Imperial College London

Use machine learning to extract equations from data

Wen Ze Liu Chen
CID: 01526073
Supervisor: Dr George Papadakis
Submitted on: 13/09/2019

MSc in Advanced Computational Methods for Aeronautics, Flow Management and Fluid Structure Interaction

Abstract

This project studies a machine learning algorithm which is able to reconstruct the unsteady solution of a velocity field from the simulation data. The project can be divided into three stages: data generation, data pre-processing and algorithm implementation. The first stage consists to generate data by running the simulation using a commercial CFD software called Star-CCM+. The flow around the NACA 0012 airfoil, with an incidence of 16 degrees, is simulated with $Re = 600$ for 70 physical time, and 7000 .csv date files were generated. A Reduced Order Model (ROM) is then reconstructed from these simulation data files using the Proper Orthogonal Decomposition (POD). The first four modes are found to be the most energetic, their coefficients are extracted and used as the input to the system identification algorithm known as the Sparse Identification of Nonlinear Dynamics (SINDy). The system of equations for POD coefficients identified by SINDy has 4 terms in total, note the system of equations has 4 equations, so each identified equation contains only one term. Comparison is made between simulation data and results from identified system in time domain, and a frequency study has also been carried out. The average accuracy of the identified system of equations is found to be 91.90% for leaning dataset and 90.62% for testing dataset. It is showed that the identified equations are able to capture the information at an impressively high level of accuracy in both time and frequency domain.

Key words: System identification, sparse regression, symbolic regression, proper orthogonal decomposition, unsteady laminar flow

Acknowledgments

I would like to express my sincerest gratitude to my project supervisor Dr. George Papadakis for giving me such a great opportunity to work on this exciting project. And I really appreciate his invaluable guidance throughout the work on the project.

I would also like to thank my parents for their encouragement and financial support throughout the year of my MSc study.

Contents

Chapter 1

1. Introduction

1.1 Problem definition

Machine learning [1] is one of the hottest topics and also one of the fastest growing research area. The progresses in machine learning and data science [2] promoted the growth in analysis of complex data and have shown a lot of possibilities. However, the fast-growing data science allowed one to interpret analyse the data in a statistical way much easier, but it does not reveal or give the insight of the underlying physical models of the dynamic system. This puts the restriction on data science models to predict the system dynamics outside the sampled data. The extraction of governing equations from data has been very challenging. Because the amount of data tends to be enormous in to order to fully describe a system, it is of great interest to identify automatically what information in data makes a correlation significant and insightful, and this is extremely complicated to be defined algorithmically.

The importance and significance of extracting equations with physical meaning is that it provides a brand-new way to understand and discover the principles behind the systems. Discoveries of the underlying equations promotes the technological developments which change people's lives, some of the most impactful outcomes from technology are aircraft, engine, car and satellite. Very often, one may face a situation in which the data measurements of a system are given but the governing equations are unknown, and these data tend to have a very large size. This is a relatively new research area, the approaches/techniques to identity underlying models are mainly proposed over past few years. There are many works have been done to extract partial differential equations from the data [3-4] and are shown to be computationally efficient and robust. Very recently, data-driven techniques to extract Navier-Stokes equations also have been demonstrated by Raissi et al. [5-6].

In this project, the approach of extracting system dynamic's governing equations from data will be explored and discussed in deep, then the approach will be implemented on simulation data to extract the dynamic of the flow field around an airfoil. The simulation data from which to extract equations will be generated using a commercial Computational Fluid Dynamics (CFD) software. The simulation data will be then processed to obtain a simplified model through a model order reduction technique. After the successful order reduction, the pre-processed data will be used as the input to the algorithm. And finally, the algorithm will be investigated to distil the governing dynamic equations from

simulation data. A discussion will be made on results' accuracy and complexity. The accuracy can be measured by comparing model's results with data ones, and the complexity of model is simply measured by looking at its number of terms.

1.2 Literature Review

1.2.1 Symbolic Regression

About one decade ago, Bongard with Lipson [7] and Schmidt with Lipson [8] have broken through the difficulties, the data have been processed and analysed in a much deeper way which allow to distil the models that are based on physical meanings and are able to give the insight of system's physics. The proposed approach involves the use of symbolic regression [9] which is a method that searches the space of mathematical expressions to match the data. Unlike the traditional linear regression which only search for the parameters to fit a given expression, symbolic regression searches both the parameters and the form of equation. The symbolic regression starts with a competition between various symbolic equations with each other to fit the given data. New symbolic equations forms by incorporating sub-expressions to previously generated equations. The algorithm retains those equations which fit the given data better and eliminates those that gives worse fits. This process is iterated and stops when the error between the data and a specific equation achieves a predefined level. This algorithm tries to balance the complexity of the expression with its accuracy to data, and returns the equation that is most likely to conform with underling physics of the system. However, the method of symbolic regression is very computationally expensive, especially for the system of very large size, it may be very badly scaled and is under the risk of overfitting to the data. To avoid the overfitting, the balance between the expression complexity and predictive ability need to be carefully defined.

1.2.2 Sparse Identification of Nonlinear Dynamics (SINDy)

A new approach known as Sparse Identification of Nonlinear Dynamics (SINDy) was proposed very recently by Brunton, Proctor and Kutz [10-11]. The proposed method uses the idea of sparse regression and compressed sensing. The method took advantage of the fact that there is only a very small amount of relevant terms that govern the system's dynamic. This is the crucial assumption which makes the search space of mathematical expressions becomes sparse, and fortunately, this assumption holds for most physical systems. Progresses in sparse regression and compressed sensing made this method computationally tractable, as they allow to find out which terms of the dynamic system are non-zero, and thus, the governing equation can be defined accurately with the fewest terms possible. Wang, Yang, et al. have predicted catastrophes by using the compressed sensing [12]. Unlike the method based on symbolic regression, SINDy is a more advanced approach which is able to give a parsimonious model that inherently balances the complexity with accuracy, and prevents the overfitting to the data. In the

work of Brunton, et al., the SINDy used a sparse regression technique known as the sequential thresholded least-squares, this technique works better and more efficiently for large data sets. SINDy was successfully implemented on several types of systems with different complexity, including simple canonical system, chaotic Lorenz system, linear & non-linear oscillators and vortex shedding behind an object. The approach/algorithm was demonstrated to be able to extract parsimonious equations from the data that is able describe and predict the dynamics accurately.

1.2.3 Sparse regression

The sparse regression is a type of regression that tries to find a set of optimal projection vectors which contain only a few nonzero entries, and all the rest being zero. Multiplying such projection vector with the matrix which contains the terms' coefficients will discard most of the matrix entries and this avoids overfitting while retaining a certain level of accuracy. This feature of sparse regression is highly desirable for data of large sizes, as it is able to give a parsimonious model without losing a lot accuracy. There exist several techniques that are under sparse regression classification, the following three techniques will be briefly explained in this section: the best subset selection [13-14], the forward stepwise regression [15-16] and the least absolute shrinkage and selection operator (LASSO) first proposed by Tibshirani [17-20].

The idea of best subset selection is quite simple, it performs a brute-force search among all the sparse models with subset size predefined and then evaluates the squared error of each model to select the best one. Although the idea is simple, best subset selection was a theoretically applicable method. It is a nonconvex problem and is known to be a NP-hard problem [21]. Namely it is computationally intractable even for small number of data points and small order of sparse model. However, it has been recently showed by Bertsimas et al. [22] that classical best subset selection problem can be formulated as a mixed integer optimization (MIO) problem. With recent progresses in MIO algorithms, best subset selection problem is now able to solve problem with much larger size.

Forward stepwise regression start with an empty model, could have constant terms but not variable terms yet. Then the model iteratively adds the variable term that best improve the fit to the data until the level of accuracy is achieved. The size of model varies by performing the iteration, and it is important to mention that none of these models with different sizes are generally the globally optimal model. The variable terms are selected by evaluating the squared errors, thus only those variables that are most correlated with the response/data will be added to the model.

LASSO is a method that shrinks the regression coefficients by imposing a penalty function on their size, it is about solving a ℓ_1-norm regularized least square. It combines both variable selection and regularization to search the predictive model, this results in a model with high predictive ability. LASSO

forces the sum of the absolute value of the coefficients to be less than a predefined value, which consequently makes those less correlated coefficients become zero, and then the variable selection is performed in a much smaller subset size. Thus, parsimonious model can be selected without including those less correlated variables. LASSO avoids overfitting to the data and is able to give a model with high level of accuracy. In addition, unlike best subset selection, LASSO is a convex and highly structured problem, so it is computationally tractable.

1.3 Objectives

The principle aim of this project is to extract the dynamic equations of the flow field around an airfoil simulation data, focusing on the technique proposed by Brunton et al. SINDy. The objectives of the project are summarised as follow:

- Research on sequential thresholded least-squares and dynamic system discovery technique SINDy
- Use commercial CFD software package to generate simulation data of the flow filed around an airfoil
- Apply Proper Orthogonal Decomposition (POD) on data to extract the dynamic part of the data
- Develop a code that performs the SINDy algorithm to extract governing equations that is able to predict the dynamic of the velocity field
- Evaluate the performance of the model in terms of its accuracy and its complexity

Chapter 2

2. Simulation and Flow Configuration

2.1 Background

In order to obtain the data from which to extract equations, a simulation is ran around an airfoil to generate the data of flow velocity field. The CFD (Computational Fluid Dynamic) is a commonly used numerical analysis tool to simulate the flow field. The commercial CFD software used in this project is known as Star-CCM+ developed by the computer software company called CD-adapco (Computational Dynamics-Analysis & Design Application Company Ltd). Star-CCM+ is a powerful CFD software able to simulate both 2D and 3D cases. It uses a client-server architecture, which allows the users to solve complex fluid related problems on their own computers, while the expensive computations are done on a remote machine which is much computationally powerful. This project only considers the flow simulation around a 2D airfoil, as the main objective is to extract the equations from a 2D flow field.

The simulation of the flow around an object follows a general process. The first step is to define the geometry of the flow domain, it can be done by either importing a geometry from an external computer aided drawing software or create the geometry directly using the sketching tools provided by Star-CCM+. Once the geometry is created, it needs to be discretised into smaller elements in order to be solved numerically by the solvers. After that, boundary and initial conditions need to be defined properly depending on the flow case. Then, define the desired type of simulation and flow properties. And finally, run the simulation and it will generate the results. The simulation from Star-CCM+ generates the results of many different physical quantities, including velocity, pressure, vorticity, etc. and the results can be either visualised in scalar field or in vector field. As this project aims to extract the dynamic system of the flow field, so only the velocity field is considered.

The flow configuration of the simulation has been set to be the same as the one from previous work done by Zhang and Samtaney [23] at 2006. This allows to have some reference data to compare with and thus verify the correctness of the results. The airfoil used in this existing work was the NACA 0012, the angle of attack was set to be $16°$ and the Reynolds number was set to be around 600. A more detailed explanation of the simulation setup and the process is given is the following sections.

2.2 Simulation setup

2.2.1 Creation of flow domain geometry

The very first step of preparing a simulation is to create the geometry of the flow domain. As mentioned in last section, the same airfoil as the one researched by Zhang and Samtaney [23] in their work, which is the NACA 0012, is used in this project in order to have some results to compare with. To create the flow domain geometry, the data points of the NACA 0012 are first imported into the Star-CCM+ and its surface are used as the inner boundary of the flow domain. The chord of NACA 0012 is set to be 1 and the angle of attack is set to $16°$. The outer boundary of the domain is created by using the embedded sketching tools and it has a shape composed by a semi-circle with a rectangle. Although only the region near the airfoil is interested, the flow domains are always created to be large enough to include the very far region from the airfoil where the velocity equals to freestream velocity, this ensures that the far field boundary condition can be imposed appropriately on the boundaries. The geometry of the flow domain and its dimensions are shown in *Figure 1*.

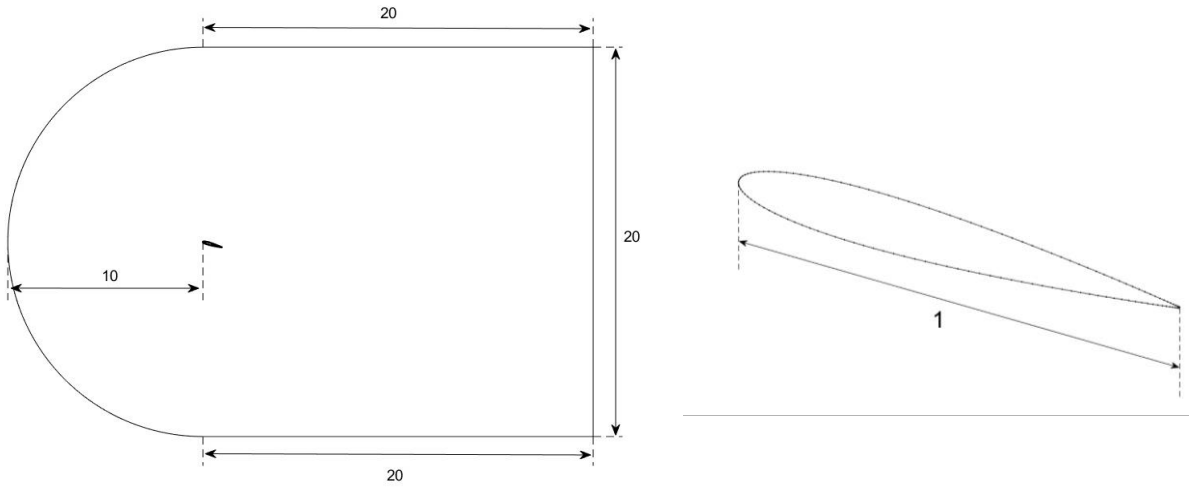


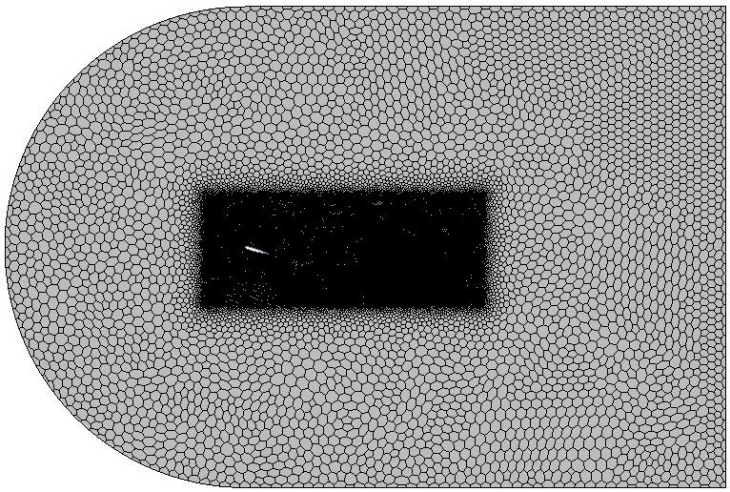
Figure 1: Geometry of flow domain and airfoil

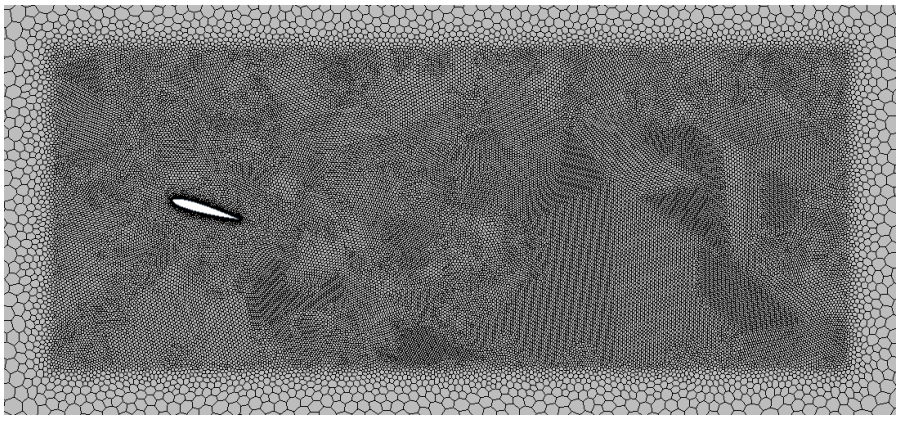
2.2.2 Mesh generation

For the most of fluid mechanic problems, they often involve Partial Differential Equations (PDE) that need to be solved using numerical methods. To solve these differential equations, one has to discretise the flow domain into small elements called mesh cells and then implement numerical methods. The discretisation process of the domain is known as the mesh creation/generation process and it is a very important step in CFD simulations. A bad quality mesh will lead to results of low accuracy, but good quality mesh does not necessarily produce accurate results. An ideal perfect discretised representation for the domain would be a mesh with infinite number of infinitesimally small mesh elements. However,

due to the limitation of computational power in nowadays, using such perfect quality mesh to solve fluid problem is unfeasible. Finer mesh means more costs in computational resource and time, so the balance between the mesh quality and costs need to be considered carefully when perfuming simulations.

