Modelling and Reconstruction of Fluid Equations Using Machine Learning

A DISSERTATION

Submitted in partial fulfillment of the requirement for the award of the degree of

Masters of Technology

Bioinformatics

Department of Applied Science

Indian Institute of Information Technology - Allahabad

Jay Vyas

MBI2020016

Under the guidance of :

Dr. Srijit Bhattacharjee

Academic year 2021-2022

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD
(A centre of Excellence in Information Technology Established by Govt. of India)

CANDIDATE'S DECLARATION

I, **Jay Vyas**, enrollment number: **MBI2020016**, certify that this thesis work entitled **"Modelling and Reconstruction of Fluid Equations Using Machine Learning"** is submitted by me in partial fulfillment of the requirement of the Degree of **Master of Technology** to the **Department of Applied Sciences, Indian Institute of Information Technology, Allahabad**.
I understand that plagiarism includes:

1. Reproducing someone else's work (fully or partially) or ideas and claiming it as one's own.

2. Reproducing someone else's work (Verbatim copying or paraphrasing) without crediting.

3. Committing literary theft (copying some unique literary construct).

I have given due credit to the original authors/sources through proper citation for all the words, ideas, diagrams, graphics, computer programs, experiments, results, websites, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

I affirm that no portion of my work is plagiarized. In the event of a complaint of plagiarism, I shall be fully responsible. I understand that my Supervisor may not be in a position to verify that this work is not plagiarized.

Date: /05/2022
Place : IIIT Allahabad,Prayagraj

Jay Vyas
MBI2020016
Department of Applied Sciences
IIIT Allahabad

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD
(A centre of Excellence in Information Technology Established by Govt. of India)

CERTIFICATE FROM SUPERVISOR

This is to certify that the statement made by the candidate is correct to the best of my knowledge and belief. The master's thesis titled **"Modelling and Reconstruction of Fluid Equations Using Machine Learning"** is a record of the candidate's work carried out by him under my guidance and supervision. I do hereby recommend that it should be accepted for the fulfillment of the requirements for the degree of **Master of Technology** in **Bioinformatics**.

Date: /05/2022
Place : IIIT Allahabad,Prayagraj

Dr. Srijit Bhattacharjee
(Supervisor)
Department of Applied Sciences
IIIT Allahabad

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD

(A centre of Excellence in Information Technology Established by Govt. of India)

CERTIFICATE OF APPROVAL

The forgoing thesis is hereby approved as a credible study in the area of Applied Science and its allied areas carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it is submitted.

Name and Signature of the Committee Members on final examination for approval of the thesis.

Acknowledgment

The extensive endeavour, bliss euphoria that accompanies the successful completion of any project work would be impossible without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts. This thesis was not only an endeavour but also an interesting learning experience for me and it bears the imprint of a number of people who directly or indirectly were a great source of help and constant encouragement.

I would like to express my sincere thanks to my mentor Dr. Srijit Bhattacharjee, for his continuous motivation and guidance. His valuable suggestions, comments and support were an immense help for me. I am grateful to him for taking out time from his busy schedule and being very supportive in guiding my work. Also I would like to thank Dr. Tapobrata Lahiri. Lahiri sir taught us a course in Machine Learning where I developed intuitions about machine learning which motivated me for this thesis work. I would also like to extend my sincere thanks to Dr. Ratan Saha, HOD of the department, for helping us always in every possible manner.

Special thanks to my parents for always supporting me throughout my journey. I would also like to extend my gratitude to my friends Deep Joshi and Zarna Joshi and my peers at IIIT-A for their efforts and cooperation. The interesting and informative discussions we had together greatly contributed to the completion of this work.

Jay Vyas
MBI2020016

Abstract

Differential equations govern all time varying processes and systems. Their analysis reveal the behaviour and properties of the system. In this thesis, an attempt is made to model and reconstruct partial differential equations for fluid systems using machine learning techniques. Data for advection equation, inviscid and viscid Burgers' equation is generated using numerical schemes. Artificial neural network is used to model advection and inviscid Burgers' equation. Reconstruction of viscid Burgers' equation is done using LASSO regression, where clean and noisy data is used to unearth the underlying partial differential equation governing the system. This model can be further extended to data from other system from the domains of biology, chemistry, and physics to uncover the governing equation that control those system. These equations can then to used to predict the system behaviour for future states and for different conditions.

