POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Modeling from Measurments

PhD Course
Mechanical Engineering - Ingegneria Meccanica

Alessio Prini, 998999

Professors:
Prof. J. Nathan Kutz
Prof. Alberto Berizzi

Phd Cycle:
XXXVII

Abstract: Understanding and analyzing complex systems have always been one of the main goals in research. The tools developed to do this have for a long time been limited to the definition of difficult analytical models and to the description of simple empirical models. Today, thanks to the development of information technology and the availability of computing power, it is possible to analyze and model extremely complex phenomena events more effectively and in greater detail. Remarkably, "giants of the web" are investing in the deployment of such tools.

Key-words: Data Analysis, Dynamic Mode Decomposition, ANN, SINDy

1. Introduction and Overview

The work shows different techniques for analyzing, predicting, and manipulating data. The exercises will allow to experiment with some techniques, become familiar with the tools for their use, and understand their potential and limits. Each exercise will be reported in the form of a MATLAB or Jupyter notebook on the GitHub repository to leave a trace of the procedures carried out. The document is divided into sections dedicated to introducing specific aspects of the techniques used. In Section 2, the theoretical bases of techniques used will be introduced. In Section 3 the implemented algorithms and tools used will be introduce. In Section 4 the results of the experiments will be shown. Finally, in Section 5 we will focus on the final results and conclusions

2. Theoretical Background

2.1. DMD - Dynamic Mode Decomposition

The idea behind the Dynamic Mode Decomposition (DMD) is that a series of data could be reconstructed and so forecast using a series of modes associated with a frequency and growth or decay rate. DMD algorithm is based on SVD (Singular Values Decomposition) from which inherit the eigendecomposition capabilities [2]. The algorithm analysis start from the data representation; consider a matrix X made by a series of system state snapshots:

$$X = [x_1 \; x_2 \; \dots \; x_n] \tag{1}$$

Every vector $x_i = x(t_i)$ represents the list of the state variables at t_i instant. Now, considering the system composed by a series of a linear system, the equation 2 could represent the transformation between data:

$$X' \approx AX \tag{2}$$

where, considering a uniform sampling and then $x_{i+1} = x_i + \Delta t$, X' is composed by the vector series $X' = [x_2 \; x_3 \; \dots \; x_{n+1}]$. The transformation that best represent the system now could be:

$$A \approx argmin \|X' - AX\|_F = X'X^\dagger \tag{3}$$

Where $\|\|_F$ is the Frobenius norm, and $\dagger$ represents the pseudo-inverse. For high-dimensional x, the calculus of A and its decomposition is computationally heavy. To avoid this high computational cost the SVD capabilities are exploited to select dominant components without needing matrix A calculus. One of the strengths of the DMD algorithm is the dimensionality reduction. The exact DMD algorithm is composed as follows:

1. Computation of SVD of X matrix:
$$X = U\Sigma V^* \tag{4}$$
matrix $X \in \mathbb{R}^{m\times n}$, $U \in \mathbb{C}^{m\times m}$, $\Sigma \in \mathbb{C}^{m\times n}$ and $V \in \mathbb{C}^{n\times n}$ where m is the system state vector dimension. Analysing the resulting modes, it is possible to operate a dimension reduction considering just the first r dominant modes. In this case
$$X \approx \tilde{U}\tilde{\Sigma}\tilde{V}^* \; with \; U \in \mathbb{C}^{n\times r}, \Sigma \in \mathbb{C}^{r\times r}, V \in \mathbb{C}^{m\times r} \tag{5}$$
2. From above:
$$A = X'V\Sigma^{-1}U^* \; or \; considering \; mode \; reduction \; A \approx X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^* \tag{6}$$
3. Introducing W and Λ respectively the eigenvectors and eigenvalues matrix of $\tilde{A}$, and Φ the high dimensional mode matrix of A
$$\Phi = X'\tilde{V}\tilde{\Sigma}^{-1}W \tag{7}$$
with few step it comes to:
$$x_k = \Phi\Lambda^{k-1}b \tag{8}$$
with mode amplitude b given by $b = \Phi^\dagger x_1$

With DMD, it is possible to approximate every system's behavior as a linear system composition. For non-linear system this is clearly an approximation. For DMD is important that the system dynamics is completely represented from the measured data. Sometimes this is impossible, for example, when some state variable could not be measured. In this case, the associated dynamic show is influenced but not explicitly. In this case, one can speak of **latent** or **hidden variables**. The solution to this problem is to project the state vector onto a higher dimension and build a linear model in this space. This can be made by the **time-delay embedding** [3] where the matrix data X can be constructed assembly delayed copies of the matrix X.

$$H = \begin{bmatrix} x(t_1) & x(t_2) & \dots & x(t_M) \\ x(t_2) & x(t_3) & \dots & x(t_{M+1}) \\ \vdots & \vdots & \ddots & \vdots \\ x(t_k) & x(t_{k+1}) & \dots & x(t_{M+k}) \end{bmatrix} \tag{9}$$

Equation 9 is called Hankel matrix for a delay k. Exact DMD algorithm is sensitivity to noise signal, which in general, brings a bias in the resulting eigenvalues. Many solutions can be adopted to limit this problem. The most effective is the **optimal DMD** [1] that uses the variable projection algorithm and tries to minimize the difference between the measured data and the reconstructed. Optimal DMD shows its robustness and efficacy but remains sensitive to the initial guess provided to the algorithm.