The mesh can be classified into three types: structured mesh, unstructured mesh and hybrid mesh. Structured mesh is a mesh with well-structured mesh elements, namely, it has regular connectivity. As opposite to structured mesh, unstructured mesh has irregular connectivity, so the mesh elements are arranged arbitrarily within the flow domain. Hybrid mesh is the mixture of both, that is, some portion of domain is structured mesh and the rest is unstructured. As simulation aims to simulate a laminar viscous flow, so a hybrid mesh composed of prism layer structured mesh with polygonal unstructured mesh has been adopted. The prism layer mesh is a few layers of structured mesh of quadrilateral shape near the surface which catches the viscous behaviour of the fluid, unstructured mesh is unable to do so. The rest of the flow domain are filled with unstructured mesh of polygonal cells, and the region near to the airfoil is refined by using the "custom volumetric control" option, as a result, there are denser mesh cells at this region as shown in *Figure 2*.





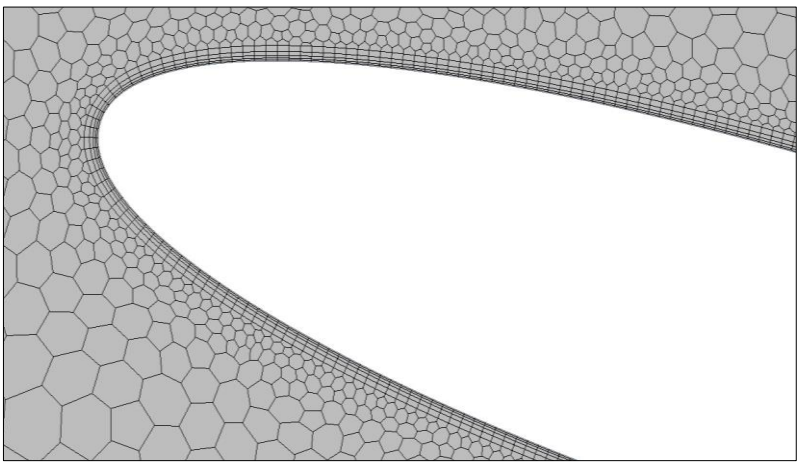


Figure 2: Mesh details of flow domain

The following Table *1* shows the number of cells, the number of interior faces and the number of vertices of the mesh.

Table 1: Specification of the mesh

Parameters	**Values**
Number of cells	53,601
Number of interior faces	157,492
Number of vertices	104,621

2.2.3 Flow configuration

Once the mesh is created, the boundary conditions need to be implemented on the boundaries. The left and right outer boundaries are set to be the inlet and outlet respectively; the upper and lower outer boundaries are set to be symmetric plane; and a wall boundary condition was implemented on the inner boundary which represents the airfoil. As it is aimed to simulate a two dimensional, unsteady, incompressible, viscous and laminar flow, so the inlet freestream velocity is set to be at low speed enough in order to guarantee a laminar flow, a value of 1 was chosen, and it has only the horizontal component.

$$\boldsymbol{u} = (u_\infty, v_\infty) = (1, 0) \tag{1}$$

In order to ensure the flow is laminar, the Reynolds number (Re) need to be around a few hundreds. To accord with the reference data, the Re is intentionally set to be 600. This value of Re can be easily achieved by choosing properly the value of fluid's density and fluid's dynamic viscosity, as it is known that:

$$Re = \frac{\rho u_\infty l}{\mu} \rightarrow 600 = \frac{1 \cdot 1 \cdot 1}{\mu} \rightarrow \mu = \frac{1}{600} \approx 0.00167 \quad (2)$$

Since both the freestream velocity and the chord is set to be 1, the air density is also assumed to be 1 for simplicity. Then it can be deduced that a dynamic viscosity of 0.00167 gives a Re of 600.

The implicit unsteady solver was selected to do the simulation. A second order time discretisation was used in order to guarantee the accuracy of the simulation results. The time step was set to be 0.01 to satisfy the CFL condition which ensures the numerical scheme is convergent and stable. The flow around the airfoil is simulated for 70 physical time unit, long simulation duration will allow flexibility in selecting range for learning data set and testing data set. Each iteration takes 5 internal iterations to converge to the desired level of accuracy, so the whole simulation took 35000 iterations. The flow configurations are summarised in following *Table 2*.

Table 2: Flow configurations

Parameters	**Value**
Solver	Implicit unsteady
Initial condition (m/s)	0
Inlet velocity (m/s)	1
Air density (kg/m^3)	1
Fluid Dynamic viscosity $(Pa \cdot s)$	0.00167
Reynolds Number	600
Time step	0.01
Total physical time (s)	70
Total Iteration steps	35000

2.3 Simulation Results

2.3.1 Data extraction

To extract the desired data from the simulation results in Star-CCM+, one need to create a control volume and then fill the control volume with the measurement points. The measurement points are evenly distributed so that they form a uniform measurement grid. This measurement grid has a local coordinate system, its point of origin corresponds to the leading edge of the airfoil. The extraction of data is performed by creating the "XYZ internal table" in the Star-CC+ and then selecting the measurement grid as the region of extraction. The measurement grid is placed around the airfoil, the data recorded at these points are used to extract the governing equations of the velocity field. The grid has 70×70 points, but note the points inside the airfoil region has been omitted, see *Figure 3.* Since

there are total of 41 points inside the airfoil, so the resulting grid has $70 \times 70 - 41 = 4859$ points that records the data. The extracted data are then all saved in separate .csv files at time step of $\Delta t = 0.01$ dimensionless time unit.

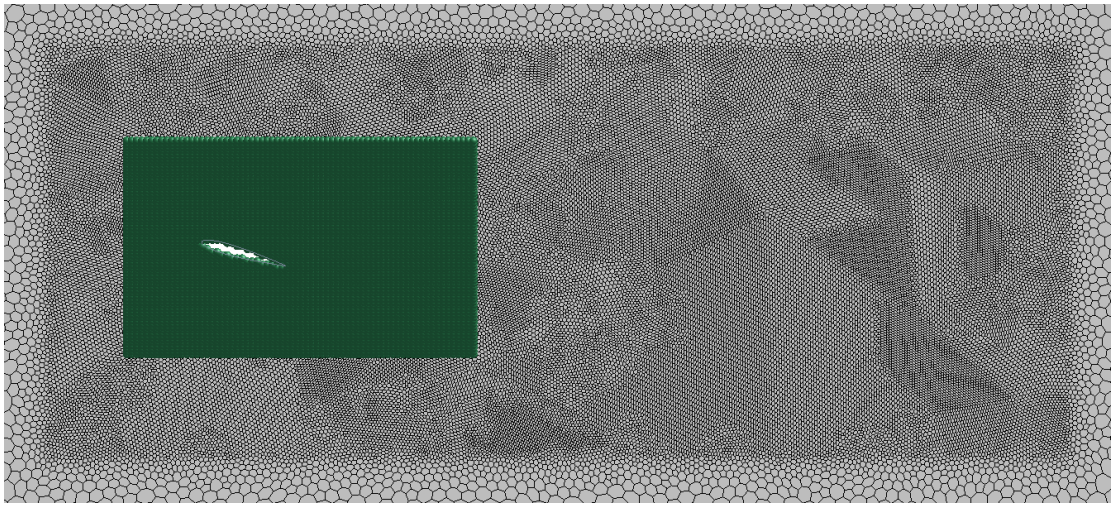
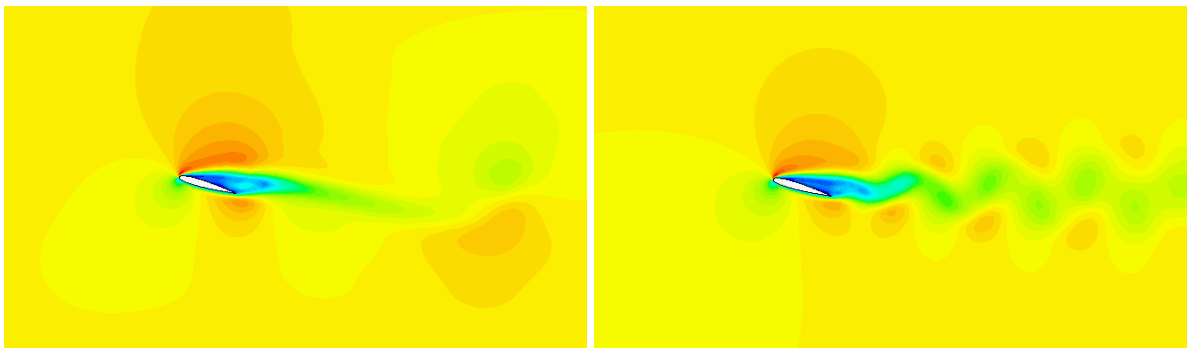


Figure 3: Measurement points around the airfoil

2.3.2 Result visualisation

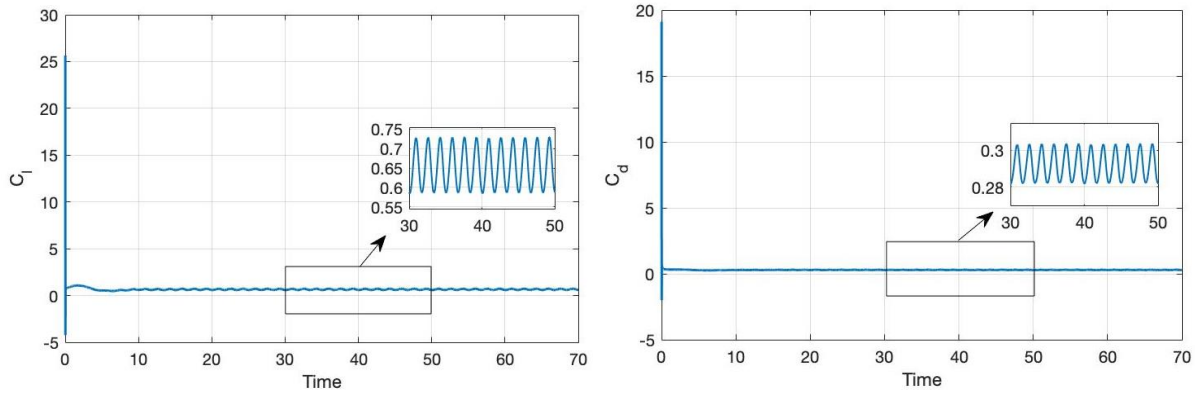
The visualisation of velocity field results has been created in order to make them easier to interpret, see *Figure 4.* As it is an unsteady flow case, the solution does not converge to a specific value, instead it converges to a periodic behaviour and the solution varies with the time. To capture this periodic behaviour of the field, the results are recoded and output by every 0.01 time steps to .csv file. As the total physical times is 70, so there are in total 7000 .csv files generated with each file containing the data of 4859 points around the airfoil. These data are then used to perform the SINDy algorithm to find the governing dynamic equation of the field.



(a) Velocity field at t = 5 (b) Velocity field at t = 50

Figure 4: Velocity field at different time instant

Apart from the velocity filed, the lift and drag coefficient of NACA 0012 airfoil at 16 degrees are also recorded during the simulation, and the results are depicted in *Figure 5*. As can be clearly observed, both lift and drag coefficient show a periodic behaviour after reaching the steady state. The value of lift coefficient at steady state oscillates between 0.59 and 0.73; the drag coefficient oscillates from 0.282 to 0.303. And the time averaged values, at steady state, of C_l and C_d are found to be 0.66 and 0.29 respectively.

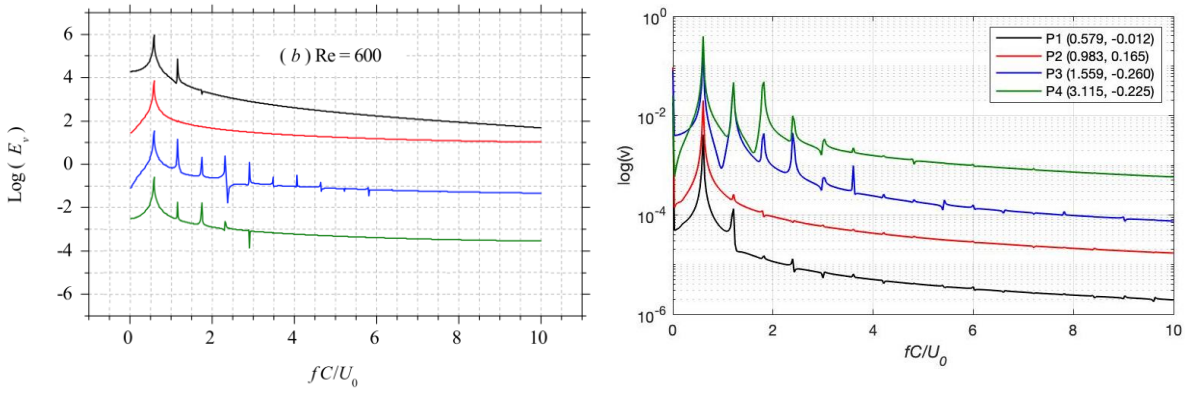


(a) Lift coefficient (b) Drag coefficient

Figure 5: Aerodynamic characteristics of the airfoil.

2.3.3 Result validation

Before make use of the simulation data, they need to be validated to ensure the correctness of themselves. As mentioned in the previous section, the simulation has been set up to have the same configuration as the existing work of Zhang and Samtaney [23] with the aim to have reference data to compare with. The simulation was made on NACA 0012 airfoil at angle of attack of 16 degrees with various Reynolds number. But as the simulation of this project was made with $\mathrm{Re} = 600$, only results from the case of $\mathrm{Re} = 600$ is interested. This research paper includes a plot of frequency domain of the transverse velocity v at five different points on the domain: one is placed above the middle point of airfoil (P1), other is placed above the trailing edge (P2) and the rest are placed in the near and far wake region (P3-P5). The location of the last point (P5) lies outside the region of measurement, thus this point is excluded from the comparison. Note although the coordinates of those points were given in the paper, but the coordinate differs from the one used in this project, so the points at similar locations were chosen in this project and they might be slightly different to those in the paper. The frequency content of the results is shown in the following *Figure 6.*



(a) Frequency plot from Zhang and Samtaney [23] (b) Frequency plot of simulation

Figure 6: Frequency domain of transverse velocity at different locations

Comparing the frequency plot in *Figure 6a* with the one in *Figure 6b*, it can be seen that the positions of the curves from the simulation appears in reverse order to those from the existing work. The reason of this is likely due to the artificial separation of signals in *Figure 6a*. But the positions of the curves are trivial, other characteristics such as the peak position and overall tendency of the frequency response are the factors that need to be compared with. Looking at the two plots in *Figure 6*, the general shape of each curves from simulation results matches quite well with corresponding curves from the paper. While the results from the paper looks smoother and cleaner, the simulation results seem to contain some small noise. The level of noise is negligible and is unlikely to affect the data. Most importantly, the peak locations of simulation result match very nicely with those of paper, which are all situated at about $fC/U_0 = 0.6$. Hence, the similarities of the trends and the nice matching of the peak location are enough to validate the data of velocity field.

Chapter 3

3. Extraction of equations

3.1 Model Order Reduction

Model Order Reduction (MOR) is a technique aims to reduce the dimensional sizes and complexity of models in numerical simulations. MOR is commonly used in different scientific research areas and engineering applications such as fluid mechanics, computational biology, circuit design, etc. It is a very helpful technique that helps to analyse data and simplify the problems. There exist many types of MOR techniques, the most broadly used techniques in these days are mainly classified into following groups based on the method involved:

- Proper orthogonal decomposition methods
- Reduced basis methods
- Balanced truncation
- Dynamic mode decomposition

These MOR techniques is able to extract the parts that contain the most important information from the full model and then construct an approximated model of much lower dimensions, such model is commonly known as the Reduced Order Model (ROM). The full order models of large systems are very complex and inefficient, namely, they tend to be extremely time-consuming and expensive. Thus, in many situations, it is impractical to perform simulations using the full order model because of the limitations in computational power or cost. These are the cases where the ROM comes to be very useful, it is a model with much lower level of complexity and therefore making the simulation more feasible to perform.

