapshot matrix of the states at each instant of time, obtained through a reshape of the bidimensional vector, and then stacking the u and v variables into a single column vector. Singular Value Decomposition is then applied and it can be seen that only the very first modes are actually important. In practise, the first 10 modes are more than enough to represent the full state. Further, a simple DMD model has been built, as the SVD had already been performed. This showed that even a simple linear model could advance accurately the system in time. Due to this, it was decided to test a neural network without activation functions. This implies a linear relation between the input and the output, which is in general not desirable. In this case, however, we want to test if, given this constraint, the network is still able to advance the system in time. Three layers are used, with [20 10 20] number of neurons respectively (encoder-decoder structure) and 1000 epochs are used with a stochastic gradient descent. In figure 11 the variable u, at different instants of time (already in the testing phase), is compared with the output from the network. In figure 12 the same is done with variable v.

The agreement is good, only at the very end of the simulation, far from the initialisation point, there is a loss of energy in the neural network prediction, that leads to lower peak values. However, given the simplicity of the network and the speed of training, the result is quite remarkable.

D. Lorenz equations

In this section, we try to reproduce the integration in time of the Lorenz equations using a neural network. Different values of ρ, a parameter governing the system, are used for training, namely $\rho = [10\ 28\ 40]$. The value of the parameter is added, at each instant of time, to the input layer. The output, on the other hand, still consists only of the state of

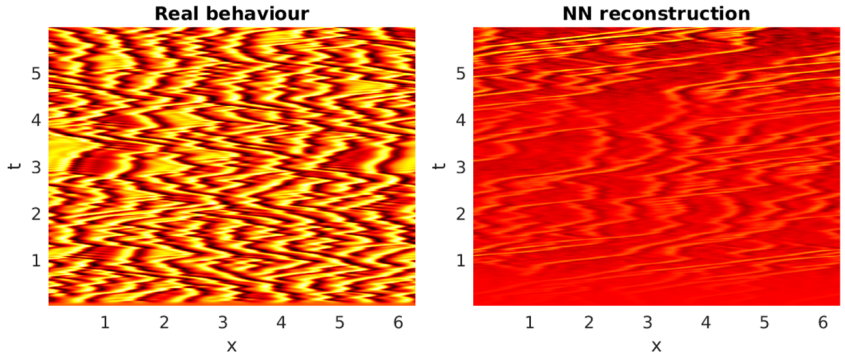

Fig. 10 Evolution in time of the Kuramoto-Sivashinsky problem. First 4 seconds: training set, last 2 seconds, testing set. Neural network trained with 1000 time histories.

the system. For each ρ value, 100 simulations are built with an ODE integrator. The architecture of the network is an encoder-decoder structure, with linear activation functions, three hidden layers with [20 10 20] neurons, and, thanks to the small size of the network, the optimiser is a second order Lavender-Marquardt method. The obtained network is then tested on a new set of 5 simulations, with ρ = [10 17 28 35 40].

Some results are reported in figure 13 and 14. In the first plot, the predicted solution for $\rho = 40$ is shown, while in the second, the solution for $\rho = 35$, value of the parameter outside the training set, is outlined.

Contrary to the case of the K-S equations, also chaotic in nature, thanks to the large number of data available, at each value of the parameter, if another initial condition is given to the network, the solution does not diverge. Actually, the prediction is quite accurate for the first seconds. However, when the parameter is changed outside the training set of parameters, the solution of the network is not representative of the reality anymore. A possible solution may be to provide a much finer discretisation of the possible values of ρ so that the network would always interpolate.

Further, using only one value of ρ, it was tested the ability of the network to predict the transition from one attractor to the other. A label consisting of a value -1 or +1 has been added to the output layer, depending on the attractor governing the state at each point in the trajectory. The label is based on a simple search of the peaks in the x value, corresponding to the transition phase. The network has then been trained again with 100 trajectories starting from different initial conditions, as already done before. Finally, a new initial condition has been provided to the system to understand if the network was able to predict the transition too. The ρ value has been removed from the inputs as it is now constant. Results are reported in figure 15.

It can clearly be seen that, even if the flag does not reach the training values, the transition can be predicted. Indeed, the prediction of the system loses its validity after approximately 5 seconds, but two jumps in the transition flag appears in this time frame. The first corresponds to the transition at the very beginning of the simulation, while the second to the one occurring at 4 seconds.

E. Belousov-Zhabotinsky chemical oscillator

In this final exercise, we try to reproduce the evolution in time of the chemical oscillator, starting from snapshots of its behaviour.