2.2. SINDy - Sparse Identification of Non-linear Dynamics

The equations are generally obtained from physical model of the system or empirically. In SINDy data itself are used to identify the equations and the model of the underlying dynamic system. In general, scientists look for models that are (1) **Parsimonious**, with few terms, (2) and Interpretable to understand the underlying physics. Dynamic system equations are typically written as:

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t)) \tag{10}$$

with $\mathbf{x} \in \mathbb{R}^n$ the state variable vector. To apply SINDy algorithm, start to collect measurements data similarly to DMD and compute its derivative matrix:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^T(t_1) \\ \mathbf{x}^T(t_2) \\ \vdots \\ \mathbf{x}^T(t_m) \end{bmatrix} = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \dots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \dots & x_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \dots & x_n(t_m) \end{bmatrix} \dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{x}}^T(t_1) \\ \dot{\mathbf{x}}^T(t_2) \\ \vdots \\ \dot{\mathbf{x}}^T(t_m) \end{bmatrix} = \begin{bmatrix} \dot{x_1}(t_1) & \dot{x_2}(t_1) & \dots & \dot{x_n}(t_1) \\ \dot{x_1}(t_2) & \dot{x_2}(t_2) & \dots & \dot{x_n}(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x_1}(t_m) & \dot{x_2}(t_m) & \dots & \dot{x_n}(t_m) \end{bmatrix} \tag{11}$$

to derive the governing equations, SINDy start from a series of "basic" function multiplied by a weight value. The final equation identify by SINDy will be a composition of these basic functions multiplied for each weight. Mathematically SINDy reproduce the data measured as:

$$\dot{\mathbf{X}} = \mathbf{\Theta\Xi} \;\; with \;\; \mathbf{\Theta} = \begin{bmatrix} | & | & | & | & \dots & | & | & \dots \\ 1 & \mathbf{X} & \mathbf{X}^2 & \mathbf{X}^3 & \dots & sin(\mathbf{X}) & cos(\mathbf{X}) & \dots \\ | & | & | & | & \dots & | & | & \dots \end{bmatrix} \tag{12}$$

where $\mathbf{\Theta}$ is a matrix that represent the series of basic functions, provided as a matrix. Each column of $\mathbf{\Theta}$ represents the function evaluated for every instant. $\mathbf{\Xi} = [\xi_1\ \xi_2\ \dots\ \xi_n]$ is a sparse matrix of coefficients, the functions weight, that active the notable function. In our example $\mathbf{X} \in \mathbb{R}^{m\times n}$, $\mathbf{\Theta}$ is an $\mathbb{R}^{m\times r}$ matrix, with r number of notable function chosen in our library and $\mathbf{\Xi} \in \mathbb{R}^{r\times n}$.
An optimization algorithm is then applied to every column of (12) to find the coefficients of ξ_k related to every state variable. The most used algorithm for this scope are the least absolute shrinkage and selection operator (LASSO), least-square regression, elastic-net algorithm or ridge regression algorithm.

2.3. ANNs - Artificial Neural Networks

Artificial Neural Networks (ANN or simply NN) are composed of single artificial neurons interconnected each other by various patterns. Each neuron can be considered as a node that receives and produce signal transmitted through links. Every node can receive signals from other nodes and produce a signal transmitted to next nodes. Every node has an activation function, generally nonlinear, that transforms the signal received. Each link connect two neuron and has a weight that amplifies or mitigates the signal. Weights determine how much a node influence the others. This signal transmission is similar to the biological process produced by the axon-synapse-dendrite connections. Neurons or nodes are organized in layers. Neurons of one layer can only be linked previous or next layer. Input layer can communicate just with the following layer and output layer just with the preceding layer. Layers that communicate with preceding and following layer are called hidden layers. Mathematically the process described can be summarized as follow. First layer is described by [2]:

$$\mathbf{x_1} = \mathbf{f_1}(\mathbf{A_1}, \mathbf{x}) \tag{13}$$

Applying the same description to all the layer it comes to:

$$\mathbf{y} = \mathbf{f_n}(\mathbf{A_n},\ \dots\ \mathbf{f_2}(\mathbf{A_2}, \mathbf{f_1}(\mathbf{A_1}, \mathbf{x}))) \tag{14}$$

Changing parameters characterizing matrix A_k (weights) it is possible to change the network behaviour. Learning process is an adaptation process of the links' weights to better fits data given or in general improve the accuracy of a objective results given. Backpropagation technique calculate the gradient of an error function (loss function) w.r.t. the weights, to train the network or in other terms to find the best set of link weights. Backpropagation results strongly depends on the **Hyperparameter** chosen. Below is a non-exhaustive list of hyper network parameters:

- **Number of Hidden Layers**: The number of layer that are connected to preceding and following layer
- **Learning Rate**: Backpropagation algorithm is performed in several steps. Learning rate represent the amount of weights' change for every step
- **Activation function**: The node's function. Typical function are ReLU, Tanh, Sigmoid, Heaviside, Logistic
- **Batch size**: Dimension of the sample provided to the optimization algorithm. The optimization will be performed providing little samples of data in different steps.
- **Epochs**: How many time the complete dataset will be trained. Ideally higher is the number of epochs, more accurate will be the result

3. Algorithm Implementation and Development

Matlab and Python are the most widely used programming languages for data analysis and system identification and many library and tools have been created for this scope. In the next section library and tools used are cited and presented.