It is also very important to mention that ROM does have good and bad quality, a good quality ROM has the following characteristics: be accurate, so it should have very small errors compared to full order model; be conservative, it should be able to conserve the properties and features of the full order model; and most essentially, be computationally efficient and robust. A good quality ROM is highly desirable and is more useful than the full order model in many cases.

Remember from Chapter 2 that the simulation performed in Star-CCM+ lasted 70 physical time units. Excluding the time lasts to settle down to the periodic response, which is about 20 physical time units,

the data of periodic response contains 50 physical time units. And the measurement grid was set to measures and extracts 4859 points at a time step of 0.01 physical time unit, so it is deducted that the output data has a size of 4859×5001, which is a considerably large number. Since the dimensionality of the extracted data is extremely high and it is unfeasible to analyse and identify equations from such amount of data, a MOR is performed to reduce the dimensionality of the data. Among the numerous MOR techniques, the Proper Orthogonal Decomposition (POD) is chosen, which is a very commonly used technique, to create a ROM from the simulation data and thus making the implementation of SINDy algorithm realizable under the limited conditions. The principle behind the POD and its implementation is discussed in detail in following section.

3.2 Proper orthogonal decomposition (POD)

The POD method is widely used across different fields, and depending on the field where it is applied it has different names. For example, it is known as the Singular Value Decomposition (SVD) in Linear Algebra, Empirical Modal Analysis (EMA) in structural dynamics or Empirical Orthogonal Functions (EOF) in meteorological sciences and son on. Although these names look quite different and the formulations might be written in slightly different ways, the principles behind them are all essentially the same.

POD method is one of the most commonly used MOR techniques in fluid mechanics and has also become increasingly popular in fluid mechanics over the recent years, as it is found to be a powerful mathematical tool able to identify coherent structures in turbulent flow. Some interesting previous researches include using POD to separate large eddies in shear flow, done by Lumley [24]; applying POD in vibration systems by Feeny and Liang [25]; and very recently, Kramer and Willcox have implement POD on non-linear problems [26].

POD is a statistical process in which an orthogonal transformation is performed to the data matrix and results in a sum of linearly uncorrelated components in form of vectors, they can be also seen as the sum of weighted basis functions. These resulting vectors are basis set that are orthogonal to each other. POD has an amazing property that makes it a very powerful tool, by performing the POD, the resulting components is automatically arranged in order, with first component being the most energetic one, the second component being the second most energetic, so on and so forth. This very nice property allows one to analyse the original data/model in a very efficient way. Without any further searches, by taking only the first few components, one can easily approximate the original model, and this is extremely helpful in cases where the dimensionality of the data is huge, which appears to be very common.

3.2.1 POD implementation

The decomposition can be carried by two kinds of method, the classical method [24] and the snapshot method [27], both methods should give the equivalent results. There is a paper written by Chen at el. [28] who applied POD using the snapshot method on turbulent flow inside combustion engine. The process of POD implementation carried out here is highly in accordance to this paper with some minor modifications to adapt the project.

Consider a snapshot sequence of velocity field $\boldsymbol{V^{(k)}} = (u, v)_{i,j}^{(k)}$, where k is the snapshot index, i and j are the indices of the grid point in x and y direction respectively. More intuitively, a snapshot can be thinking as the velocity field at a specific instant. As mentioned in last section, the idea of POD is to decompose the snapshots into a sum of linearly independent basis functions, also called POD modes. These modes are orthogonal with each other and each of them is associated with a corresponding POD mode coefficient:

$$\boldsymbol{V^{(k)}} = \sum_{m=1}^{M} a_m^{(k)} \varphi_m \tag{3}$$

where $a_m^{(k)}$ is the coefficient of POD modes and φ_m is the POD mode. Since only the dynamic component of the velocity filed is interested, the mean velocity has been removed from each component before the implementation of POD:

$$U_{dyn} = U - \bar{U} \tag{4}$$

$$V_{dyn} = V - \bar{V} \tag{5}$$

where $\bar{U}$ denotes the time averaged velocity, U_{dyn} and V_{dyn} are the dynamic components of the velocities in x and y directions. For purpose of concision and ease of reading, the dynamic component U_{dyn} and V_{dyn} of velocities are represented using U and V directly in following procedures. The u and v velocity of every grid points at k snapshot is represented by vectors $U^{(k)}$ and $V^{(k)}$. Taking the u velocity as an example, the velocity of every snapshot $\boldsymbol{V^{(k)}}$ can be put together to form a matrix, as shown in following:

$$U^{(k)} = \begin{bmatrix} u_{i=1,j=1}^{(1)} & u_{i=1,j=2}^{(1)} & \dots & u_{i=1,j=Y}^{(1)} & u_{i=2,j=1}^{(1)} & \dots & u_{i=X,j=Y}^{(1)} \end{bmatrix}^T \tag{6}$$

$$U = [U^{(1)} \quad U^{(2)} \quad \dots \quad U^{(k)}] = \begin{bmatrix} u^{(1)}_{i=1,j=1} & u^{(2)}_{i=1,j=1} & \dots & u^{(k)}_{i=1,j=1} \\ u^{(1)}_{i=1,j=2} & u^{(2)}_{i=1,j=2} & \dots & u^{(k)}_{i=1,j=2} \\ \dots & \dots & \dots & \dots \\ u^{(1)}_{i=1,j=Y} & u^{(2)}_{i=1,j=Y} & \dots & u^{(k)}_{i=1,j=Y} \\ u^{(1)}_{i=2,j=1} & u^{(2)}_{i=2,j=1} & \dots & u^{(k)}_{i=2,j=1} \\ \dots & \dots & \dots & \dots \\ u^{(1)}_{i=X,j=Y} & u^{(2)}_{i=X,j=Y} & \dots & u^{(k)}_{i=X,j=Y} \end{bmatrix} \tag{7}$$

The size of this matrix can be easily deducted. Since $U^{(k)}$ represents the velocity at each grid point, and remember from Chapter 2, the measurement has $70 \times 70 = 4900$ deducting 41 points situated inside the airfoil, which gives 4859 points, and thus $U^{(k)}$ is a 4859×1 vector. The snapshots were taken from 20 to 70 with a time steps of 0.01, which is 5001 snapshots in total. So, this a 4859×5001 matrix. The matrix for the v velocity can be easily obtained by following the same procedure. After obtaining both U and V matrix, the spatial correlation matrix of velocity field can be calculated using the following formulation:

$$C = \frac{1}{K}(UU^T + VV^T) \tag{8}$$

where K is the total number of snapshots, U^T and V^T are the transpose matrices of U and V. This spatial correlation matrix contains information from both U and V, and it is the matrix that will be taken to perform the decomposition. Again, the objective of POD is to find out the set of orthogonal basis functions, or more commonly known as POD modes, φ_m and their corresponding coefficients to represent the original model or data set. Mathematically, POD can be described as an optimisation problem:

$$\sum_{k=1}^{K} \left\| \boldsymbol{V^{(k)}} - \sum_{m=1}^{M} a_m^{(k)} \varphi_m \right\|^2 \rightarrow min \tag{9}$$

subject to

$$\begin{gathered} (\varphi_i, \varphi_j) = \delta_{i,j} = \begin{cases} 1 & if\ i = j \\ 0 & if\ i \neq j \end{cases} \\ i, j = 1, 2 \dots k \end{gathered} \tag{10}$$

where $\|\cdot\|$ denotes the L^2 norm and $(\cdot,\cdot)$ denotes the inner product. $\delta_{i,j}$ is the inner product of two modes, it is either 1 or 0, as the inner product of two orthogonal vectors is 0 and inner product of two equal unit vector is 1. The minimization of such problem can be achieved by solving the eigenvalue problem of the spatial correlation matrix C:

$$C\beta_m = \lambda_m \beta_m \tag{11}$$

β_m is the eigenvector and λ_m is the eigenvalue of the cross-correlation matrix. It is also important to mention that if the magnitudes of computed eigenvectors are not of unit length, they need to be normalised to avoid unwanted magnitude amplification in future calculations. The POD modes are then computed by projecting the U and V onto the eigenvector β_m:

$$\varphi_{u_m} = U \cdot \beta_m \tag{12}$$

$$\varphi_{v_m} = V \cdot \beta_m \tag{13}$$

These POD modes extracted from the velocity field are often recognized as the coherent structures, these structures are not necessarily observable in real flows. By definition, each POD mode represents a component of the flow field and the flow field can be reconstructed by summing all these weighted POD modes. The weights of each mode, or equivalently the POD coefficients, are computed by projecting the velocity fields onto the m POD modes:

$$a_{u_m} = U^T \cdot \varphi_{u_m} \tag{14}$$

$$a_{v_m} = V^T \cdot \varphi_{v_m} \tag{15}$$

$$a_m = a_{u_m} + a_{v_m} \tag{16}$$

$$a_m = \begin{bmatrix} a_m^{(1)} \\ a_m^{(2)} \\ \dots \\ a_m^{(k)} \end{bmatrix} = \begin{bmatrix} a_1^{(1)} & a_2^{(1)} & \dots & a_M^{(1)} \\ a_1^{(2)} & a_2^{(2)} & \dots & a_M^{(2)} \\ \dots & \dots & \dots & \dots \\ a_1^{(k)} & a_2^{(k)} & \dots & a_M^{(k)} \end{bmatrix} \tag{17}$$

As can be seen, the coefficient matrix has a dimension of $k \times m$. Note that the row of the coefficient matrix contains the amplitude that the corresponding mode contributes to a specific snapshot, k; and the column contains the amplitude that corresponds to a particular mode, m. Thus, the coefficient matrix a_m is able to provide insightful information on amplitudes of decomposed components from two different perspectives.

At this point, the decomposition of the velocity field is completed. Now back to the purpose of applying POD, an approximated model with reduced dimensionality is desired. This can be achieved by selecting the minimum number of modes to reconstruct the original full order model, the optimal number of modes is selected by looking at the energy level that each mode contains. The energy contributed by the mth mode at the kth snapshot is given by $\frac{1}{2}\left(a_m^{(k)}\right)^2$, and similarly, the kinetic energy by unit mass of mth mode from all snapshots is:

$$KE_m = \frac{1}{2}\sum_{k=1}^{K}\left(a_m^{(k)}\right)^2 \tag{18}$$

where

$$KE_m = \frac{KE}{\rho} = \frac{1}{2}\boldsymbol{V}^2 \tag{19}$$

To better interpret and facilitate the comparison of energy content among all modes, it is more convenient to look at the energy fraction of mth mode:

$$ke_m = \frac{KE_m}{\sum_{k=1}^{M} KE_m} \tag{20}$$

As briefly mentioned in previous section, the decomposition automatically arranges the modes in order of decreasing energy content. It is very often that the first few modes contain a great portion of total energy, and the rest are negligible. Thus, it is often enough to take a truncated set of modes to reconstruct the model, which results in a significant lower complexity model comparing to the original full order model. To reconstruct a ROM of velocity field at kth snapshot:

$$\boldsymbol{V_{ROM}} = \overline{\boldsymbol{V}} + \sum_{m=1}^{M} a_m^{(k)}\varphi_m \tag{21}$$

where $\overline{\boldsymbol{V}}$ is the mean velocity field which was deducted at the beginning and M is a small number of modes taken to reconstruct the ROM. The time-averaged velocity field can be easily computed through the time-average POD coefficients and the modes:

$$\langle V \rangle = \frac{1}{K}\sum_{k=1}^{K} V^{(k)} = \sum_{m=1}^{M}\left[\frac{1}{K}\sum_{k=1}^{K} a_m^{(k)}\right]\varphi_m = \sum_{m=1}^{M}\langle a_m \rangle^{(K)}\,\varphi_m \tag{22}$$

where $\langle a_m \rangle^{(K)}$ denotes the POD coefficients averaged over all snapshots. The same as for the velocity field, the ROM of time-averaged velocity field can be obtained by truncating the modes that have lower or negligible energy content; and the exact velocity field can be obtained by simply summing all the weighted modes.

3.3 System Identification of Nonlinear Dynamic (SINDy)

As outlined in Chapter 1, SINDy is an algorithm proposed by Brunton et al. [10] that identifies the governing equation based on the data that are collected from experiments or simulations. This algorithm

involves the use of sparse regression technique such as LASSO, or in the case of this paper, a technique called sequential thresholded least-squared. The implementation of SINDy algorithm will follow the process of the exiting work done by Brunton et al. with small modifications to adapt this project. Note only the dynamic part of the simulation data is interested, so the coefficient matrix a_m, which contain the dynamic information of the velocity field, is used as the input of the algorithm. The input data matrix a_m is further divided into learning dataset and testing dataset as mentioned in section 2.2.3, the learning dataset is used as the input of SINDy to identify the governing equations, and the testing dataset is used to validate the results predicted from the identified equations. More specifically, the snapshot range $[2000, 4000]$ is used as learning data and snapshot range $[4001, 7000]$ for testing data.

3.3.1 SINDy algorithm

Consider the following dynamical system:

$$\frac{d}{dt}\boldsymbol{a}(t) = f\big(\boldsymbol{a}(t)\big) \tag{23}$$

$$\boldsymbol{a}(t) = \begin{bmatrix} a_1(t) \\ a_2(t) \\ \dots \\ a_m(t) \end{bmatrix} \tag{24}$$

where $\boldsymbol{a}(t)$ is the vector that contains POD coefficients of up to mth mode at time t, and $f\big(\boldsymbol{a}(t)\big)$ is the nonlinear function that defines the dynamic of $\boldsymbol{a}(t)$, which is the expression wanted to be identified by the algorithm. It was found by Brunton et al. that for many existing physical systems, the nonlinear function f usually contains only a few terms. This implies high sparsity in space of possible functions, most of the functions contain a null coefficient. Advances in sparse regression allow one to find which equation terms are non-zero without having to perform brute-force search, which is time consuming and often unfeasible under limited computational resource.

To determine the function $f\big(\boldsymbol{a}(t)\big)$, the data of $\boldsymbol{a}(t)$ at different time t and its derivative $\dot{\boldsymbol{a}}(t)$ is required. The derivative of the variable can be either measured from the experiments or approximated from $\boldsymbol{a}(t)$. Here, $\dot{\boldsymbol{a}}(t)$ is approximated numerically using the fourth order central finite difference scheme:

$$\left.\frac{da}{dt}\right|_{t_i} = \frac{-a_{i+2} + 8a_{i+1} - 8a_{i-1} + a_{i-2}}{12\Delta t} \tag{25}$$

The POD coefficients $a(t)$ and its derivative $\dot{a}(t)$ at different times $t_1, t_2, \dots, t_k$ is then arranged as the following form:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a^T}(t_1) \\ \boldsymbol{a^T}(t_2) \\ \dots \\ \boldsymbol{a^T}(t_k) \end{bmatrix} = \begin{bmatrix} a_1(t_1) & a_2(t_1) & \dots & a_m(t_1) \\ a_1(t_2) & a_2(t_2) & \dots & a_m(t_2) \\ \dots & \dots & \dots & \dots \\ a_1(t_k) & a_2(t_k) & \dots & a_m(t_k) \end{bmatrix} \tag{26}$$

$$\dot{\boldsymbol{A}} = \begin{bmatrix} \dot{\boldsymbol{a}}^{\boldsymbol{T}}(t_1) \\ \dot{\boldsymbol{a}}^{\boldsymbol{T}}(t_2) \\ \dots \\ \dot{\boldsymbol{a}}^{\boldsymbol{T}}(t_k) \end{bmatrix} = \begin{bmatrix} \dot{a}_1(t_1) & \dot{a}_2(t_1) & \dots & \dot{a}_m(t_1) \\ \dot{a}_1(t_2) & \dot{a}_2(t_2) & \dots & \dot{a}_m(t_2) \\ \dots & \dots & \dots & \dots \\ \dot{a}_1(t_k) & \dot{a}_2(t_k) & \dots & \dot{a}_m(t_k) \end{bmatrix} \tag{27}$$