First, data is centered removing the means. Second, data is decomposed using a SVD, finding that a subset of around 40 modes is sufficient to reproduce the pictures. The reader should note that, if the means are not removed, one mode has a much stronger influence on the reconstruction. That is, the mode associated with the bubbles formed by the reaction has a singular value of two orders of magnitude higher than the others. This may create issues when trying to identify the correct number of modes to retain in the analysis and also may lead to a loss of variance included in the derived models.

Afterwords, a DMD model is constructed using only the first 1000 time steps (out of 1200) and the future state is

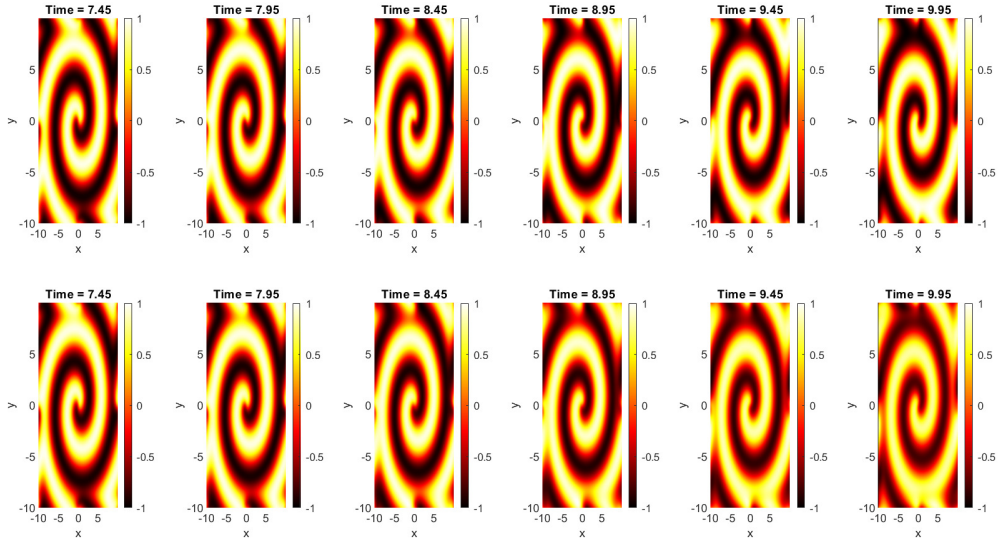

Fig. 11 Evolution in time of u variable in the Reaction-Diffusion problem. Upper row: real behaviour, lower row: predicted behaviour with a neural network.

predicted. This is presented in figure 16.

In the training phase the agreement is excellent, also for the smaller features produced by the reaction. However, in the testing phase, the last two columns, the agreement decays. In order to capture better the system behaviour, an autoencoder, able to advance in time, is build with a neural network. The architecture is such that the first layer encode the state in a number of modes equivalent to the number of DMD modes. No activation functions are used so that this encoding should learn the SVD itself. Later, three layers with clipped Relu activation functions are used to advance the solution in time. Finally, a decoder expand in the state space again. The result is reported in figure 17.

Unfortunately, due to hardware limitations, it was not possible to train with more than 300 time steps. For this reason, only the first hundreds of steps are reported in the figure. Probably, due to this, the performance of the network is limited. It is belief of the author that, with more samples for training, better agreement can be found.

VI. Conclusions

The outlined work gave the author the opportunity to gain insights in the applications of modern machine learning techniques. In particular, DMD demonstrated to be a very efficient and fast way to obtain a first, and linear, approximation of the dynamics in a system. The application of time embedding and state expansion provided and effective way to introduce nonlinearities, ideally converging to the Koopman operator.

SINDY demonstrated also to be straightforward to implement, but more physical understanding of the system is required to the programmer. Indeed, the correct choice of database function is a must. Provided this condition, the method is able to give the user an interpretable and general model for the dynamics.

Neural Networks have an enormous potential due to the virtually infinite architectures that can be used. Unfortunately, more guessing work is involved in their construction. Overall, the result of their application is strongly influenced by the experience of the programmer.

In the future, a better database will be provided to SINDY, in order to solve the predator-prey problem. Further, deeper studies will be concentrated on chaotic systems, and how to approximate them with neural networks.

References

[1] Brunton, S. L., and Kutz, J. N., *Data-driven science and engineering: Machine learning, dynamical systems, and control*, Cambridge University Press, 2019.