3.1. DMD - Dynamic Mode Decomposition

Library for DMD are present in both Matlab and Python. To analyse the population data, introduced in the next session, Matlab language was chosen. Although present in other libraries, exact DMD algorithm is implemented with few rows of Matlab code.
For optimized DMD algorithm the *optDMD* a Matlab library was used. The core of this library is the *optdmd* function. Optimized DMD could be used in conjunction with bagging technique, that is a statistical method whereby a single set of snapshots is used to produce an ensemble of optimized DMD models. This allows to better evaluate the noise sensitivity of the DMD techniques used and to define the mean and variance of the eigenvalues found. Time-delay technique is implemented in Matlab to create the Hankel matrix described in 9

3.2. SINDy - Sparse Identification of Non-linear Dynamics & Lotka Volterra

The identification was conducted using SINDy is *pysindy*, an open source library for Python. With **pysindy** we must create and ensemble optimizer (with the included STLSQ library) and use the provided **SINDy** function. The $\mathbf{\Phi}$ matrix presented in Equation 12 can be built through a variable, **feature_names**, as list of string function. Other important parameter to run the library are **threshold parameter**, the minimum value of ξ in $\mathbf{\Xi}$ to be considered not null, **alpha** is a regularization factor and **max_iter** the maximum value of iteration for the optimizer.
SciPy was used for optimization and integration of Lotka-Volterra equation and Matplotlib was used for graphs plotting.

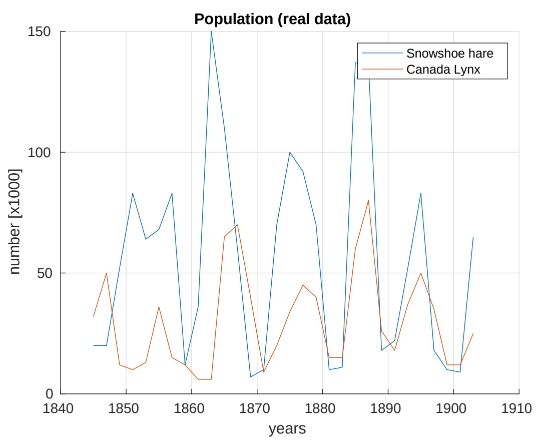

Figure 1: Population data of snowshoe hare and Canada lynx from 1845 to 1903

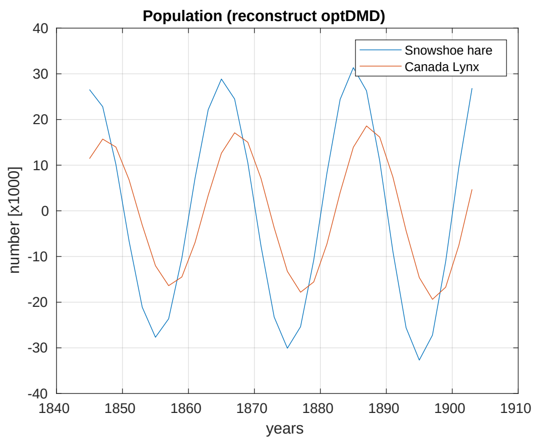

Figure 2: Population reconstructed with *optDMD* algorithm

Lotka-Volterra was implemented using the below function and integrated using *numpy.integrate.solve_ivp*.

```python
def lotkavolterra(t, z, b, p, r, d):
    x, y = z
    return [b*x - p*x*y, -d*y + r*x*y]
```

3.3. ANNs - Artificial Neural Networks

For ANN the PyThorch library was used and the instruments available in scikit-learn for learning and regression.

4. Computational Results

4.1. DMD on Population data

As the first step, a visual analysis of reported data (Figure 1) can give meaningful information. From this analysis it is possible to assert that: (1) Matrix $X \in \mathbb{R}^{2\times 30}$; (2) Snowshoe hare (the prey) population has always higher values than Canada Lynx; (3) The trends of the two series are similar, with different absolute values and a slight delay for Canada Lynx; (4) Each series seems to be composed of two sinusoidal with a period of 25-26 years. Applying the exact DMD algorithm to the matrix, two real eigenvalues were found, with values 0.7248 and 0.3784. These eigenvalues lead to a divergent solution, so is not possible to reproduce the time series given and forecast the populations trend.
Trying to improve these results the optimal DMD algorithm is applied using *optDMD* for MATLAB. As reported optimal DMD is based on an optimization algorithm that needs an initial guess. The result is showed in Figure 2. Two eigenvalues were found with values **0.0082 - 0.6301i** and **0.0082 + 0.6301i**. The solution found is composed of two complex and conjugate eigenvalues useful for reconstructing a sinusoidal function with a period of **19.94 years**. Real part of these eigenvalues are positive, which leads to slightly divergent time history. Temporal history of the Canada Lynx lags slightly behind the prey's data. A comparison between original and reconstructed data shows that only one of the two sinusoidal components is reproduced and in particular is the one with the highest values. This second sinusoidal component is part of the so-called hidden variables. Trying to make this part explicit, it is necessary to project the state vector onto a larger dimension thanks to the use time delay technique.
Next, the presented bagging technique is applied, choosing 100 as the number of permutations. The result are in Figures 3 and 4. The result found with bagging *optDMD* doesn't improve the plain usage of *optDMD* (Figure 2), but confirm imaginary and real part of the eigen values and give an indication of quite low variance. No significant outliers were found.
Trying to explicit the latent variable of the data series **time-delay** strategy were adopted.. Hankel matrix is processed through *optDMD* to obtain an analysis of the modes. Different levels of delay have been tested in order to obtain the best in terms of data series reconstruction and spacing of the delay values (k in (9)) from 2 to 10. 2 was found as the best delay value. 4 eigenvalues were found, complex and conjugate in pairs with a period of 394.45 years and 20.61 years. However it seems that the measured data series is composed of two sinusoids with the same frequency but out of phase by 180 ° and with a different amplitudes. The low frequency modes found (394.45 years) isn't the hidden variable search. Unfortunately, even with the use of time delay, is not possible to make explicit the second sinusoidal component. The result obtained with time-delay and *k=2* is depicted in Figure 5. Again the time bagging algorithm applied to the time delayed data doesn't show a significant improvement, and the reconstructed data shows the same problems.