Contents

List of Figures

List of Tables

1 Introduction

Nature speaks the language of mathematics. All the process in nature are modelled by mathematical equations. The things that evolves with time are described using differential equations. These equations govern the behaviour of the things in question and also helps in predicting its future state. Thus study of differential equations is of extreme importance . These equations play important role in almost all the fields. They govern systems in the fields of biology, physics, chemistry, engineering, economics [22]. Since almost all systems in nature behave in accordance to some governing differential equations, the prediction of those equations from the behaviour of the system can help in better understanding of that system as well as predicting the future behaviour of the system.

Initial attempt to solve any differential equation is to come up with an analytical solution which gives the correct answer to the problem. But due to its inherent complexity, only a few types of differential equations are analytically solvable. Now the question arises as to how to tackle such problems. One way to do it is to numerically approximate the solution. In such a case, the accuracy of the solution depends upon the sophistication of the numerical method used and degree of error that can be tolerated [22].

The solution of such differential equations that governs natural systems would help us to determine its properties at any given time and at any point in space and thus helps us to determine as well as predict the behaviour of the system. With advent of enhanced power of computation and newer, superior algorithms, we can deploy certain machine learning models that could learn and extract patterns and information from the system data. This could be an aid to determine and predict the behaviour of systems that could otherwise be extremely difficult. In can be used to captures the essence of the system behaviour and also helps us in prediction. Thus it could be very useful if the underlying differential equation can be extracted from the data. Once this equation is extracted, it can then be used for all sorts of analysis of the system and also helps in predicting the system properties in different conditions.

These differential equations existing in nature have physical significance. Each term of the equation has a physical meaning. Thus there are very few terms in the equation which makes it interpretable and relatable to natural processes. So to unearth such equations from the data, the model has to predict an implicit function of derivatives, power terms of system variables and their products which the data follows. As there are many partial derivatives and many powers, thus the library of features used as input is very large compared to the number of features actually important for that system. Thus the function which the model generates must have zero coefficients for a majority of terms. Hence to solve this, sparse regression techniques are utilised [5].

In this thesis, fluids system and their governing equations are studied and prediction techniques are developed around it. Fluids have been one the most prominent and the most essential part of our universe. Study of fluids is considered as one the very important pursuits owing to its wide-scale presence. This is not particularly easy due to the complex nature of the fluids. The immense appearance of fluids in engineering applications makes it even more tempting to dive deep in to the field and price out solutions to difficult problems. Ever since the times of Archimedes, Da Vinci and Newton, fluids are studied with great mathematical rigor. The study of Fluids is very complicated and thus requires sophisticated techniques. This is the reason behind selecting this area so that advanced methods can be developed for its prediction which can then be used and applied in other domains.

The most basic property of fluids is the flow. This time and space varying nature of fluids

is expressed using differential equations that provides a relationship between various properties. Differential equations relates the properties such as position, velocity, force, temperature, pressure, density, viscosity, etc and their time and space derivatives, explaining the behaviour of fluids.

Here an attempt to approximate one dimensional advection equation and one dimensional inviscid Burgers' equation using various supervised learning algorithms is made. This helps in dealing with fluids using machine learning model. The main theme of this thesis revolves around developing/ predicting the governing differential equation for fluid systems from raw fluid data, that can then be used to predict the behaviour of the system. Fluid velocity data that mimics Burgers' equation is used to reconstruct the Burgers' equation thereby testing the model for use in other domains. The data is generated using numerical schemes. This model takes in raw data of the system and reconstructs the governing equation that controls the system. Also one dimensional advection and one dimensional inviscid Burgers' equation are used to train supervised learning algorithms that predicts the velocity of the fluid of a system that follows those equations.