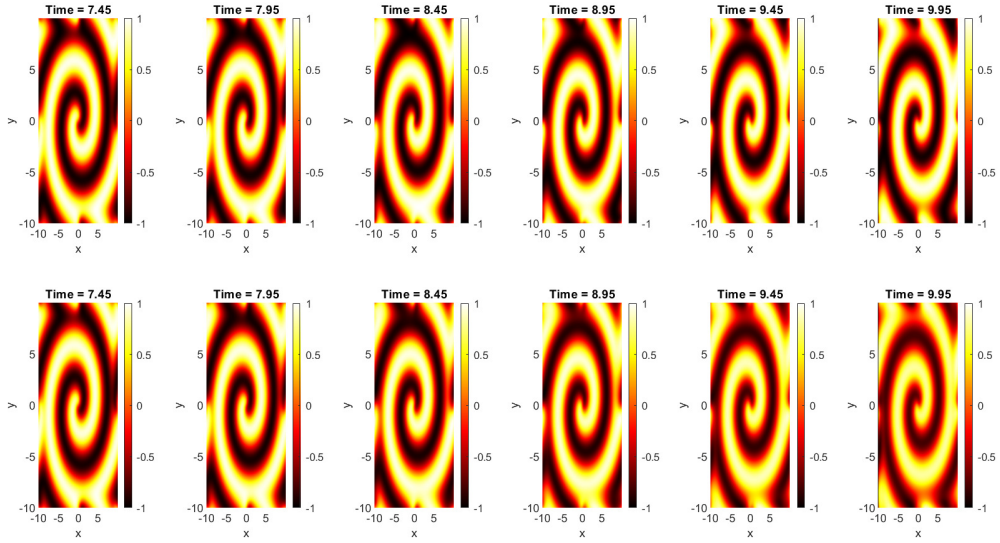

Fig. 12 Evolution in time of v variable in the Reaction-Diffusion problem. Upper row: real behaviour, lower row: predicted behaviour with a neural network.

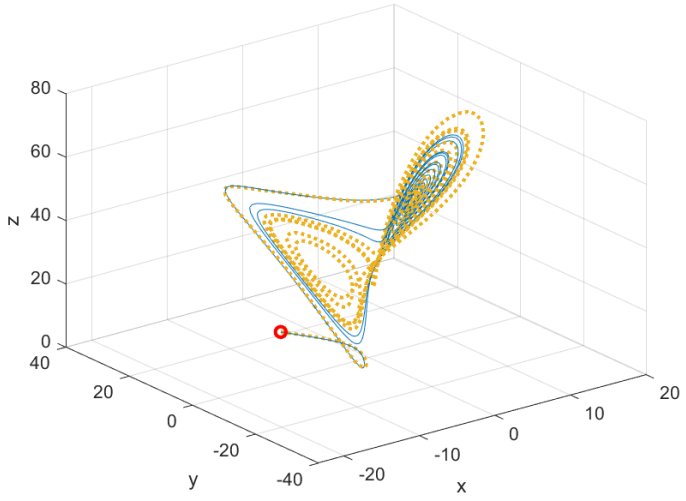

Fig. 13 Evolution in time of the state in a Lorenz system. Comparison between ODE integrator (blue line) and neural network (dashed line). $\rho = 40$.

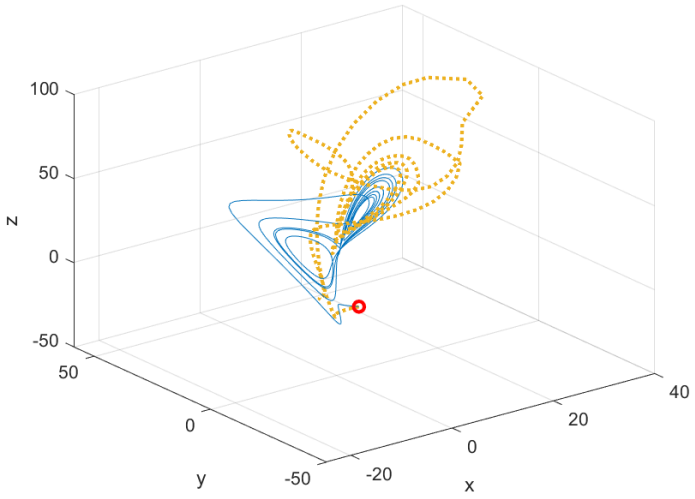

Fig. 14 Evolution in time of the state in a Lorenz system. Comparison between ODE integrator (blue line) and neural network (dashed line). $\rho = 35$**.**