After obtaining these two matrices, a library $\boldsymbol{\Theta}(\boldsymbol{A})$ that contains the candidate functions of the columns of $\boldsymbol{A}$ matrix need to be created. The candidate functions could be either linear or nonlinear, so the library $\boldsymbol{\Theta}(\boldsymbol{A})$ may have constant, polynomials of different orders and trigonometric functions, as shown in following:

$$\boldsymbol{\Theta}(\boldsymbol{A}) = \begin{bmatrix} | & | & | & | & | & | & | & | & | & | \\ \boldsymbol{1} & \boldsymbol{A} & \boldsymbol{A}^{P_2} & \boldsymbol{A}^{P_3} & \dots & \sin(\boldsymbol{A}) & \cos(\boldsymbol{A}) & \sin(2\boldsymbol{A}) & \sin(2\boldsymbol{A}) & \dots \\ | & | & | & | & | & | & | & | & | & | \end{bmatrix} \tag{28}$$

where $\boldsymbol{A}^{P_2}$ and $\boldsymbol{A}^{P_3}$ denotes the second and third order polynomials respectively. This notation is used in accordance with the paper of Brunton et al., as an example, the quadratic polynomial $\boldsymbol{A}^{P_2}$ is given by following matrix:

$$\boldsymbol{A}^{P_2} = \begin{bmatrix} {a_1}^2(t_1) & a_1(t_1)a_2(t_1) & \dots & {a_2}^2(t_1) & a_2(t_1)a_3(t_1) & \dots & {a_m}^2(t_1) \\ {a_1}^2(t_2) & a_1(t_2)a_2(t_2) & \dots & {a_2}^2(t_2) & a_2(t_2)a_3(t_2) & \dots & {a_m}^2(t_2) \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ {a_1}^2(t_k) & a_1(t_k)a_2(t_k) & \dots & {a_2}^2(t_k) & a_2(t_k)a_3(t_k) & \dots & {a_m}^2(t_k) \end{bmatrix} \tag{29}$$

Each column of the library matrix $\boldsymbol{\Theta}(\boldsymbol{A})$ represents a potential function for the right-hand side of the dynamical system defined at the beginning of this section 3.3.1 $f\big(\boldsymbol{a}(t)\big)$. There are no restrictions on what kind of candidate functions are allowed in library matrix, any kind of nonlinearities can be included in this library, and depending on type of system to be identified, the libraries may differ a lot. Based on the existing physical systems, it is anticipated that most of the nonlinearities in library would be zero and only a few nonlinearities will be active. Thus, a sparse regression problem can be set up to find out the sparse vector $\boldsymbol{\Xi} = [\boldsymbol{\xi_1} \quad \boldsymbol{\xi_2} \quad \dots \quad \boldsymbol{\xi_m}]$ that determines which nonlinearities are active and the corresponding coefficients of these active terms. The sparse regression problem is defined as:

$$\dot{\boldsymbol{A}} = \boldsymbol{\Theta}(\boldsymbol{A})\boldsymbol{\Xi} \tag{30}$$

In above equation, both $\dot{\boldsymbol{A}}$ and $\boldsymbol{\Theta}(\boldsymbol{A})$ is known, and $\boldsymbol{\Xi}$ is the sparse matrix to be determined by sparse regression methods. Each column $\boldsymbol{\xi_m}$ of $\boldsymbol{\Xi}$ determines which determines the active terms in the right-

hand side of $\frac{d}{dt}a_m(t) = f_m(a_m(t))$. Once the sparse matrix is determined, each equation of the dynamical system can be constructed using the formulation below:

$$a_m(t) = f_m(a_m(t)) = \boldsymbol{\Theta}(\boldsymbol{a}^T)\boldsymbol{\xi_m} \tag{31}$$

where $\boldsymbol{\Theta}(\boldsymbol{a}^T)$ is a library matrix that contains the symbolic functions of variable $\boldsymbol{a}$, note it is a completely different thing to $\boldsymbol{\Theta}(\boldsymbol{A})$, which is a matrix that contains numerical values. In other words, $\boldsymbol{\Theta}(\boldsymbol{A})$ is used to perform the sparse regression and $\boldsymbol{\Theta}(\boldsymbol{a}^T)$ is used to output the expression of governing equation found. A more generic expression for the model is given by:

$$\frac{d}{dt}\boldsymbol{a} = \boldsymbol{f}(\boldsymbol{a}) = \boldsymbol{\Xi}^T\left(\boldsymbol{\Theta}(\boldsymbol{a}^T)\right)^T \tag{32}$$

It is also important to mention that when solving the sparse regression problem, the columns of the candidate function library $\boldsymbol{\Theta}(\boldsymbol{A})$ may need to be normalized to ensure the sparse regression can be performed correctly. Especially when the entries of $\boldsymbol{A}$ is very small, as the powers of $\boldsymbol{A}$ will be an extremely small number. A schematic of SINDy algorithm implementation on a Lorenz system is illustrated below in *Figure 7*, it gives a more intuitive view on how the algorithm works and helps to better understand the algorithm.

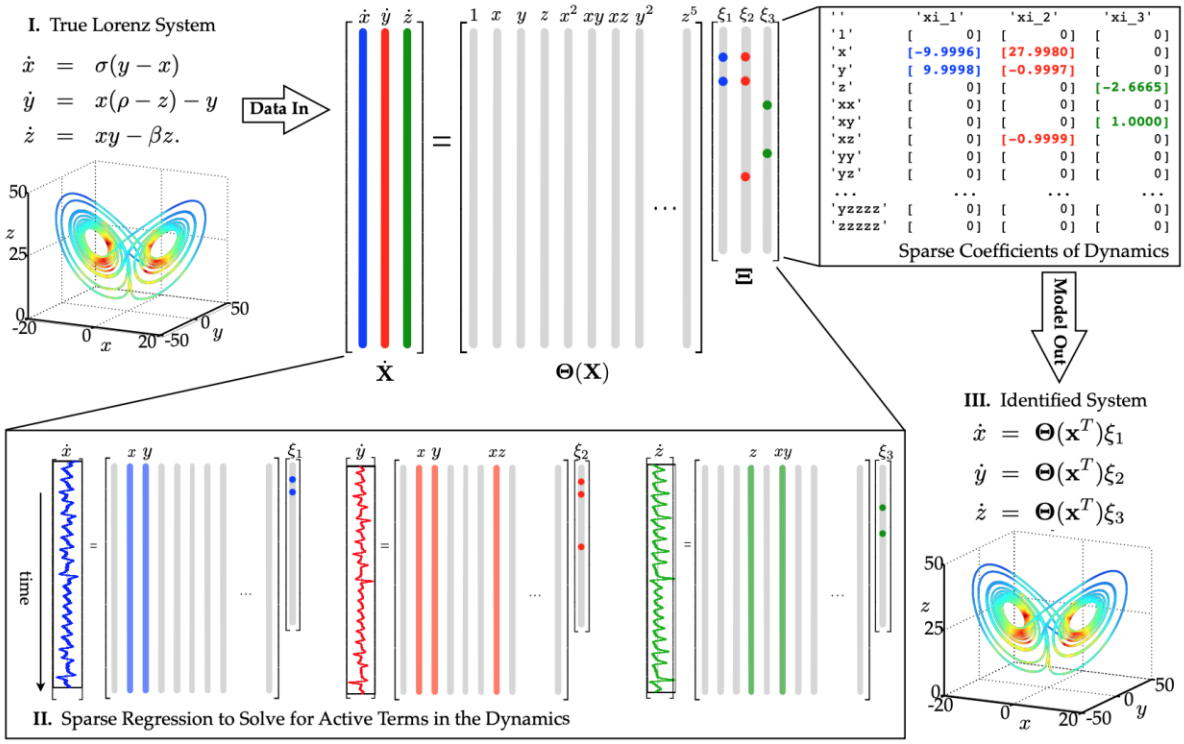


Figure 7: SINDy algorithm on chaotic Lorenz System (Brunton et al. [10])

3.3.2 Sequential thresholded least-squares

In the previous section, it was mentioned that a sparse regression need to be carried out in order to determine the sparse vector $\boldsymbol{\Xi}$. A sparse regression method known as the sequential thresholded least-squares is implemented in the algorithm proposed by Brunton et al. As the name indicates, it is a method based on least-squares regression, it is actually an extension of least-square with incorporation of a new parameter λ, thresholding value, which allows one to have a sparse solution. This method is computationally efficient and therefore it is very suitable for large size data.

The method is fairly simple, it starts by solving the sparse regression problem defined in equation 30 using a normal least square regression to get the coefficient matrix $\boldsymbol{\Xi}$. The entries of the coefficient matrix $\boldsymbol{\Xi}$ that are below the specified thresholding value λ are then zeroed. Each matrix entry has a matrix index that represents its corresponding position in the matrix, after identifying the indices of non-zero entries, another least-square is performed on the entries of identified indices. The new coefficient matrix $\boldsymbol{\Xi}$ will become sparser and the matrix elements are thresholding again using the value λ. This process is iterated again and again until the non-zero entries of the matrix $\boldsymbol{\Xi}$ converge.

The thresholding value λ is a key parameter which has a direct influence on the complexity and accuracy of identified equations. Larger value of λ results in a simpler but less accurate system; in opposite, smaller value of λ gives more accurate but also more complicated system, and may have the risk of overfitting the data. Thus, the value of λ need to be carefully chosen to balance the complexity and accuracy of the system. Note there is no restriction on which sparse regression should be used to determine the sparse coefficient matrix, other sparse regression methods are also allowed and can be easily adapted into the SINDy algorithm without any problem.

Chapter 4

4. Results and discussion

4.1 POD Results

4.1.1 Energy fraction

The principles behind the POD and its implementation process is explained in detail in Chapter 3, and the corresponding MATLAB code is also included in the Appendix A.1. In this section, the results from POD as well as the ROM of the velocity will be visualized to give a more intuitive way of understanding them. First of all, the fraction of energy content in each mode (coherent structure), computed using the equation 20 in section 3.2.1, is presented in following:

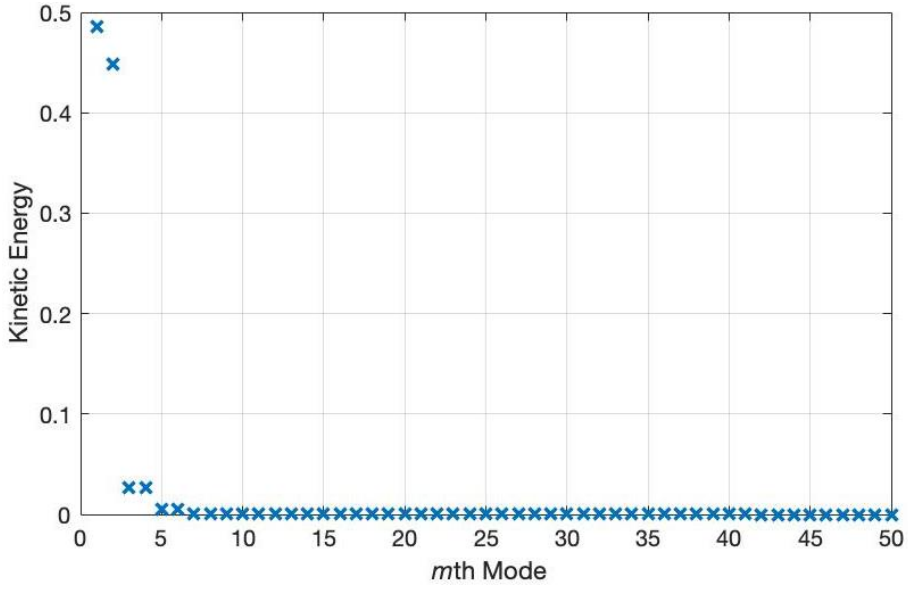


Figure 8: Kinetic energy fraction in each POD mode

The numerical values of energy fraction in first 10 modes are shown in the *Table 3* below:

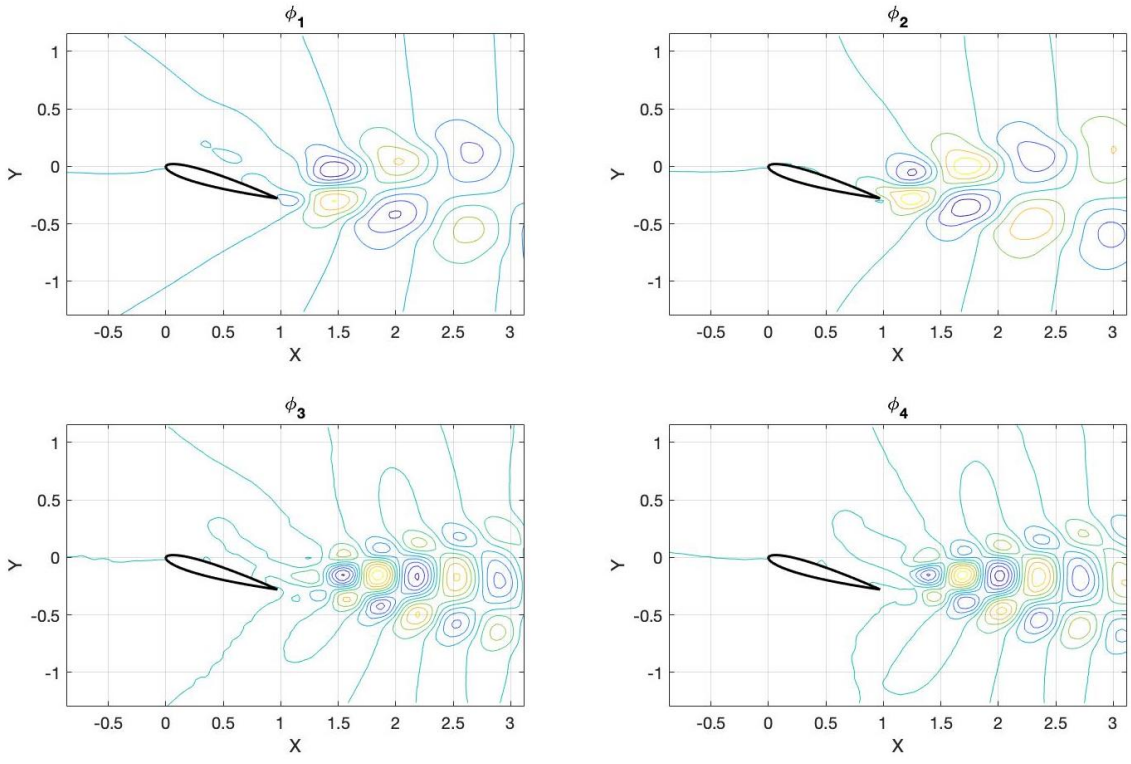
Table 3: Kinetic energy fraction in first 10 modes

***m*th Mode**	**KE fraction**	***m*th Mode**	**KE fraction**
1^{st}	0.486	6^{th}	0.0052
2^{nd}	0.449	7^{th}	$6.285 \cdot 10^{-4}$
3^{rd}	0.027	8^{th}	$6.166 \cdot 10^{-4}$
4^{th}	0.026	9^{th}	$7.485 \cdot 10^{-5}$
5^{th}	0.0055	10^{th}	$7.314 \cdot 10^{-5}$

As can be observed from the *Figure 8*, the modes are ordered in decreasing amount of energy, with the first mode carrying the highest amount of energy. It can be easily calculated from *Table 3* that about 98.8% of total energy is carried by the first 4 modes. This implies that a ROM constructed using first 4 modes is able to capture nearly all the information of dynamic of the original full order model. It is also interesting to point out that the all POD modes come in pairs, the energy content of each pair is within the same order of magnitude and the energy content of mode pairs decreases in order of 10^{-1}.

4.1.2 POD modes

The first 10 modes of u velocity component are visualised to help to interpret results in a more intuitive way. These 10 modes contain about 99.997% of total energy as well as the dynamics of the velocity field.