Current development in this domain has been with simulated data as the one being used in this thesis. The data generated/ simulated has to be corrupted with external random noise such that it mimics the real time data. Techniques are developed which processes the noisy data to get cleaner and smoother data. Cleaner data is used to train the model as it follows the original underlying pattern and helps in uncovering it. The work with real time data will require advanced tools for measurement of properties and fine tuned techniques that can deal with external noise. This methodology could be extended to the domains of biology, chemistry and engineering in which case the study and evaluation of the system from the data is inherently tough due to the scale and inaccessibilty. Such a technique which could price out the underlying equations could be very useful and revolutionary. The problem of unavailability of real time data must also be solved so that the models can be further improved and made more general purpose [24, 25, 3, 7, 16, 30].

The outline of this thesis is as follows. Chapter 2 includes the literature survey that states all the prerequisite knowledge that is required to understand the work. Chapter 3 deals with the generation of data that is required to test and train the model using numerical schemes. The neural network model, its working, architecture for prediction of behaviour of advection and inviscid Burgers' equation and results are described in chapter 4. Chapter 5 contains the main theme of the thesis. The entire methodology and results of reconstruction of the governing Burgers' equation from clean and noisy data is described in that chapter. The Last chapter, chapter 6 contains the concluding remarks and what further advancements could be made in this domain.

2 Literature survey

2.1 Advection - Diffusion processes

The advection – diffusion processes is a combination of the diffusion and advection process. It describes the natural phenomenon where matter, energy, or other physical quantities are transferred within a system. Advection is the phenomenon of transport particles due to the bulk flow of the fluid. It depends on the velocity of the fluid. Diffusion on the other is also a transport phenomenon of particles but takes place in random motion and due to the difference in concentration/ temperature i.e due to the presence of a concentration gradient [2].

The fluid's motion is described mathematically as a vector field, and the transported material is described by a scalar field showing its distribution over space.

2.2 PDEs

A partial differential equation (PDE) is a differential equation which establishes a relationship between the various partial derivatives of a function. To get the solution of a PDE, we need initial conditions and boundary conditions. The boundary conditions are with respect to space whereas the initial conditions are with respect to time. [22]

A PDE with derivatives in only one spatial dimension and the temporal dimension describes how physical quantities change in say a horizontal direction and with time. Here we will discuss two such equations, the advection PDE and Burger's equation.

2.2.1 The advection PDE

It is of the form :

$$\frac{\partial u}{\partial t} + k\frac{\partial u}{\partial x} = 0[1]$$

Here $u = u(x,t)$ is the function of space and time representing the velocity field of the fluid. So $\frac{\partial u}{\partial t}$ represents the temporal rate of change of velocity and $\frac{\partial u}{\partial x}$ is the spatial rate of change of velocity. k is the speed of progression of the waveform, which in our case is taken as 1. The advection equation transmits the initial waveform, at the velocity k. The actual form of this waveform is maintained in this process.
It is a linear PDE which means that the dependent variable (velocity) and all its partial derivatives occur linearly. It is also a single degree PDE.

2.2.2 The Burgers' Equation

Burgers' equation is the PDE which exhibits the effects of both non linear advection and diffusion processes. It is the simplest such model and helps in analysing more complex models exhibiting similar behavior like the Navier Stokes equation. It physically signifies the transport of matter/ energy [6].
For a given velocity field $u(x,t)$ and diffusion coefficient or kinematic viscosity ν , the general form of Burgers' equation (also known as viscous Burgers' equation) in one spatial dimension:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial^2 x}[6]$$

.

When the diffusion term is absent i.e. $\nu = 0$, Burgers' equation becomes the inviscid Burgers' equation :

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0$$

. This equation without the viscous term ν results in generation of shock waves, breaking of waves at certain points resulting in discontinuities.
The presence of viscous term ν helps in subduing the breaking of waves which in turn irons out shock discontinuities. As a result a smooth and ordered solution.

2.2.3 Initial and boundary conditions

PDE is a function of a physical quantity or vector field and its space and time derivatives. The exact solution of the PDE is the definition of the vector field i.e. value of that field at all points. This requires specification of the initial and boundary conditions [22].