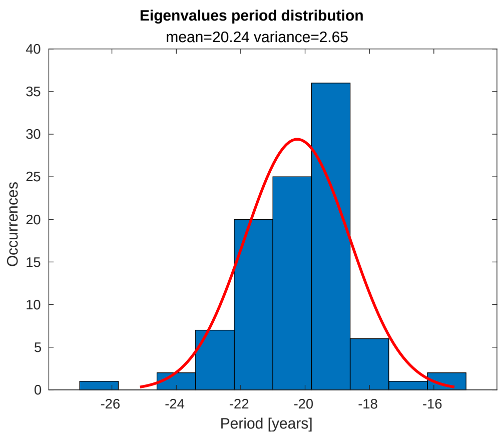

Figure 3: Period distribution for *optDMD* applied to population data

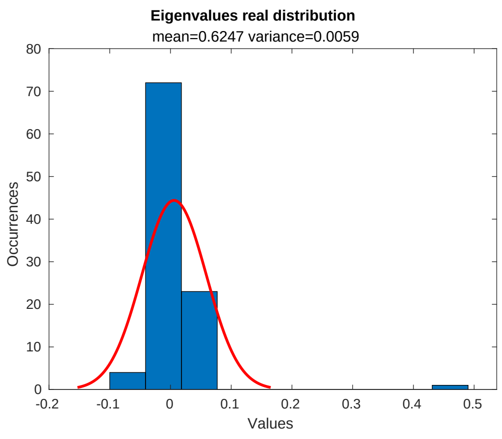

Figure 4: Real eigenvalues part distribution for *optDMD* applied to population data

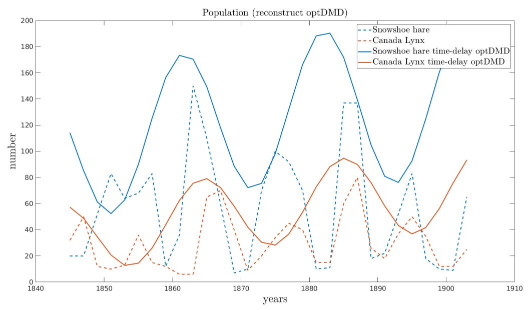

Figure 5: Comparison between original data and time-delay reconstructed data

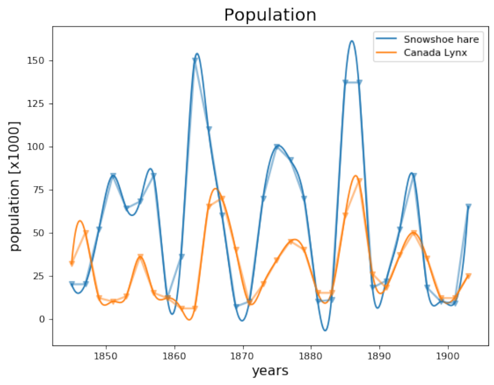

Figure 6: Spline interpolation of population data

4.2. Lotka-Volterra optimization on Population data

As reported in 3.2 the Lotka-Volterra function was implemented and $solve\_ivp$ function was used. In order to optimize the parameters of Lotka-Volterra $scipy.optimize.fmin$ was used. The values found are $b = 1.5, p = 1.017, r = 0.97, d = 3.05$ and the comparison with the real data in Figure 7 show how distant is the solution. Trying to improve the result the $scipy.interpolate.iterp1f$ function was used with the $kind = cubic$ argument that brings to a spline interpolation. The result of this interpolation can be seen in 6. Now the same algorithm was applied to try to improve the Lotka-Volterra approximation. The result is in Figure 8 and is it possible to see how far is the result from the real curve. It seems that in this case, the optimizer identifies some high-frequency contributions that try to fit just the individual points and not all the curve. The approach chosen seems to be not useful to solve the problem.

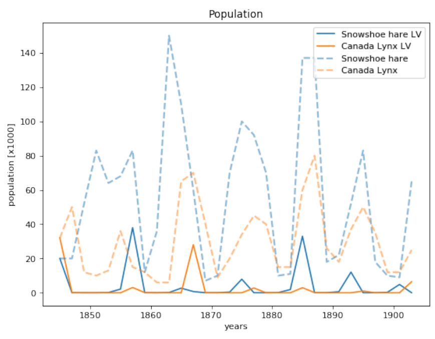

Figure 7: Lotka-Volterra parameter optimization (I trial, no interpolation) vs. Real Data

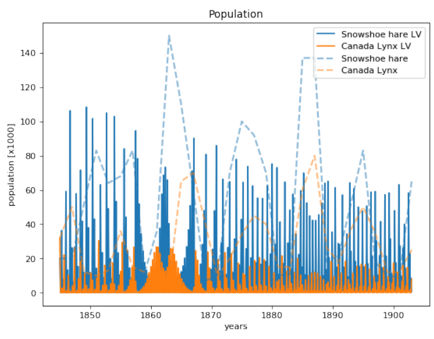

Figure 8: Lotka-Volterra parameter optimization (II trial, no interpolation) vs. Real Data