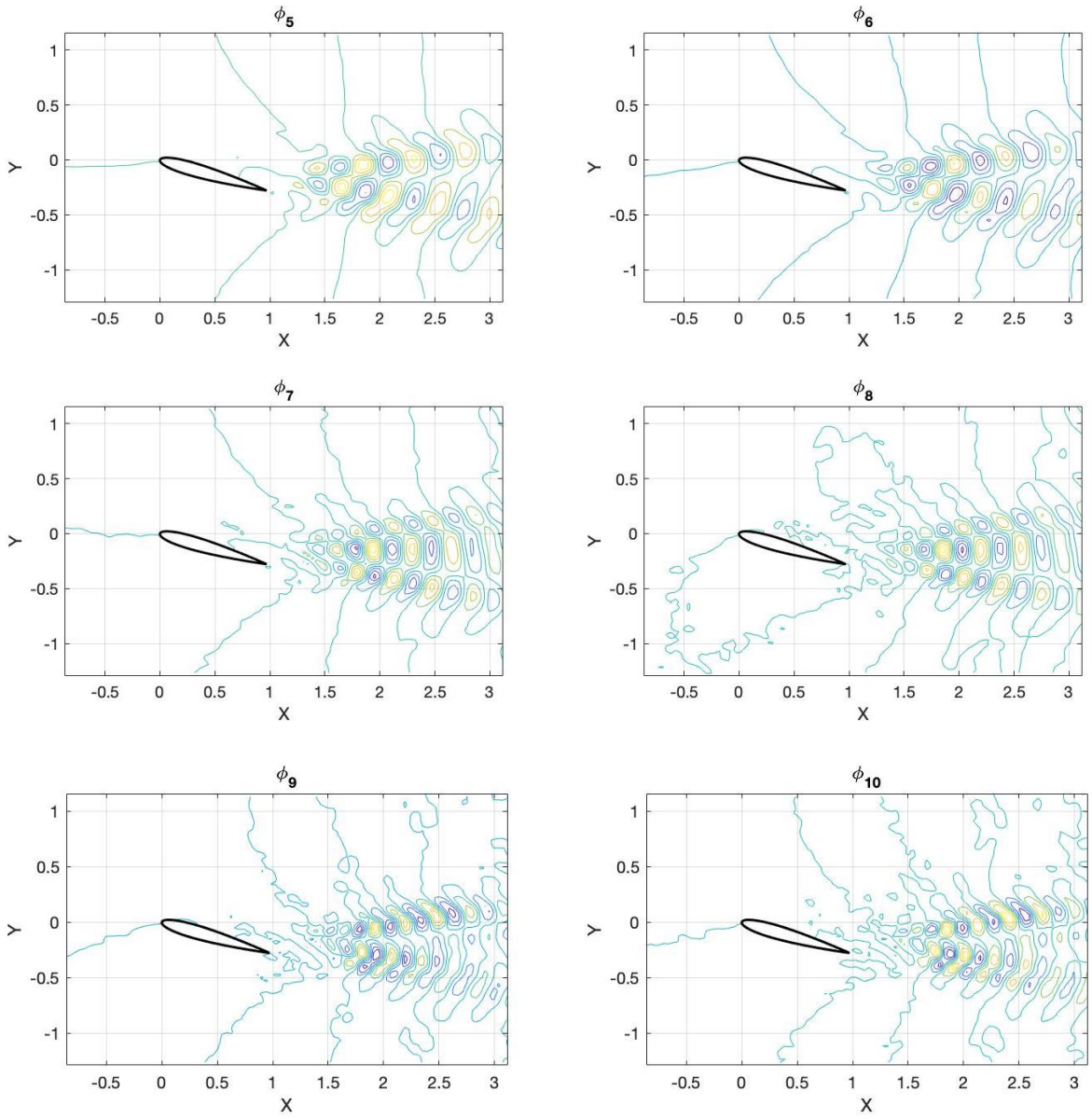


Figure 9: Visualisation of first 10 modes

As can be seen from the *Figure 9*, the pattern of the modes appears in pairs, each pair has their corresponding pattern. The first 2 modes are the most dynamically and energetically important feature of the velocity field, and they clearly capture the most dominant feature of the original flow field, the vortex shedding in the wake region. It can be also easily observed the frequency variation of the mode pattern as the mode number increases, the frequency of the first mode pair's pattern is the lowest, and the frequency becomes higher for the afterwards mode pairs. Remember that although incorporating more modes in a ROM will result in a more accurate model, the contribution from the modes beyond the fourth mode are negligible comparing to the first 4 modes, thus it is unnecessary to incorporate too many modes when constructing a ROM.

4.1.3 ROM construction

According to the kinetic energy fraction shown in *Table 3*, the first 4 modes should be able to capture about 98.8% of total energy. A ROM of velocity field has been reconstructed using these 4 modes and will be used to compare with the original model to proof the quality of the ROM. Since the POD was implemented separately onto the u velocity and v velocity, the ROM of the velocity magnitude field can be reconstructed via the ROM of u and v velocity field:

$$\boldsymbol{V_{ROM}} = \sqrt{\boldsymbol{u_{ROM}^2} + \boldsymbol{v_{ROM}^2}} \tag{33}$$

where

$$\boldsymbol{u_{ROM}} = \overline{\boldsymbol{u}} + \sum_{m=1}^{4} a_m^{(k)} \varphi_{u_m} \tag{34}$$

$$\boldsymbol{v_{ROM}} = \overline{\boldsymbol{v}} + \sum_{m=1}^{4} a_m^{(k)} \varphi_{v_m} \tag{35}$$

Note that there are in total 5001 snapshots of the velocity field representing 5001 instants of velocity field between $t = 20$ to $t = 50$, the ROM should be able to match with original velocity field at any instants. Since the mean velocity was deducted before performing the POD, it needs to be added back to ensure the magnitude of velocity field is reconstructed correctly.

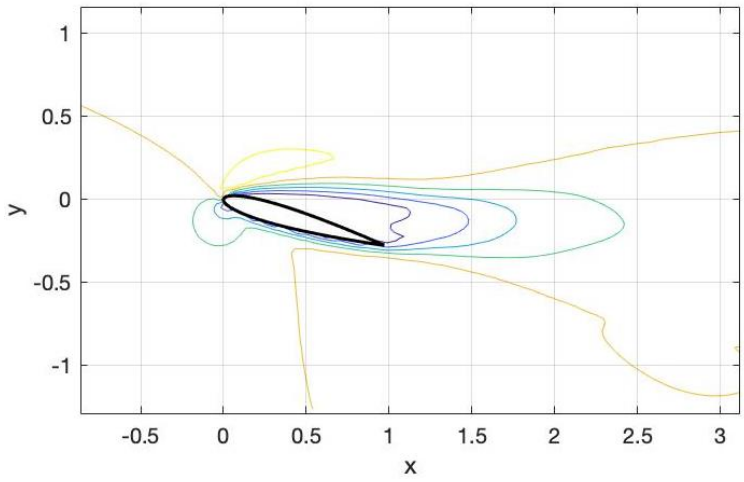
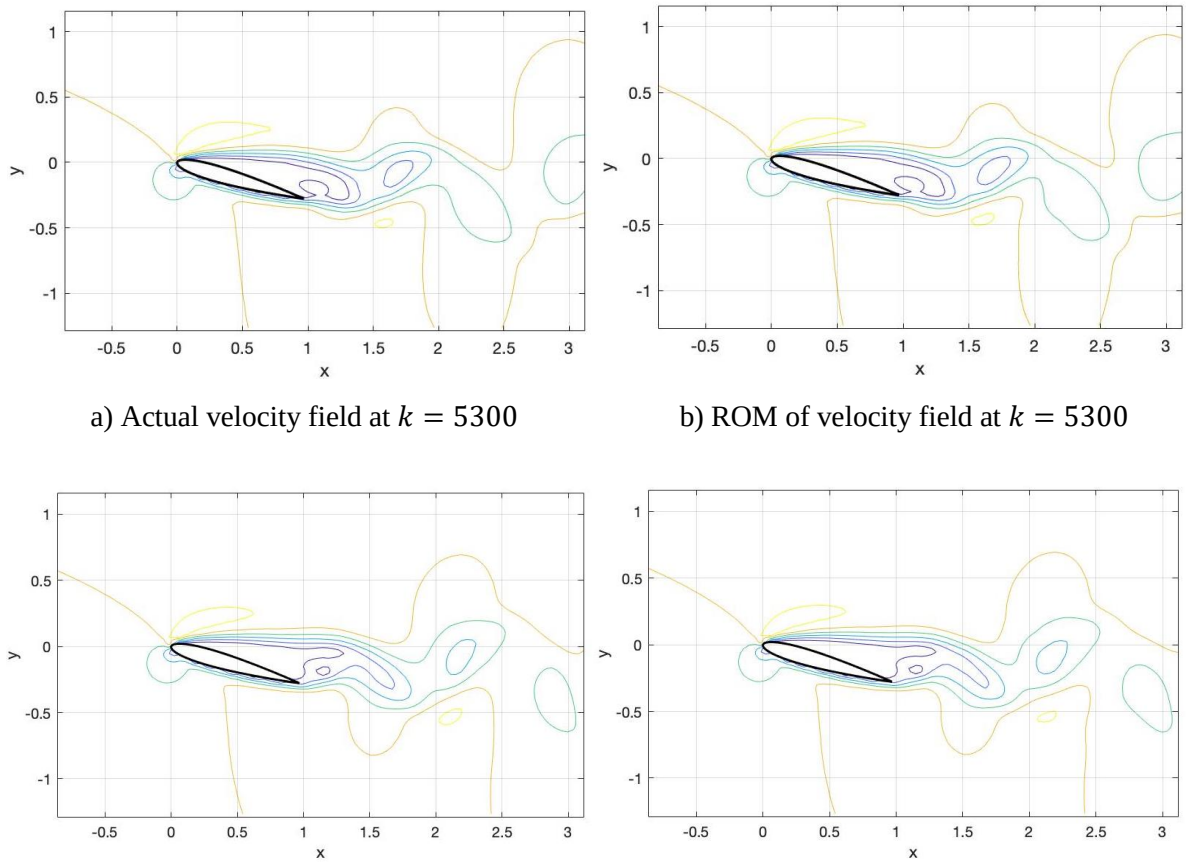


Figure 10: Mean velocity field

Figure 10 shows the mean velocity (time-averaged) field. Once the ROM is reconstructed, the velocity field at two arbitrary snapshots $k = 5300$ and $k = 6700$ are chosen to compare with the original full order velocity field at the same snapshots.



a) Actual velocity field at $k = 5300$

b) ROM of velocity field at $k = 5300$

c) Actual velocity field at $k = 6700$

d) ROM of velocity field at $k = 6700$

Figure 11: Comparison of velocity field

From four plots in *Figure 11*, it can be seen that the difference between actual velocity field and ROM is negligible at two chosen time instants, this implies the ROM of four modes is able to recover the original velocity field at a high accuracy and good quality. As mentioned in Chapter 3, it is desirable to keep the number of modes to a minimum, since this can significantly reduce the complexity of the model. And while the use of ROM makes the implementation of SINDy algorithm on the velocity field computationally feasible, the accuracy of the model is also guaranteed.

4.2 SINDy Results

4.2.1 Equation extraction

The code of SINDy implementation on the coefficient matrix a_m is attached in the Appendix A.2. This code is developed based on the code provided by Brunton et al. [10], modification has been made to adapt the input matrices, as data are divided into learning dataset and testing dataset as mentioned in section 3.3. It was discussed in the Chapter 3 that the sequential thresholding method used in SINDy has a key parameter, thresholding value λ, that determines the complexity and the accuracy of the

resulting governing equations. Several values of thresholding value have been tried to see its effect on the resulting equations from learning dataset. The number of active terms in identified set of equations for a_m and its error with respect to actual a_m value using different value of λ are compiled in the following table:

Table 4: Complexity and accuracy of identified system of equations using different λ

λ	Number of terms	Error	λ	Number of terms	Error
0.1	32	18.41	**0.26**	13	2.97
0.12	30	-	**0.28**	13	2.97
0.14	25	3.99	**0.30**	13	1.95
0.16	25	3.99	**0.32**	13	1.95
0.18	25	3.99	**0.34**	4	0.46
0.20	24	3.91	**0.36**	4	0.46
0.22	13	2.97	**0.38**	4	0.46
0.24	13	2.97	**0.40**	4	0.46

where the error is defined in following way:

$$ERROR = \sqrt{\sum_{i=1}^{N} \frac{\left(a_{m_n} - \tilde{a}_{m_n}\right)^2}{N}} \tag{36}$$

N is the total number of elements in a_{m_n} matrix, and $\tilde{a}_{m_n}$ is the resultant coefficient matrix from equations identified by SINDy algorithm from the learning dataset. The omitted value in error column represents that a_m cannot be computed from the identified equations because of the singularities found when solving Ordinary Differential Equations (ODE) in MATLAB. Observing *Table 4*, both the number and the error decreases when increasing the thresholding value, and both converged to a fixed value. The relationship between complexity and accuracy is more straight forward then expected, instead of having a convex relation with an optimal value at the bottom, they converge to a minimum (which is also optimal) value directly. Any thresholding value above 0.34 gives the identical system of equations, variation in λ does not produce influences anymore in the resulting system. No reason has been found for this unexpected and much simpler behaviour so far. However, the optimal value of λ can be easily chosen in this case, any value above 0.34 should result in the same identified system of equations for a_m, hence $\lambda = 0.34$ is chosen trivially. The systems of equations extracted from data using SINDy is then shown as following:

$$\dot{a}_m = \begin{cases} \dfrac{da_1}{dt} = 3.93 \cdot a_2 \\ \dfrac{da_2}{dt} = -3.63 \cdot a_1 \\ \dfrac{da_3}{dt} = 7.66 \cdot a_4 \\ \dfrac{da_4}{dt} = -7.45 \cdot a_3 \end{cases} \tag{37}$$

The systems of equations extracted are unexpectedly simple, all of the equations contain only one variable with a constant positive or negative coefficient. If looking more carefully in these equations, a very interesting relationship between POD coefficients is found, the dynamic of $\dot{a}_1$ is constrained only by a_2 and the dynamic of $\dot{a}_2$ is constrained by a_1, which happens the same in $\dot{a}_3$ and $\dot{a}_4$. This means that the system is decoupled, that is, each mode can only influence its own pair mode but cannot affect modes in other pairs. The mode pairs oscillate independently from each other because the system has reached a limit cycle. In addition, each pair has a positive and negative coefficient, and the absolute values of the coefficients' value for each pair are very close to each other. Since the a_m behaves periodically, the opposite sign implies that a_m within a pair has a phase difference of 90 degrees.

4.2.2 Results visualisation

The results from extracted equations are compared with those from learning data in following *Figure 12*.

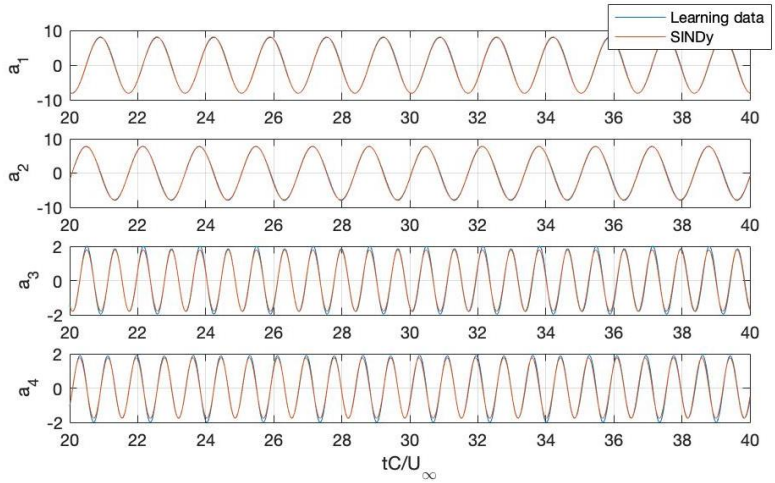


Figure 12: Comparison between coefficients from learning dataset and from SINDy

As can be observed from the plots in *Figure 12*, the results from the identified model match surprisingly well with the learning data, particularly for the POD coefficients of the first two modes. Although the peaks of a_3 and a_4 are not matched perfectly, the results from SINDy is able to capture their dynamics

to a satisfactory level. Three arbitrary points are chosen for comparison of u and v velocities, coordinates of these three points are $P1$: $[0.637, -0.0123]$, $P2$: $[1.04, 0.165]$, $P3$: $[1.617, -0.261]$.