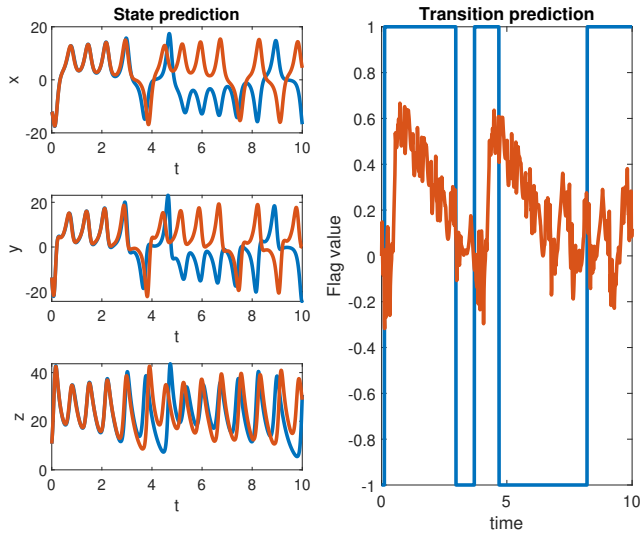

Fig. 15 Evolution in time of the state and the transition point in a Lorenz system. Comparison between ODE integrator (blue line) and neural network (dashed line). $\rho = 40$**.**

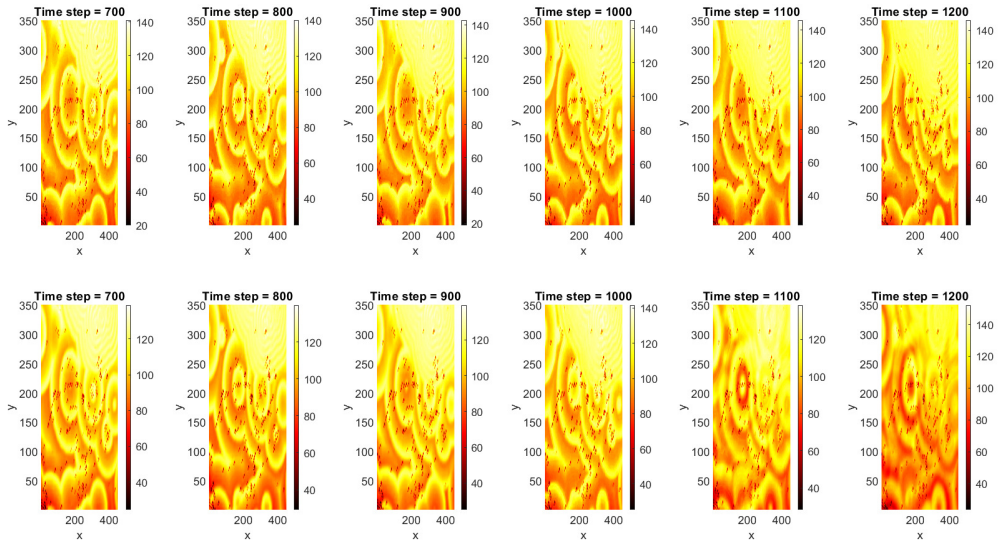

Fig. 16 Evolution in time of a chemical reaction. Upper row, real behaviour; lower row, DMD model.

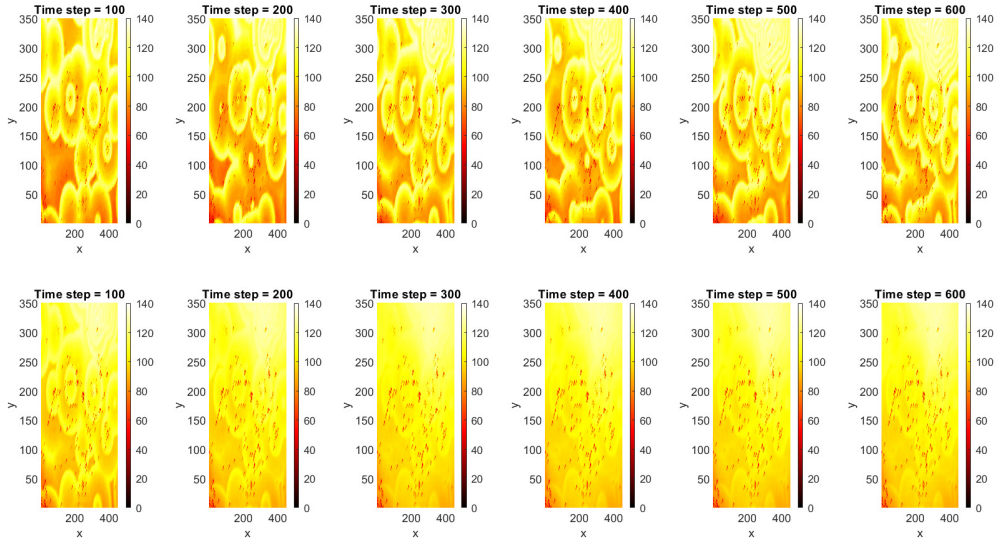

Fig. 17 Evolution in time of a chemical reaction. Upper row, real behaviour; lower row, NN autoencoder model.