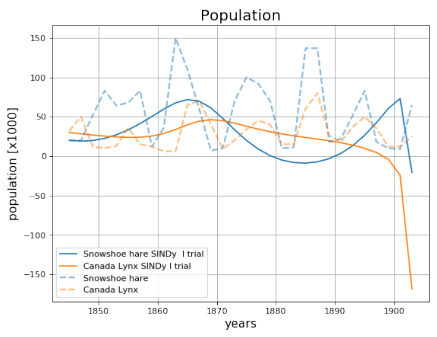

Figure 9: SINDy interpolation (I trial, no interpolation) vs. Real Data

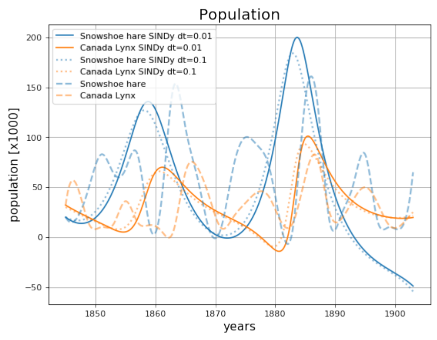

Figure 10: SINDy interpolation (II trial, spline interpolation) vs. Real Data

4.3. SINDy on Population data

The algorithm is applied to the same population dataset seen before (Figure 1). Python and *pysindy* library were used for this analysis. The implementation is explained in 3.2. Different tests were performed to identify the best function list to model the system. A combination of x and y powers were considered first.

```
feature_names = ['x','y','x^2','y^2','xy','1/x','1/y','1/x^2','1/y^2','x^3','y^3','x^4','
    y^4']
threshold=0.00005
alpha=0.5
max_inter=2000
```

These values were found after several tests. The result can be represented by Figure 9 The resulting model is:

```
(x)'  = 8.074 1 + -0.002 x + -0.087 y + 0.003 x y + -0.009 y^2
(y)'  = -4.983 1 + -0.032 x + 0.338 y + 0.001 x^2 + 0.001 x y + -0.007 y^2
```

Trying to improve the model another **feature library** is chosen. The population data present a sinusoidal trend, so seems to be natural to add *sin* and *cos* functions to the library. The new **feature library** is:

```
feature_names = ['x','y','x^2','y^2','xy','1/x','1/y','1/x^2','1/y^2','x^3','y^3','x^4','
    y^4','sin(x)','cos(x)','sin(y)','cos(y)']
```

The model found is completely the same of the first trial. Quadratic terms are the highest found from the algorithm. The identified curve does not represent well the starting data. Trying to improve the result a spline interpolation will be considered. The interpolation considered is the same showed in 4.2 and in particular in Figure 6. Initially we consider an interpolation with a dt of $0.1 years$. The solution found in terms of model is.

```
(x)'  = 12.604 1 + 0.122 x + -0.591 y + -0.001 y^2
(y)'  = -5.409 1 + -0.047 x + 0.268 y + 0.001 x^2 + 0.001 x y + -0.007 y^2
```

And the data series is depicted in Figure 10. After a narrower interval was chosen with a $dt = 0.01$. As in Figure 10 the results don't change so much w.r.t. the $dt = 0.1$ case and still doesn't represent really well the data. Anyway from SINDy we can confirm that a second order is the minimum in order to represent the population data. So probably the Lotka-Volterra could be not enough to represent this population data.

4.4. Kuramoto-Sivashinsky system

Kuramoto–Sivashinsky were derived to model the diffusive thermal instabilities in laminar flame front. A NN were used to estimate the trend of the curve. The network is trained to estimate the function's value at the instant $t + \Delta t$ starting from the value at t. NN is composed by an input and output layers with the same dimension of vector x of the function. Hidden layers were tested of varying sizes to determine the best configuration. First NN tested is composed in this way:

```
NeuralNetwork(
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=128, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=256, bias=True)
    (3): ReLU()
    (4): Linear(in_features=256, out_features=256, bias=True)
    (5): ReLU()
    (6): Linear(in_features=256, out_features=128, bias=True)
  )
)
```