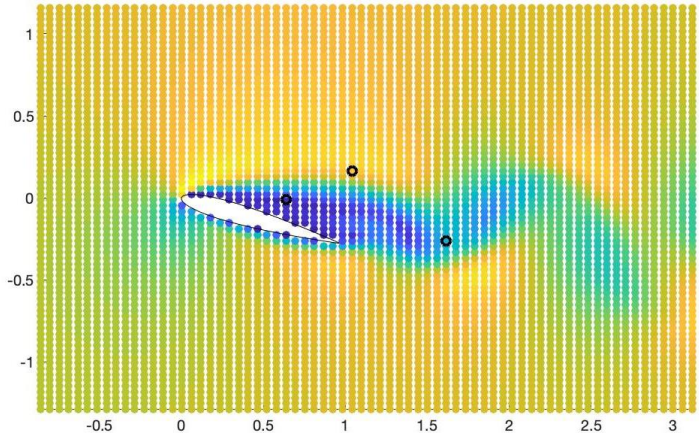
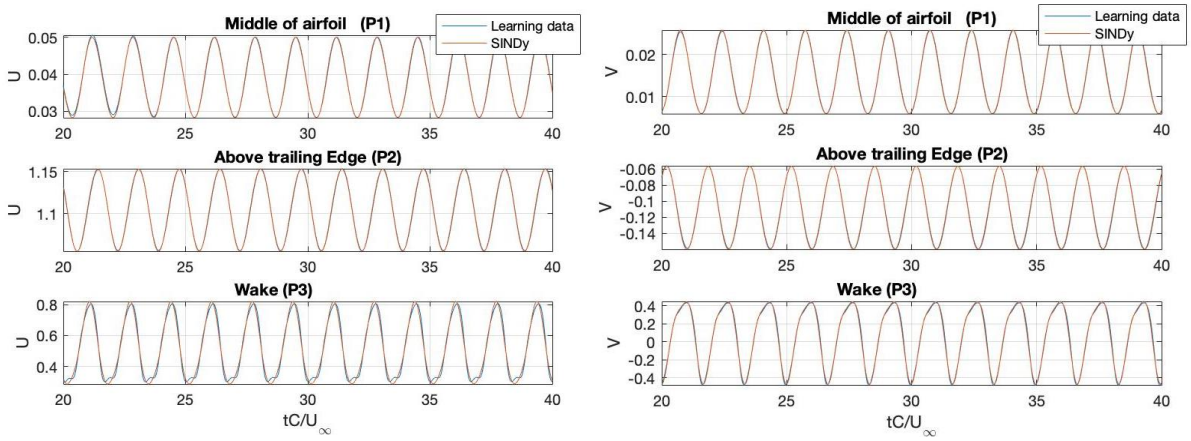


Figure 13: Point locations, from left to right: $P1, P2, P3$

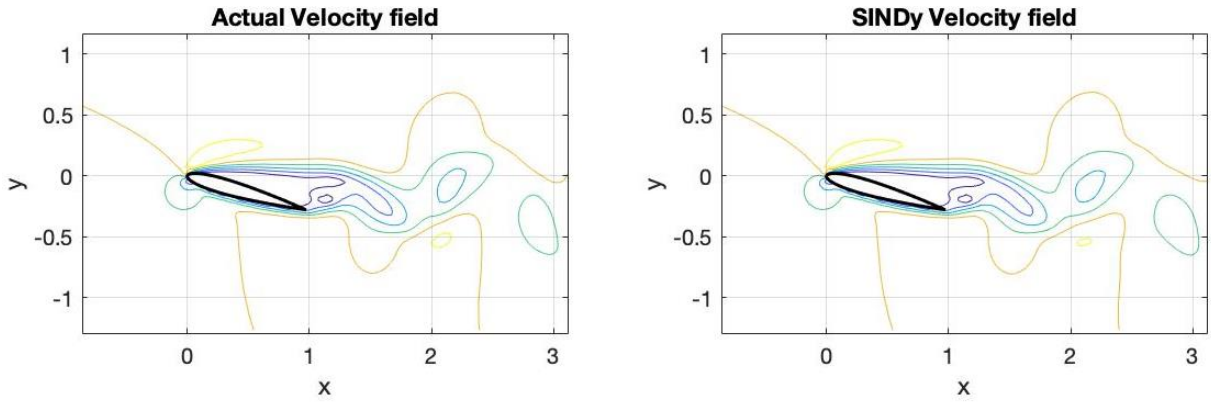
Figure 13 shows the corresponding location of *P1*, *P2* and *P3*. The comparison of u and v velocities at these points are depicted in the plots of following *Figure 14*.



a) u velocity b) v velocity

Figure 14: Comparison of velocity components

As expected, the velocities from SINDy also match very well with actual velocities within the learning dataset range. The results at the point situated in the wake seems to have less accuracy, as there are more irregularities in wake region, but again the accuracy level is acceptable. To give a more intuitive way of how well the identified system matches to the learning data, the entire velocity field reconstructed from SINDy is visualised, shown in following *Figure 15*.



a) Actual velocity field at $k = 3700$ b) SINDy velocity field at $k = 3700$

Figure 15: Comparison between velocity fields from learning dataset and from SINDy

The reconstructed velocity field from SINDy highly recovers the original velocity field, the velocity field are almost identical. Although some regions in SINDy velocity field is slightly distorted, the distortion is negligible. However, note the comparisons made here is between the learning data from which the SINDy algorithm is trying to fit and the results produced by the resulting fitting model. In other words, this comparison made is only able to tell that the model found fits very well to the learning dataset but does not evaluate the quality and the correctness of the model in terms of its dynamic. In order to evaluate the quality of the identified model in a deeper level, its ability of predicting future dynamic should be examined. To do this, a comparison between the testing data set and the model is made to validate the identified model.

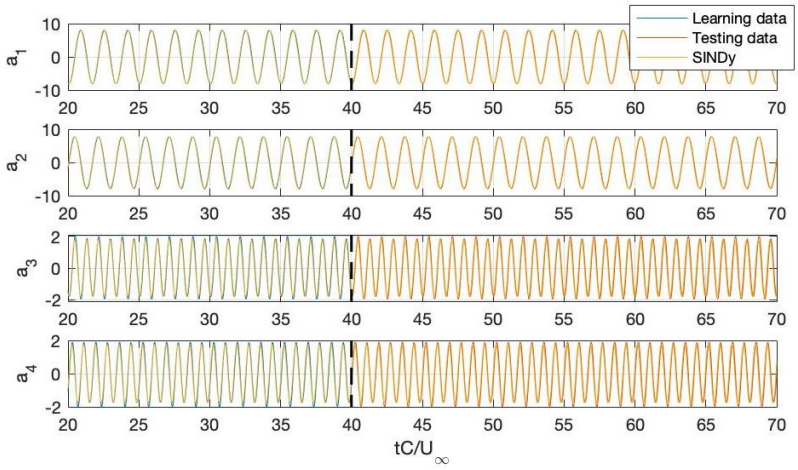
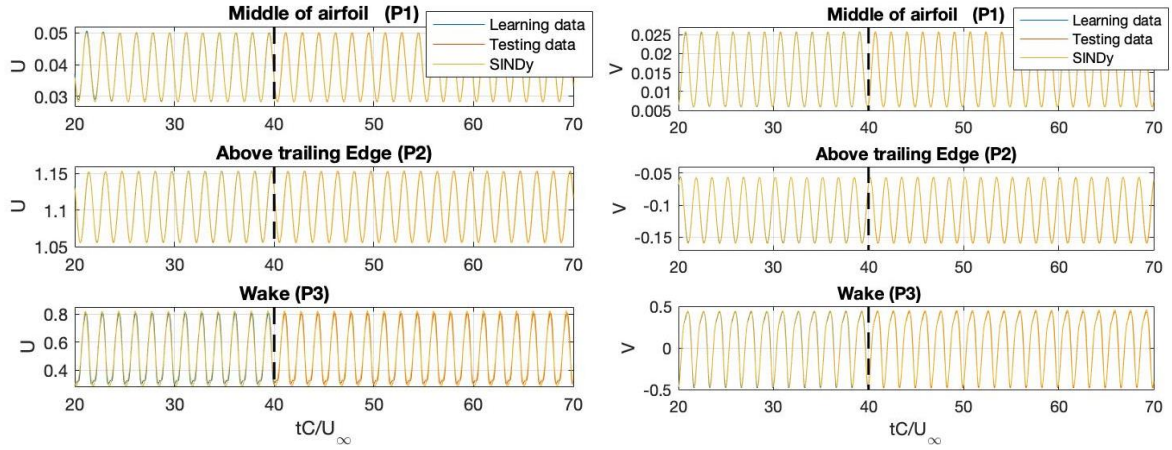


Figure 16: Comparison between coefficients from testing dataset and from SINDy

Figure 16 shows the results of a_m obtained from simulation and SINDy. The range of 20 to 40 physical time, namely, the left-hand side of the dashed line is the range where SINDy algorithm is applied to extract the governing dynamic equations; and the right-hand side of the dashed line are results predicted

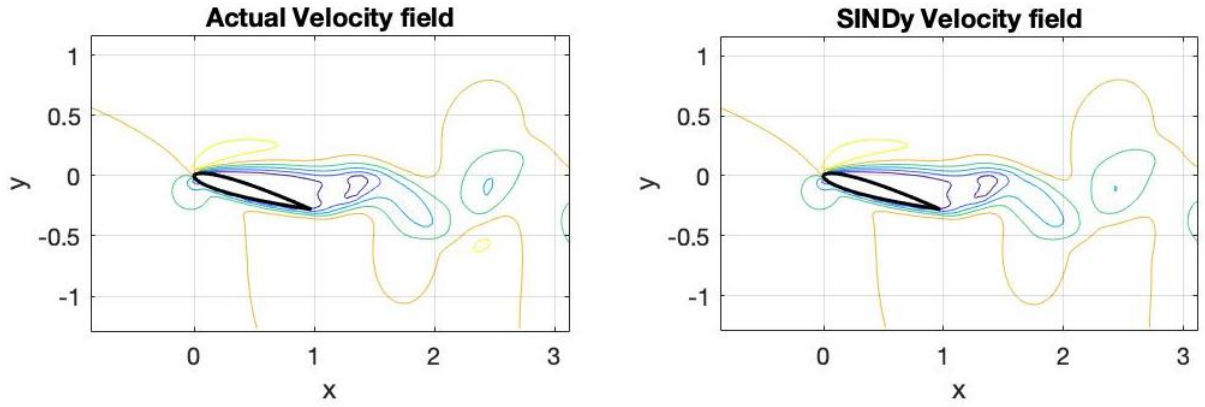
by the identified governing system of equations 37. As can be seen, the results predicted by theses equations also matches very well with simulation results. The results of u and v velocity at three points indicated in *Figure 13* are also compared for testing dataset.



a) u velocity b) v velocity

Figure 17: Comparison of velocity components

Observing the right-hand side part of the dashed line, the results for velocity components also match very well in testing dataset range. As a result, the model identified by the SINDy algorithm is validated. To see how well the model captures the information of velocity field, the coefficients are projected onto the POD modes and has produced the following velocity field shown in *Figure 18b.*



a) Actual velocity field at $k = 5400$ b) SINDy velocity field at $k = 5400$

Figure 18: Comparison between velocity fields from testing dataset and from SINDy

As can be seen from *Figure 18*, the velocity field predicted by the coefficient equations identified by SINDy is able to capture almost all the information of actual velocity field, the SINDy velocity field has a very high similarity to the actual velocity field. And note SINDy reconstructed this velocity without data from simulation, it is fully based on data from learning dataset. Although it is also true that

there are some observable differences in the lower part of the wake region, the accuracy of the SINDy identified system is still at an impressively high level.

4.2.3 Accuracy study

It is also worthy to see how well exactly do the estimated data match to actual simulation data, to quantify the similarity between these two results, the following definition is used:

$$SIMILARITY = 1 - \frac{\left\| f_{u_n} - \tilde{f}_{u_n} \right\|}{\left\| f_{u_n} - \bar{f}_{u_n} \right\|} \tag{38}$$

where f denotes the data from simulation, $\tilde{f}$ is its results from SINDy and $\bar{f}$ is the mean of f. This measurement of similarity is applied on a_m, U and V for learning data and testing data separately, the similarities in percentage are given in following *Table 5*.

Table 5: Similarities between actual and SINDy results

Learning dataset		**Testing dataset**	
Variable	**Similarity in %**	**Variable**	**Similarity in %**
a_m	89.32	a_m	87.33
U	97.72	U	97.28
V	88.66	V	87.25
Average	91.90	**Average**	90.62

From *Table 5*, it can be seen that the accuracy of the results from the SINDy algorithm is at a surprisingly high level. The accuracy level of the results between learning dataset range and testing dataset range does not differ a lot, the differences are within one decimal places, with the testing dataset portion having a slightly lower level. The averaged accuracy among these three variables for the learning data and testing data is 91.90% and 90.62% respectively, which is really high, this high level of accuracy implies that the system identified has successfully captured the underlying dynamic information behind the learning dataset.

If comparing the accuracy between these three variables, it is found that the accuracy of u velocity is about 9% higher than the that of v velocity. This can be explained by the fact that the u velocity is the dominant component in the simulation, which means that it contains more information of coherent structures and dynamics, this could result in higher retention of information after being performed a series of algorithm. Very interestingly, the accuracy of u velocity is even higher than that of POD coefficients, which is quite unexpected. Since theoretically, the results of u velocity are obtained by multiplying the coefficient matrix with the POD mode matrix, and this extra step of computation should

cause a further potential loss of information. Although the underlying reason of this phenomenon remains clear, it is always positive to see more accurate results.

4.2.4 Frequency study

In last section, the results are validated by comparing values in time domain. To explore the correctness of the model further, the frequency domain of the results is investigated in following. By considering both the results in time domain and the form of the identified equations 37, which are first order linear Ordinary Differential Equations (ODE), it is expected to have the solution in following form:

$$a_i = \hat{a}_i e^{i\omega t} \tag{39}$$

where $\hat{a}_i$ is a constant and ω is the frequency in radians. Substitute the expression 39 into the identified equations set 37 and rearrange gives:

$$\begin{aligned} i\omega\hat{a}_1 e^{i\omega t} - 3.93 \cdot \hat{a}_2 e^{i\omega t} &= 0 \\ i\omega\hat{a}_2 e^{i\omega t} + 3.63 \cdot \hat{a}_1 e^{i\omega t} &= 0 \\ i\omega\hat{a}_3 e^{i\omega t} - 7.66 \cdot \hat{a}_4 e^{i\omega t} &= 0 \\ i\omega\hat{a}_4 e^{i\omega t} + 7.45 \cdot \hat{a}_3 e^{i\omega t} &= 0 \end{aligned} \tag{40}$$

The term $e^{i\omega t}$ on the left-hand side can be eliminated as the right-hand side is 0, and this leads to two systems of two equations. Now if writing the systems into matrix form, it is found that they actually give rise to an eigenvalue problem:

$$\begin{bmatrix} i\omega & -3.93 \\ 3.63 & i\omega \end{bmatrix} \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \end{bmatrix} = 0 \tag{41}$$

$$\begin{bmatrix} i\omega & -7.66 \\ 7.45 & i\omega \end{bmatrix} \begin{bmatrix} \hat{a}_3 \\ \hat{a}_4 \end{bmatrix} = 0 \tag{42}$$

The frequency can be calculated by equalling the determinant of the 2×2 matrix to zero:

$$\begin{vmatrix} i\omega & -3.93 \\ 3.63 & i\omega \end{vmatrix} = 0 \rightarrow \omega = 3.77\ radians \tag{43}$$

$$\begin{vmatrix} i\omega & -7.66 \\ 7.45 & i\omega \end{vmatrix} = 0 \rightarrow \omega = 7.55\ radians \tag{44}$$

In order to obtain frequency domain information of the POD coefficients obtained from simulation, a fast Fourier transformation is performed to those data matrix. The frequency contents of the first 4 POD coefficients are depicted in following *Figure 19*.

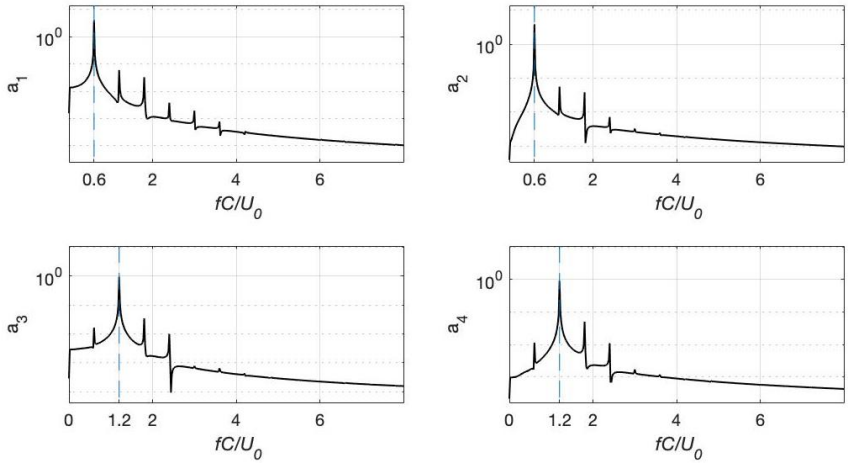


Figure 19: Frequency content of POD coefficients (simulation results)

The blue dashed line indicates the peak frequency location. As can be seen from the plots, the peak frequency of a_1 and a_2 is situated at 0.6; and for a_3 and a_4, the peak frequency is situated at 1.2, which is the double of the first pair mode. The results from fast Fourier transformation are in hertz, converting them into radians using the equation $\omega = 2\pi f$ gives 3.769 and 7.539 for the first and second pair mode respectively. If comparing these results with those calculated from identified equations, shown in equation 43 and 44, it is found that the results from identified equations are highly consistent with results from simulation. Thus, the frequency of the identified system is validated. Another important parameter to be checked in frequency domain is the phase difference between the modes.