The initial condition specifies how the quantity or the vector field or any of its derivatives behaves at the starting time i.e. $t = 0$. So it can be defined as :

$$u(x, 0) = f(x)$$

where f is the function which demonstrates how the field will behave at $t = 0$.

Also a fixed solution to any PDE requires the fixed boundary inside which the quantity is observed, measured and evaluated. So the boundary conditions are the conditions which specifies the behaviour of the vector field at the spatial boundary inside which the problem is defined for example at $x = x_{start}$ and $x = x_{end}$.

$$u(x_{start}, t) = g(t)$$

$$u(x_{end}, t) = h(t)$$

here, g and h are time varying functions explaining the behaviour of fluid at the boundaries of spatial dimension x.

Periodic boundary conditions
Periodic boundary condition are a set of boundary conditions which are such that the value of the vector field at the entry/ starting point of the boundary is equal to the value of the vector field at the exit/ end point of the boundary for a given time stamp. In this case the periodic boundary condition is used such that the value of the velocity field at $x = x_{start}$ is the same as the value of the velocity field at $x = x_{end}$ for any time t, i.e.

$$u(x_{start}, t) = u(x_{end}, t)$$

2.3 Multiple Linear Regression

Linear regression is used to predict or forecast the value of a dependent variable based on the value of other independent variables. If this relationship between the dependent variable and the independent variables is established , it will help in making predictions about the dependent variable in terms of the independent variables. Regressions based on more than one independent variable are called multiple linear regressions.
Suppose that the dependent variable y depends on n independent features x_s. Then the equation which helps us estimate y in terms of x_s can be written as [28]:

$$y = \theta_0 + \theta_1 x_1 + \theta_1 x_2 + \ldots + \theta_n x_n + \epsilon$$

Here θ_0 is the bias term. ϵ is the error term which follows normal distribution.
The predicted value of the dependent variable y_{pred} can thus be expressed as:

$$y_{pred} = \theta_0 + \theta_1 x_1 + \theta_1 x_2 + \ldots + \theta_n x_n$$

This is the equation of a hyperplane in $\mathbb{R}^n$. Here the goal is to find the most optimal values of θ_s such that the hyperplane formed best fits the data.

The hyperplane thus formed is used to estimate the values of the dependent variable y for any new data point. The optimal values of θ_s can be obtained by minimising the cost function. This is done using the technique of gradient descent.

2.3.1 Cost Function

As the name suggests, cost function gives us the cost which we need to pay in terms of accuracy in case we are using the predicted value y_{pred} in place of the actual value y. Cost functions are used to estimate how badly models are performing. Thus our goal is minimising the cost function so as to get accurate results. Generally mean squared error between the actual value y and the predicted value y_{pred}, is used as the cost function. It is a function of θ_s as we change that to get the minimum value of the cost function [28, 4].

Let Θ be the vector in $\mathbb{R}^{n+1}$ having coordinates as $(\theta_0, \theta_1, \ldots, \theta_n)$. The cost function is a function J of Θ such that [28, 4]:

$$J : \mathbb{R}^{n+1} \to \mathbb{R}$$

$$J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - y_{pred}^{(i)})^2$$

Here m is the number of data points with which the model is trained.

Since cost function $J(\theta)$ is the mean squared error, it is a convex function; thus has only one minima , and hence convex minimisation algorithms can be used. Thus the multiple linear regression model learns i.e. find optimal values of θ_s, by minimising the cost function.

2.3.2 Gradient Descent

Now that we know that models learn by minimizing the cost function, our objective is to find its minima. This can be done using the gradient descent algorithm. Gradient descent is a first-order iterative optimization algorithm for finding a local minima of any function using gradient of that function. Since the gradient of the function is the vector of partial derivatives of the function with respect to each of the independent variables, the necessary condition for gradient descent to work efficiently is the differentiability of the function.

The most important property of a gradient of any function f is the fact that direction of the gradient is the direction in which the function increase the fastest. So in order to reach the point representing the local minima from any other point, the direction opposite to the direction of gradient at the point must be traversed. This path is the fastest path as the direction opposite to the direction of the gradient vector will be the direction in which the function decrease the fastest [28, 4].
Mathematically gradient of a function $f(x, y, z)$, which is a function of independent variable x, y and z, is given by :

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$$

It is a vector of the partial derivatives of the function w.r.t each independent variable, whose magnitude gives the rate of change of the function.