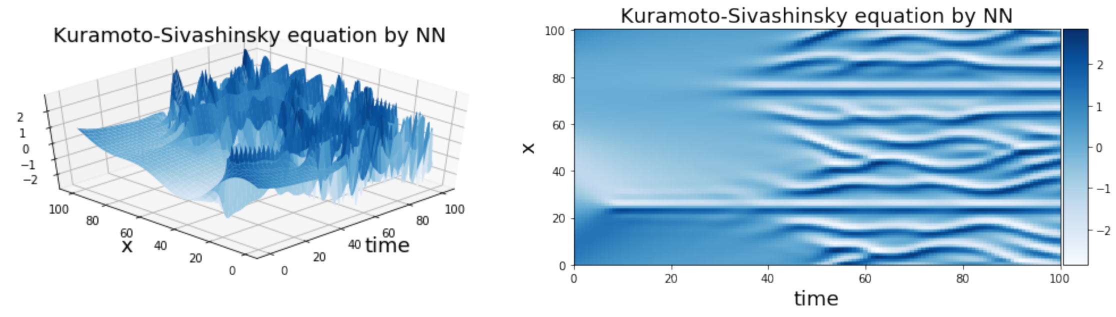

Figure 11: Data series forecast from NN starting from the same initial condition

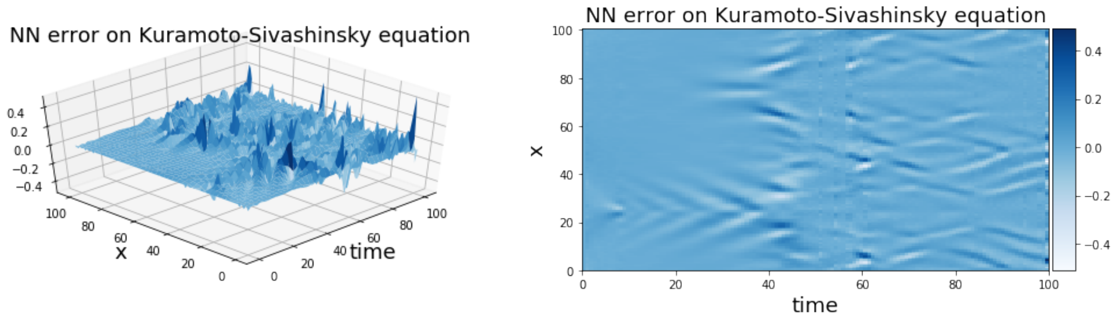

Figure 12: Difference between integration and NN solution

The hyperparameters been tested and the best have been chosen; **learning rate** = 1e-4, **batch size**=16, **epochs**=5000. Results found for the different tests carried out are shown below. The network was trained using an *MSELoss* function and *Adam* optimizer. After training same initial conditions were provided to the NN to test the results (Figure **??** and 12). The first figure shows the trend of the curve created with the NN, while the second shows the error with respect to the integrated solution. Chosen network, methods, and hyperparameters seem to predict the integrated curve with sufficient accuracy. The magnitude of the error is, in fact, an order of magnitude smaller than the absolute value of the function. The result changes if we consider instead the initial conditions used for training a different one (Figure 13). In this case the error has the same magnitude of the data series. Trying to improve the estimation two solutions were tested. The first used a Dataset including different initial conditions in the same matrix. The second change the network shape. Testing the first solution seems that the output is again not accurate, even starting from the same initial conditions on which the network is trained. The second solution exploited different networks. The first has a greater number of layers, but each layer has the same number of nodes of the first used (long NN). The second has the same length, but narrower layers (strict NN). The results (Figures 14 and 15) for both networks are not satisfactory as those found previously, even tested on the same initial conditions. Difference

4.5. Reaction-Diffusion system

Reaction-diffusion (RD) systems describe macroscopically how two media react and move among them. After the code implementation of the differential equation in MATLAB the *ode*45 algorithm was used to integrate the equation. Data were ported in Python, an SVD was applied to the data to understand the dominant singular values and apply a possible dimension reduction.
As presented in Figure 17 the first four modes are the most important, and the first six modes were used to apply

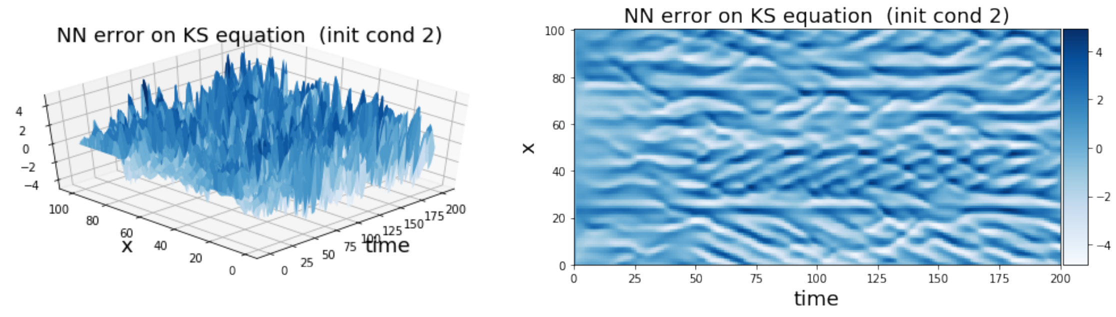

Figure 13: Difference between integration and NN solution for different initial condition

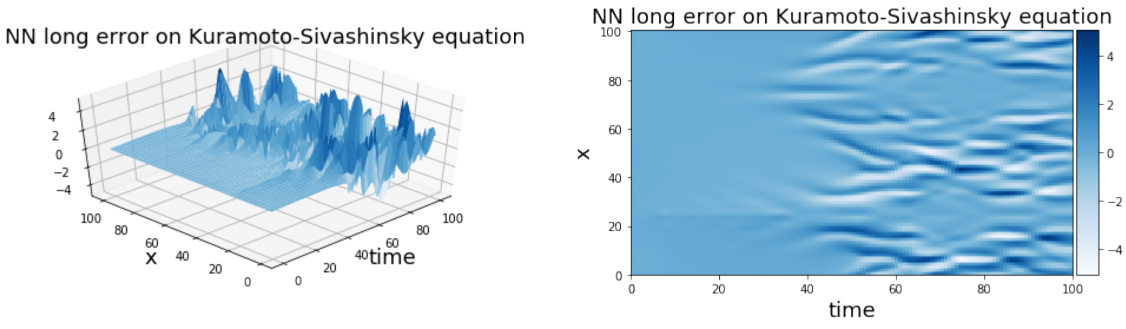

Figure 14: Difference between integration and NN solution for different initial condition

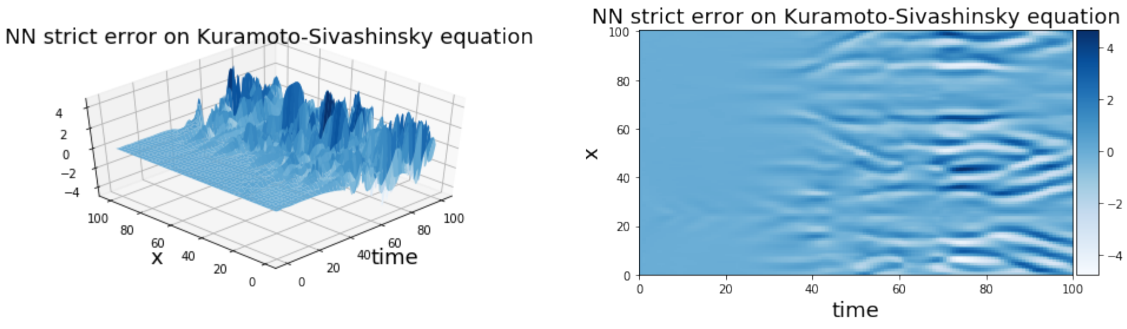

Figure 15: Difference between integration and NN solution for different initial condition