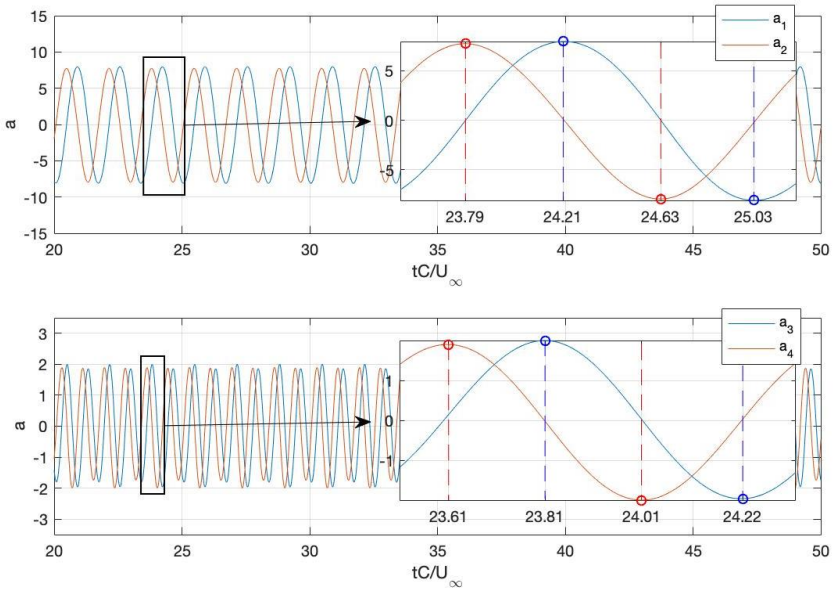


Figure 20: Phase difference of POD coefficient pairs (simulation results)

It can be clearly seen a phase difference for each mode pair from *Figure 20*. The phase difference can be roughly calculated by looking at the visualised results. To do that, firstly, semi-period is calculated

by looking at the difference between an adjacent maxima and minima. The values of maxima and minima are shown in *Figure 20*, taking those on blue line, which is a_1 and a_3, the semi-period for the first mode pair is calculated to 0.82 and for second mode pair it is 0.41. After that, the time difference between the maxima, or any point at the same magnitude, of different curve need to be found out. Again, from the plot, the difference for the first mode pair is 0.41 and for the second pair it is 0.21. Now, the phase difference can be calculated by multiplying 180 degrees with the ratio of minima difference to semi-period.

$$\phi_{12} = 180 \cdot \frac{0.42}{0.82} = 92.2° \tag{45}$$

$$\phi_{23} = 180 \cdot \frac{0.21}{0.41} = 92.2° \tag{46}$$

For the identified system of equations, the phase difference can be computed by looking at the eigenvectors of the eigenvalue problem defined in 41 and 42. The eigenvectors can be easily obtained by performing a row elimination on the system matrix, and they are found to be:

$$v_1 = \begin{bmatrix} 0.721 + 0i \\ 0 - 0.693i \end{bmatrix} \tag{47}$$

$$v_2 = \begin{bmatrix} 0.712 + 0i \\ 0 - 0.7022i \end{bmatrix} \tag{48}$$

The phase difference is the angle difference between the two complex numbers:

$$\phi_{12} = \tan^{-1}\left(\frac{0}{0.721}\right) - \tan^{-1}\left(\frac{-0.693}{0}\right) = 90° \tag{49}$$

$$\phi_{23} = \tan^{-1}\left(\frac{0}{0.712}\right) - \tan^{-1}\left(\frac{-0.7022}{0}\right) = 90° \tag{50}$$

As can be seen, the phase differences from the identified equations are found to be 90 degrees exactly for both mode pairs. These results of phase difference are almost identical to those of simulation results, implying that, apart from the frequencies information, the identified equations are also able to capture the phase difference information to a very accurate level.

4.3 Limitations

The SINDy has been proved to be a very powerful and useful algorithm to extract equations from the existing simulation or experimental data. However, it does have some limitations or restrictions. In most of cases, this algorithm is limited to 2D flows, as the highly complicated computations involved in algorithm requires a lot of computer resources. Implementation of SINDy algorithm on 3D problems

may be unfeasible for high-performance computers, it is extremely time consuming and expensive. Even for the 2D flow fields data, as the case of this project, the dimensionality of data has been firstly reduced using MOR technique and then use the ROM as the input of the algorithm. In other words, this algorithm is not suitable to deal with data of very large sizes.

For the unsteady flow case like in this project, where there is a Karman vortex street behind the airfoil, the algorithm is restricted to identify the dynamics of the field after reaching steady state. When performing the POD and SINDy algorithm, the transient period must be deducted from the input data in order to ensure a correct system identification. Incorporating the transient period leads to a non-sparse matrix $\boldsymbol{\Xi}$, which implies that most of the terms become active and the extracted system of equations become extremely complex with largely decreased accuracy. The results with transient data, using the corresponding optimal thresholding value $\lambda = 0.44$, are depicted in following *Figure 21*.

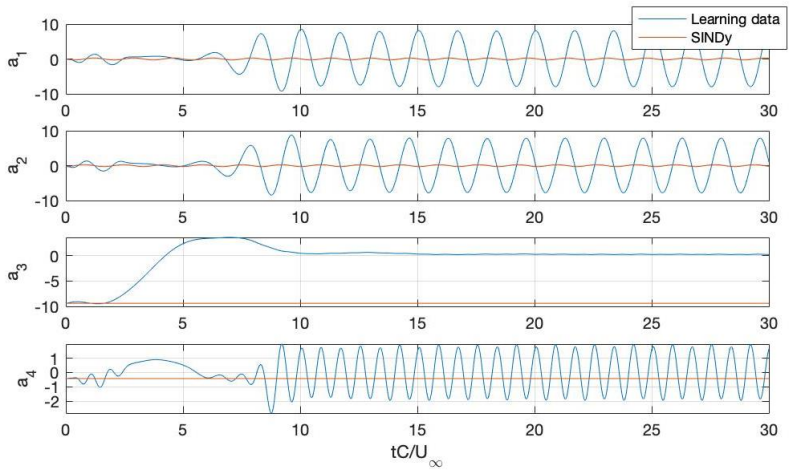


Figure 21: Comparison of simulation and SINDy results with transient data included

As can be seen, with transient data included, the system of equations identified by the SINDy fails to fit with the simulation results, the results do not match neither for transient period nor for the steady state. Hence, SINDy algorithm is unable to capture the dynamic when transient period is included, and the data of transient period need to be removed in order to ensure a correct identification.

Another limitation of SINDy is possibly the selection of the candidate function. Although there is a high flexibility of choosing which candidate function to be included in the library, it is an essential factor that affects the correctness and efficiency of the algorithm. A good function library will highly increase the efficiency of SINDy, which will avoid many unnecessary extra costs. However, this algorithm is designed to find underlying governing equations, which are not knonw, from the data, and thus is almost impossible to determine a concise library of high quality, which is a quite contradicting situation.

Chapter 5

5. Conclusion

Overall, the project has been carried out successfully and the results generated by the SINDy is very satisfactory. At the very first, a simulation has been made using the commercial CFD software Star-CCM+ to produce data of an unsteady flow case. The correctness of the simulation results was verified by comparing the frequency content to an existing paper. Then, a POD has been performed on the data to extract the time-dependent POD coefficient function, which is used as the input of the SINDy algorithm. The energy fraction and the pattern of each modes were investigated, it was found that the first 4 modes contains roughly 98.8% of energy and the 10 modes roughly 99.997%. The results of POD were validated by comparing the ROM reconstructed via the first few POD modes to the original full order model.

Once POD coefficients are extracted and verified to be correct, the coefficient matrix is divided into two portions. The first portion contains the coefficient data in time interval of 20 to 40, and this portion of data is used as the learning data for algorithm to identify and extract the equations; the rest of the data is used as the testing data, which aims to verify the results predicted by the extracted equations. Several thresholding value λ has been plugged in to find the optimal one, which is found to be $\lambda = 0.34$. Setting the λ to such value leads to a system of 4 equations with only 4 terms in total, which is really simple. It is found from the identified equations that the dynamics of the coefficients tends to be coupled with pairs, each pair tends to be independent to other pairs. This relationship between coefficients could be another interesting research topic.

The coefficient results from SINDy have been projected onto the POD modes to compute u and v velocity, and the velocity field has been reconstructed. The accuracy of the SINDy velocity field was found to be at a surprisingly high level, the average accuracy of the three measured variables was 91.90% for leaning data set and 90.62% for testing data set. Frequency study of the identified equations also showed that their information in frequency domain are highly consistent with those of simulation results. Thus, it can be concluded that the aim of extracting equation from data is totally achievable and additionally, the identified equations are able to capture the information in both time and frequency domain at an impressively high level of accuracy.

Bibliography

[1] Jordan M. and Mitchell T. (2015), *Machine learning: Trends, perspectives, and prospects.* Science 349(6245):255–260.

[2] Marx V. (2013), *Biology: The big challenges of big data.* Nature 498(7453):255–260

[3] Rudy S., Brunton S., Proctor J. and Kutz J. (2017), *Data-driven discovery of partial differential equations.* Sci. Adv. 3, e1602614.

[4] Schaeffer H., Caflisch R., Hauck C. and Osher S. (2013), *Sparse dynamics for partial differential equations.* Proc. Natl. Acad. Sci., 110, pp. 6634-6639

[5] Raissi M. and Karniadakis G. (2017), *Hidden physics models: Machine learning of nonlinear partial differential equations.* arXiv preprint arXiv:1708.00588.

[6] Raissi M., Yazdani A. and Karniadakis G. (2018), *Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data*. arXiv preprint arXiv: 1808.04327.

[7] Bongard J. and Lipson H. (2007), *Automated reverse engineering of nonlinear dynamical systems.* Proc Natl Acad Sci USA 104(24):9943–9948.

[8] Schmidt M. and Lipson H. (2009), *Distilling free-form natural laws from experimental data.* Science 324(5923):81–85.

[9] Koza J. (1992), *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA.

[10] Brunton S., Proctor J. and Kutz J. (2016), *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*. Proc. Natl Acad. Sci. USA 113 (15), 3932–3937.

[11] Loiseau J. and Brunton S. (2018), *Constrained sparse Galerkin regression.* J. Fluid Mech., vol. 838, pp. 42–67.

[12] Wang WX., Yang R., Lai YC., Kovanis V. and Grebogi C. (2011), *Predicting catastrophes in nonlinear dynamical systems by compressive sensing.* Phys Rev Lett 106(15):154101.

[13] Beale E., Kendall M. and Mann (1967), *The discarding of variables in multivariate analysis.* Biometrika 54(3/4), 357–366.

[14] Hocking R. and Leslie R. (1967), *Selection of the best subset in regression analysis.* Technometrics 9(4), 531–540.

[15] Efroymson M. (1966), *Stepwise regression—a backward and forward look.* Eastern Regional Meetings of the Institute of Mathematical Statistics.

[16] Draper N. and Smith H. (1966), *Applied Regression Analysis.* Wiley

[17] Tibshirani R. (1996), *Regression shrinkage and selection via the lasso.* J R Stat Soc, B 58(1):267–288.

[18] Chen S., Donoho D. and Saunders M. (1998), *Atomic decomposition for basis pursuit.* SIAM Journal on Scientific Computing 20(1), 33–61.

[19] Hastie T., Tibshirani R. and Friedman J. (2009), *The Elements of Statistical Learning.* Springer, New York, Vol 2.

[20] James G., Witten D., Hastie T. and Tibshirani R. (2013), *An Introduction to Statistical Learning.* Springer, New York.

[21] Bovet D. and Crescenzi P. (2006), *Introduction to the Theory of Complexity*. Prentice Hall. p. 69. ISBN 0-13-915380-2.

[22] Bertsimas D., King A. and Mazumder R. (2016), *Best subset selection via a modern optimization lens.* The Annals of Statistics 44(2), 813–852.

[23] Zhang W. and Samtaney R. (2016), *Biglobal linear stability analysis on low-re flow past an airfoil at high angle of attack*, Physics of Fluids 28(4), 044105. doi: 10.1063/1.4945005; 30.

URL: https://doi.org/10.1063/1.4945005

[24] Lumley J. (1967), *The structure of inhomogeneous turbulence.* In: Tararski Ya (ed) Atmospheric turbulence and wave propagation. Moscow: Nauka.

[25] Feeny B. and Liang Y. (2003), *Interpreting proper orthogonal modes of randomly excited vibration systems*, Journal of Sound and Vibration 265(5), 953–966.

[26] Kramer B. and Willcox K. (2018), *Nonlinear model order reduction via lifting transformations and proper orthogonal decomposition*, arXiv preprint arXiv:1808.02086.

[27] Sirovich L. (1987), *Turbulence and the dynamics of coherent structures 1. Coherent structures.* Quart Appl Math; 45: 561–571.

[28] Chen H., Reuss D., Hung D. and Sick, V. (2013), *A practical guide for using proper orthogonal decomposition in engine research*, International Journal of Engine Research 14(4), 307–319.

Appendix: MATLAB code

A.1 POD extraction

```
% A1 Proper Orthogonal Decomposition
% Wen Ze Liu Chen 07/09/19
% Code developed based on the code from Chen et al. (2012)

%% A1.1 Read data
clear all; close all

%import data from simulation data files
BaseName='vel_';
% discard transient period, about 20 physical time
tp_len = 2000;
for i = 2000:7000
    FileName=[BaseName,num2str(i),'.csv'];
    dat = csvread(FileName, 1, 0);
    U(:,i-1999) = dat(:,2);
    V(:,i-1999) = dat(:,3);
end
X = dat(:,5);
Y = dat(:,6);

%% A1.2 POD modes extraction

% mean velocity
U_mean = mean(U,2);
V_mean = mean(V,2);

% deduct mean velocity from data
U_dyn = U - U_mean;
V_dyn = V - V_mean;

% compute spatial correlation matrices
Cu = U_dyn'*U_dyn;
Cv = V_dyn'*V_dyn;
C = (Cu + Cv)/length(U);

% sigular value decomposition
% evec = eigenvector; eval = eigenvalue
[evec, eval] = svd(C, 'econ');

% compute POD modes
phix = U_dyn * evec;
phiy = V_dyn * evec;

% normalise the basis function
grid_num = size (X, 1) ;
for j = 1:size(U,2)
    phi_mag=0;

    for i = 1:grid_num
        phi_mag = phi_mag + phix(i,j)^2 + phiy(i,j)^2;
    end

    phi_mag = sqrt(phi_mag);

    phix (:,j) = phix (:,j)/phi_mag;
    phiy (:,j) = phiy (:,j)/phi_mag;
end
```

```
%compute POD coefficients
a_u = U_dyn' * phix;
a_v = V_dyn' * phiy;
a = a_u + a_v;

% initialisation of matrices
U_rom = zeros(size(U,1),size(U,2));
V_rom = zeros(size(V,1),size(V,2));
% Reconstruct the ROM of velocity field
for j =1:size(U,2)
   U_pod = U_mean;
   V_pod = V_mean;

   for i = 1:4 %number of pod modes used
      U_pod = U_pod + a(j,i) * phix(:,i);
      V_pod = V_pod + a(j,i) * phiy(:,i);
   end
   U_rom(:,j) = U_pod;
   V_rom(:,j) = V_pod;
end

%% A1.3 kinetic energy

KE_m = 0.5*sum(a.^2);
ke_m = KE_m./sum(KE_m);

%plot energy fraction
plot(ke_m(1:50),'x','LineWidth',2);
grid on;
xlabel('{\it m}th Mode');
ylabel('Kinetic Energy')

%% A1.4 airfoil

%rotation matrix, 16 degrees
TT = [cos(0.279253) -sin(0.279253); sin(0.279253) cos(0.279253)];
airfoil = csvread('0012.csv', 0, 0);
airfoil = airfoil*TT;

%% A1.5 results visualisation

%actual velocity field
mag = sqrt(U.^2 + V.^2);
snp_sht = 3300;
figure('Name', 'Velocity Magnitude');
[xx,yy] = meshgrid( linspace(min(X),max(X)),linspace(min(Y),max(Y)) );
zz = griddata(X,Y,mag(:,snp_sht), xx, yy);
contour(xx,yy,zz);
hold on;
grid on;
xlabel ('x'); ylabel('y');
title('Velocity Magnitude');
axis equal;
fill( airfoil(:,1), airfoil(:,2),'w','LineWidth',1.5);

% pod velocity field
mag_pod = sqrt(U_rom.^2 + V_rom.^2);
figure('Name', 'POD Velocity Magnitude');
zz = griddata(X,Y,mag_pod(:,snp_sht), xx, yy);
contour(xx,yy,zz);
hold on;
grid on;
xlabel ('x'); ylabel('y');
title('POD Velocity Magnitude');
axis equal;
fill( airfoil(:,1), airfoil(:,2),'w','LineWidth',1.5);
```

```
% pod modes 1-4
figure;
for i = 1:4
    subplot(2,2,i);
    zz = griddata(X,Y,phix(:,i),xx,yy);
    contour(xx,yy,zz);
    hold on;
    axis equal;
    grid on;
    fill( airfoil(:,1), airfoil(:,2),'w','LineWidth',1.5);
    xlabel('X');
    ylabel('Y');
    title(['\phi_',num2str(i)]);
end

% pod mdoes 5-8
figure;
for i = 5:8
    subplot(2,2,i-4);
    zz = griddata(X,Y,phix(:,i),xx,yy);
    contour(xx,yy,zz);
    hold on;
    axis equal;
    grid on;
    fill( airfoil(:,1), airfoil(:,2),'w','LineWidth',1.5);
    xlabel('X');
    ylabel('Y');
    title(['\phi_',num2str(i)]);
end

% pod 9-10
figure;
for i = 9:10
    subplot(1,2,i-8);
    zz = griddata(X,Y,phix(:,i),xx,yy);
    contour(xx,yy,zz);
    hold on;
    axis equal;
    grid on;
    fill( airfoil(:,1), airfoil(:,2),'w','LineWidth',1.5);
    xlabel('X');
    ylabel('Y');
    title(['\phi_{',num2str(i),'}']);
end

% mean velocity field
mag_mean = mean(mag,2);
figure('Name', 'Mean velocity');
zz = griddata(X,Y,mag_mean, xx, yy);
contour(xx,yy,zz);
hold on;
grid on;
xlabel ('x'); ylabel('y');
title('Mean Velocity');
axis equal;
fill( airfoil(:,1), airfoil(:,2),'w','LineWidth',1.5);

%% A1.6 save variables for SINDy algorithm

save('SINDy_data.mat','a','airfoil','mag','mag_pod',...
'phix','phiy','U','U_rom','V','V_rom','X','Y')
```