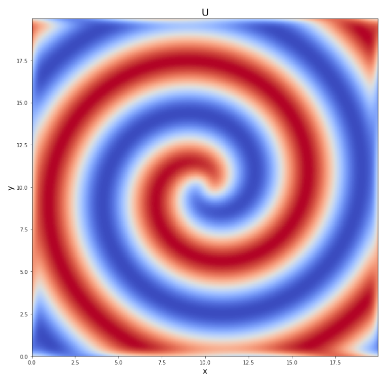

Figure 16: Reaction Diffusion U variable (time=0)

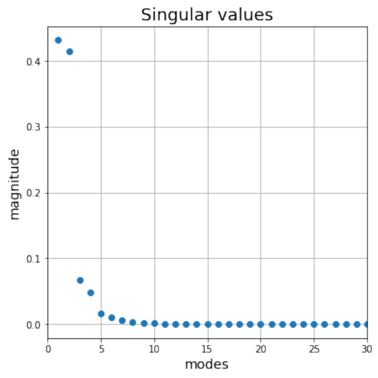

Figure 17: Reaction-Diffusion data Singular values

the reduction. The first six modes for the 98.6% of the energy, so using first six modes could be considered a good approximation of the real data. Now using the U matrix, the left singular vectors matrix, restricted to the first six singular values, the dimension reduction was made by projecting the data on the reduced space with:

$$X_{red} = U^* X \tag{15}$$

and X_{red} is used to train the network. To train the network the first temporal 90% of the data series will be used. The final 10% will be used to verify that the network correctly forecasts the series. *train_test_split* from *sklearn* for data series split in batch. NN shape is:

```
NeuralNetwork(
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=6, out_features=60, bias=True)
    (1): ReLU()
    (2): Linear(in_features=60, out_features=30, bias=True)
    (3): ReLU()
    (4): Linear(in_features=30, out_features=60, bias=True)
    (5): ReLU()
    (6): Linear(in_features=60, out_features=6, bias=True)
  )
)
```

MSELoss function and *Adam* optimizer have been used. Once the network was trained, data were compared with the original as depicted in Figure 19

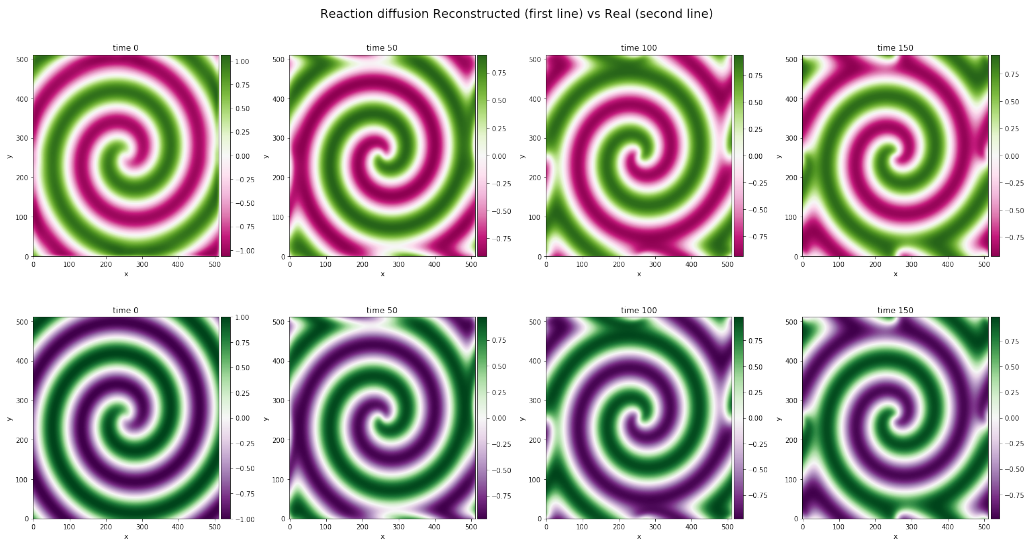

Figure 18: Comparision between Neural Network data and ODE data for different time instant

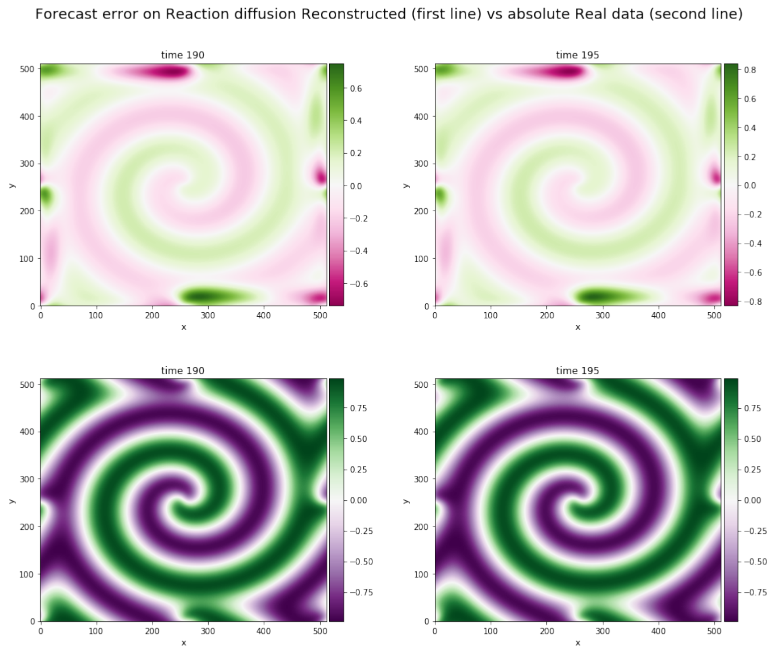

Figure 19: Comparison of ODE data (second line) and Neural Network error for different time instant

As in Figure **??** the Neural Network well reproduces data obtained from the model integration. The same network can also be used for forecasting. The last 10 sec of the data series is used to verify if the NN correctly forecasts the data series. The result is presented in **??** in terms of percentage error (first line). The figure clearly shows that the error is generally kept below 20%. Only for some peripheral parts, the error rises to values of up to 80%.

4.6. Lorenz equations

Lorenz equations are differential equations useful to describe e chaotic behavior. For this equations, our ability to predict its future course will always fail in the absence of perfect knowledge of the initial conditions. Lorenz equation describes the motion in 3 dimensions and is characterized by three parameters σ, β, and ρ. The goal is to predict the Lorenz result at $t + \Delta t$ instant, starting from the solution at t using a NN trained for this purpose. The prediction must also be respected by changing the ρ parameter values. For this reason is decided to train a network with 4 input (x, y, z, ρ) and 3 output (x, y, z). Three hidden layers were placed with a dimension of 30 and ReLU activation nodes and linear transformation between nodes were used. Main hyparameters chosen are **learning rate** = 1e-4, **batch size**=16, **epochs**=500. The network is trained and tested with different initial conditions and ρ values (10, 28, 35). In Figure 20 and 21 the NN test is compared with the ODE result for an initial condition used during the training, ρ=10 and ρ=28, while in notebook the same network is tested with the same initial condition but ρ=17 and ρ=40. In 22 the absolute error in terms of distance ($\sqrt{(x_{nn} - x_{ode})^2 + (y_{nn} - y_{ode})^2 + (z_{nn} - z_{ode})^2}$) for different values of ρ is showed. As depicted, lower is the ρ values (10, 17) lower is the error, but for these ρ values also the absolute function values are

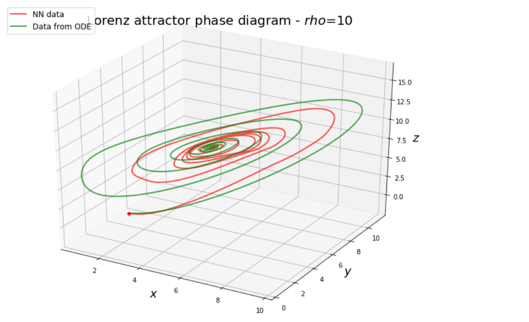

Figure 20: NN vs. ODE Lorenz equation, $\rho = 10$

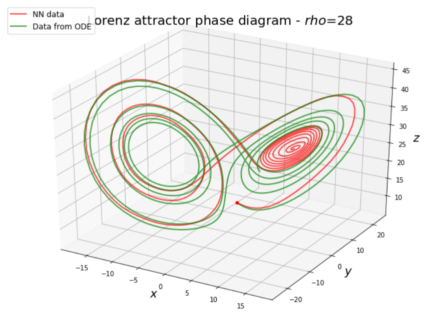

Figure 21: NN vs. ODE Lorenz equation, $\rho = 28$

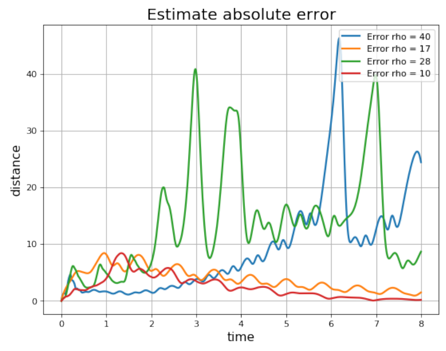

Figure 22: Absolute error NN vs. ODE for Lorenz equation for different ρ values

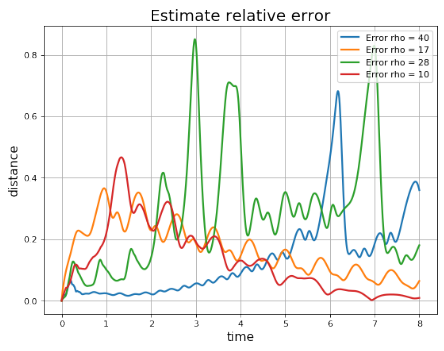

Figure 23: Relative error NN vs. ODE for Lorenz equation for different ρ values

lower. Trying to better understand the error magnitude, distance values are normalize w.r.t. the maximum distance to the origin $max(\sqrt{x_{nn}^2 + y_{ode}^2 + z_{ode}^2})$. The results in Figure 23 show that the indicator's errors decrease with ρ.

5. Summary and Conclusions

Whitin this article different data analysis and system identification techniques were tested. Model reduction technique were first use with DMD. This technique show some difficult to forecast the data tested, but seems to be a good instrument for better understand data components. Optimization techniques applied to Lotka-Volterra shows a bad result. The only hypotesis is that the optimization technique used is not sufficient to reproduce the data. For model identification SINDy is for sure a powerful instrument, but in our test show some critical issue in the hidden variable identification. An hypothesis is that is not able to identify a variable with the same period. Finally NN were used in different cases. For Kuramoto–Sivashinsky dataset weak results were found. The network is able to reproduce the data if the initial condition for the test are the same of the training. For reaction-diffusion dataset the network seems to correctly reproduce a forecast the data. While for Lorenz data the NN correctly reproduce the data just for low ρ values. From our test NN can be used for data regression, but have problems with some type of dataset, in my opinion the chaotic data. All the test and data can be found at *this repository*

6. Bibliography and citations

References

[1] Travis Askham and J. Nathan Kutz. Variable projection methods for an optimized dynamic mode decomposition, 2017.

[2] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.

[3] Daniel Dylewsky, Eurika Kaiser, Steven L. Brunton, and J. Nathan Kutz. Principal component trajectories for modeling spectrally-continuous dynamics as forced linear systems, 2020.