A.2 SINDy algorithm

```
% A2 Sparse Identification of Nonlinear Dynamics
% Wen Ze Liu Chen 07/09/19
% Code developed based on the code from Brunton et al. (2016)

%% A2.1 Read data produced from A1 POD code
clear all; close all;

%load variables
load('SINDy_data.mat');

%% A2.2 Derivatives of POD coefficients

%length of learning data (LD), testing data (TD) and entire data
len_LD = 2001;
len_TD = 3000;
len_ALL = 5001;

%coefficients in LD range
a_LD = a(1:len_LD,:);

%coefficients in TD range
a_TD = a(len_LD+1:len_ALL,:);

%define constants
M = 4; %mode number
dt = 0.01; %time step
n_t = size(a_LD, 1); %number of time steps
n_p = size(phix, 1); %number of grid points

%derivatives (4th order central scheme)
dadt = (1/(12*dt))*(-a_LD(5:n_t, 1:M)+8*a_LD(4:n_t-1, 1:M)-8*a_LD(2:n_t-3, 1:M)+a_LD(1:n_t-4, 1:M));

%resize the a matrices to match with dadt matrices
a_rs = a_LD(3:end-2, 1:M);

%% A2.3 SINDy implementation

% define parameters
polyorder = 2; %order of polynomials
usesine = 0; %don't include trigonometry functions

%create the candidate function library
Theta = poolData(a_rs,M,polyorder,usesine);
m = size(Theta,2);

% thresholding value
lambda = 0.34
%identify the sparse coefficient matrix
Xi = sparsifyDynamics(Theta,dadt,lambda,M)

% print active terms of sparse matrix
poolDataLIST({'a1','a2','a3','a4'},Xi,M,polyorder,usesine);

% integrate the identified system of equations
tspan = [0:.01:50];
options = odeset('RelTol',1e-8,'AbsTol',1e-8*ones(1,M));
a_0 = a_rs(1,:);
[tD,a_sindy]=ode45(@(t,x)sparseGalerkin(t,x,Xi,polyorder,usesine),tspan,a_0,options);

%% A2.4 Construct veloctiy field using SINDy results

% compute mean velocity field
U_mean = mean(U,2);
```

```
V_mean = mean(V,2);

for j =1:length(tspan)
    U_pod_sindy = U_mean;
    V_pod_sindy = V_mean;

    for i = 1:4 %number of pod modes used
        U_pod_sindy = U_pod_sindy + a_sindy(j,i) * phix(:,i);
        V_pod_sindy = V_pod_sindy + a_sindy(j,i) * phiy(:,i);
    end
    U_sindy(:,j) = U_pod_sindy;
    V_sindy(:,j) = V_pod_sindy;
end

%% A2.5 Results visualisation/comparison

% POD coefficients, learning data
figure('Name','a SINDy');
for i = 1:4
    subplot(4,1,i);
    plot((20:0.01:40),a_LD(:,i));
    hold on;
    plot((20.02:0.01:40),a_sindy(1:1999,i));
    ylabel(['a_{',num2str(i),'}']);
    grid on;
    if i == 1
    legend('Learning data','SINDy');
    end
end
xlabel('tC/U_\infty')

% velocity components, learning data
figure('Name','U SINDy');
p = [2506, 2863, 2072];
loc = ['Middle of airfoil   (P1)';
       'Above trailing Edge (P2)';
       'Wake (P3)               '];
for i = 1:3
    subplot(3,1,i);
    plot((20:0.01:40),U(p(i),1:2001));
    hold on;
    plot((20.02:0.01:40),U_sindy(p(i),1:1999));
    ylabel(['U']);
    grid on;
    title(loc(i,:));
    if i == 1
    legend('Learning data','SINDy');
    end
end
xlabel('tC/U_\infty')

figure('Name','V SINDy');
for i = 1:3
    subplot(3,1,i);
    plot((20:0.01:40),V(p(i),1:2001));
    hold on;
    plot((20.02:0.01:40),V_sindy(p(i),1:1999));
    ylabel(['V']);
    grid on;
    title(loc(i,:));
    if i == 1
    legend('Learning data','SINDy');
    end
    axis tight;
end
xlabel('tC/U_\infty')
```

```
% Actual velcity field
mag = sqrt(U.^2 + V.^2);
mag_sindy = sqrt(U_sindy.^2 + V_sindy.^2);
snp_sht = 3400;
figure('Name', 'Velocity Magnitude');
[xx,yy] = meshgrid( linspace(min(X),max(X)),linspace(min(Y),max(Y)) );
subplot(1,2,1);
zz = griddata(X,Y,mag(:,snp_sht+2), xx, yy);
contour(xx,yy,zz);
hold on;
grid on;
xlabel ('x'); ylabel('y');
title('Actual Velocity field');
axis equal;
fill( airfoil(:,1), airfoil(:,2),'w','LineWidth',1.5);

% SINDy velocity field
subplot(1,2,2);
zz = griddata(X,Y,mag_sindy(:,snp_sht), xx, yy);
contour(xx,yy,zz);
hold on;
grid on;
xlabel ('x'); ylabel('y');
title('SINDy Velocity field');
axis equal;
fill( airfoil(:,1), airfoil(:,2),'w','LineWidth',1.5);


% POD coefficients, testing data
figure('Name','a SINDy');
dashlu = [40 40 10 -10;
        40 40 10 -10;
        40 40 2.1 -2.1;
        40 40 2 -2];
for i = 1:4
    subplot(4,1,i);
    plot((20:0.01:40),a_LD(:,i));
    hold on;
    plot((40.01:0.01:70),a_TD(:,i));
    plot((20:0.01:70),a_sindy(:,i));
    ylabel(['a_{',num2str(i),'}']);
    grid on;
    plot(dashlu(i,1:2),dashlu(i,3:4),'--k','LineWidth',1.5);
    if i == 1
    legend('Learning data','Testing data','SINDy');
    end
end
xlabel('tC/U_\infty')

% velocity components, testing data
figure('Name','U SINDy');
dashlu = [40 40 0.052 0.027;
        40 40 1.16 1.05;
        40 40 0.85 0.3];
for i = 1:3
    subplot(3,1,i);
    plot((20:0.01:40),U(p(i),1:2001));
    hold on;
    plot((40.01:0.01:70),U(p(i),2002:5001));
    plot((20.02:0.01:70),U_sindy(p(i),1:4999));
    plot(dashlu(i,1:2),dashlu(i,3:4),'--k','LineWidth',1.5);
    ylabel(['U']);
    grid on;
    title(loc(i,:));
    if i == 1
    legend('Learning data','Testing data','SINDy');
    end
```

```
    axis tight;
end
xlabel('tC/U_\infty')

figure('Name','V SINDy');
dashlu = [40 40 0.027 0.005;
        40 40 -0.04 -0.17;
        40 40 0.5 -0.5];
for i = 1:3
    subplot(3,1,i);
    plot((20:0.01:40),V(p(i),1:2001));
    hold on;
    plot((40.01:0.01:70),V(p(i),2002:5001));
    plot((20.02:0.01:70),V_sindy(p(i),1:4999));
    plot(dashlu(i,1:2),dashlu(i,3:4),'--k','LineWidth',1.5);
    ylabel(['V']);
    grid on;
    title(loc(i,:));
    if i == 1
    legend('Learning data','Testing data','SINDy');
    end
    axis tight;
end
xlabel('tC/U_\infty')

%% A2.6 Accuracy

% accuracy for learning dataset
acy_a_LD = 1-norm(a(1:2001,1:M) - a_sindy(1:2001,:))/norm(a(1:2001,1:M) - mean(a(1:2001,1:M),'all'));
acy_U_LD = 1-norm(U(:,1:2001) - U_sindy(:,1:2001))/norm(U(:,1:2001) - mean(U(:,1:2001),'all'));
acy_V_LD = 1-norm(V(:,1:2001) - V_sindy(:,1:2001))/norm(V(:,1:2001) - mean(V(:,1:2001),'all'));

% accuracy for testing
acy_a_TD = 1-norm(a(2001:5001,1:M) - a_sindy(2001:5001,:))/norm(a(2001:5001,1:M) - mean(a(2001:5001,1:M),'all'));
acy_U_TD = 1-norm(U(:,2001:5001) - U_sindy(:,2001:5001))/norm(U(:,2001:5001) - mean(U(:,2001:5001),'all'));
acy_V_TD = 1-norm(V(:,2001:5001) - V_sindy(:,2001:5001))/norm(V(:,2001:5001) - mean(V(:,2001:5001),'all'));
```

A.3 Functions involved in SINDy (Brunton et al. [10])

```
%% Sequential thresholding
function Xi = sparsifyDynamics(Theta,dXdt,lambda,n)
% Copyright 2015, All Rights Reserved
% Code by Steven L. Brunton
% For Paper, "Discovering Governing Equations from Data:
%        Sparse Identification of Nonlinear Dynamical Systems"
% by S. L. Brunton, J. L. Proctor, and J. N. Kutz

% compute Sparse regression: sequential least squares
Xi = Theta\dXdt;  % initial guess: Least-squares

% lambda is our sparsification knob.
for k=1:10
    smallinds = (abs(Xi)<lambda);   % find small coefficients
    Xi(smallinds)=0;                % and threshold
    for ind = 1:n                % n is state dimension
        biginds = ~smallinds(:,ind);
        % Regress dynamics onto remaining terms to find sparse Xi
        Xi(biginds,ind) = Theta(:,biginds)\dXdt(:,ind);
    end
end


%% Integration by sparse Galerkin
function dy = sparseGalerkin(t,y,ahat,polyorder,usesine)
% Copyright 2015, All Rights Reserved
% Code by Steven L. Brunton
% For Paper, "Discovering Governing Equations from Data:
%        Sparse Identification of Nonlinear Dynamical Systems"
% by S. L. Brunton, J. L. Proctor, and J. N. Kutz

yPool = poolData(y',length(y),polyorder,usesine);
dy = (yPool*ahat)';


%% Print candidate function
function yout = poolDataLIST(yin,ahat,nVars,polyorder,usesine)
% Copyright 2015, All Rights Reserved
% Code by Steven L. Brunton
% For Paper, "Discovering Governing Equations from Data:
%        Sparse Identification of Nonlinear Dynamical Systems"
% by S. L. Brunton, J. L. Proctor, and J. N. Kutz

n = size(yin,1);

ind = 1;
% poly order 0
yout{ind,1} = ['1'];
ind = ind+1;

% poly order 1
for i=1:nVars
    yout(ind,1) = yin(i);
    ind = ind+1;
end

if(polyorder>=2)
    % poly order 2
    for i=1:nVars
        for j=i:nVars
            yout{ind,1} = [yin{i},yin{j}];
            ind = ind+1;
        end
```

```
    end
end

if(polyorder>=3)
    % poly order 3
    for i=1:nVars
        for j=i:nVars
            for k=j:nVars
                yout{ind,1} = [yin{i},yin{j},yin{k}];
                ind = ind+1;
            end
        end
    end
end

if(polyorder>=4)
    % poly order 4
    for i=1:nVars
        for j=i:nVars
            for k=j:nVars
                for l=k:nVars
                    yout{ind,1} = [yin{i},yin{j},yin{k},yin{l}];
                    ind = ind+1;
                end
            end
        end
    end
end

if(polyorder>=5)
    % poly order 5
    for i=1:nVars
        for j=i:nVars
            for k=j:nVars
                for l=k:nVars
                    for m=l:nVars
                        yout{ind,1} = [yin{i},yin{j},yin{k},yin{l},yin{m}];
                        ind = ind+1;
                    end
                end
            end
        end
    end
end

if(usesine)
    for k=1:10;
        yout{ind,1} = ['sin(',num2str(k),'*yin)'];
        ind = ind + 1;
        yout{ind,1} = ['cos(',num2str(k),'*yin)'];
        ind = ind + 1;
    end
end


output = yout;
newout(1) = {''};
for k=1:length(yin)
    newout{1,1+k} = [yin{k},'dot'];
end
% newout = {'','xdot','ydot','udot'};
for k=1:size(ahat,1)
    newout(k+1,1) = output(k);
    for j=1:length(yin)
        newout{k+1,1+j} = ahat(k,j);
    end
end
```

```
newout


%% Candidate function library
function yout = poolData(yin,nVars,polyorder,usesine)
% Copyright 2015, All Rights Reserved
% Code by Steven L. Brunton
% For Paper, "Discovering Governing Equations from Data:
%        Sparse Identification of Nonlinear Dynamical Systems"
% by S. L. Brunton, J. L. Proctor, and J. N. Kutz

n = size(yin,1);
% yout = zeros(n,1+nVars+(nVars*(nVars+1)/2)+(nVars*(nVars+1)*(nVars+2)/(2*3))+11);

ind = 1;
% poly order 0
yout(:,ind) = ones(n,1);
ind = ind+1;

% poly order 1
for i=1:nVars
    yout(:,ind) = yin(:,i);
    ind = ind+1;
end

if(polyorder>=2)
    % poly order 2
    for i=1:nVars
        for j=i:nVars
            yout(:,ind) = yin(:,i).*yin(:,j);
            ind = ind+1;
        end
    end
end

if(polyorder>=3)
    % poly order 3
    for i=1:nVars
        for j=i:nVars
            for k=j:nVars
                yout(:,ind) = yin(:,i).*yin(:,j).*yin(:,k);
                ind = ind+1;
            end
        end
    end
end

if(polyorder>=4)
    % poly order 4
    for i=1:nVars
        for j=i:nVars
            for k=j:nVars
                for l=k:nVars
                    yout(:,ind) = yin(:,i).*yin(:,j).*yin(:,k).*yin(:,l);
                    ind = ind+1;
                end
            end
        end
    end
end

if(polyorder>=5)
    % poly order 5
    for i=1:nVars
        for j=i:nVars
            for k=j:nVars
                for l=k:nVars
```

```
                for m=l:nVars
                    yout(:,ind) = yin(:,i).*yin(:,j).*yin(:,k).*yin(:,l).*yin(:,m);
                    ind = ind+1;
                end
            end
        end
    end
end

if(usesine)
    for k=1:10;
        yout = [yout sin(k*yin) cos(k*yin)];
    end
end
```