In gradient descent algorithm, we initially choose a random point in the Θ domain ($\Theta^{(0)}$), i.e. a random value for each θ. The value of the gradient of the cost function at that point is

calculated. The value of the gradient is used to get the next point. The main idea is to take a step in the opposite direction of the gradient (or approximate gradient) from the current point to get the next. This process is repeated until we converge to a point. That point $\Theta^{(n)}$ after an such iterations is the approximate local minima [28].

Since gradient descent is an iterative algorithm; suppose at k^{th} iteration, $\Theta^k = (\theta_0^{(k)}, \theta_1^{(k)}, \ldots, \theta_n^{(k)})$ is the point that is the value of independent variables at that iteration during the optimisation. Then the value of point Θ at the next iteration $k+1$ is given by :

$$\Theta^{(k+1)} = \Theta^{(k)} - \alpha \nabla J(\Theta^{(k)})$$

Here α is the **learning rate** of the algorithm which determines the step size at each iteration while moving toward a minimum of a loss function. $J(\Theta)$ is the cost function we need to minimise.
The term $\alpha \nabla J(\Theta^{(k)})$ is subtracted from $\Theta^{(k)}$ because we want to move in the opposite direction of the gradient, toward the local minimum.

2.4 Neural Networks

Neural networks or Artificial Neural Networks are supervised machine learning algorithms that are similar to the networks of neurons in our brain and hence the name. Just like how the nervous system neurons learns from the past experience and previously seen data, similarly ANNs draw predictions learning from the previously encounterd data. The can be used for classifications and regression. ANNs are non linear models which can be used to describe complex relationships between the output and the input variables. Thus it can be used to uncover complex patterns. It can be used to perform different types of tasks such as image recognition, text classification, medical imaging etc [19, 4].

The are build up of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node in a layer is connected to all other nodes in the previous and the next layers and each connections has its weights and biases. Neural networks depends on the training data to learn and improve their prediction accuracy over time [19].
There are 3 parts that make up the architecture of a basic Neural Network.

- Neurons
- Weights
- Biases

The value of a particular neuron is determined by summation of the product of weights and input data from all its input neurons and the bias term and then applying a suitable activation function to it and return the output which maybe the input to the next layer or the final output.

The weights and biases are what is to be optimised for accurate results. They are initially random and eventually optimised using rounds of forward propagation and backpropagation. This is coupled with an optimising algorithm like gradient descent or stochastic gradient descent. We follow the same process as in the case of regression that is of minimising the error or cost function and changing the weights and biases accordingly. In a dense neural network each node in one layer is connected to each and every node in the next layer, and is thus the input to all the nodes in the next layer.

2.4.1 Error function

The error function, as the name suggests, depicts the error between actual output and the predicted output from the network. In case of continuous value prediction, the mean squared error is mostly used along with mean absolute error. In case of categorical prediction, cross entropy can be used [19].

In this model, as stochastic gradient descent is used, the error function is half of the squared difference of predicted value of output y_{pred} and the actual output y. Error E is thus:

$$E = \frac{1}{2}(y - y_{pred})^2$$

This error is different for each data point.

2.4.2 Layers

It is a combination of neurons which takes input from previous layer, applies weights and biases and activation function to it, and gives the output which acts as input to next layer. Layers of neurons between actual input and actual output layers are called hidden layers. They are the main components of any neural network and helps to deal with complex relations and non linearity [19].

2.4.3 Forward propagation

It is the process of feeding of the input data in the forward direction through the network. Each hidden layer accepts the input data, processes it as per the activation function and passes to the successive layer. The output of one layer is the input for the next layer. In this step, the input is applied weights. The weighted sum of inputs is added a bias term for each node. An activation function is applied to this weighted sum plus the bias which is the value of that node. This continues till the last layer or the output layer and the value of node at the output layer is y_{pred} and is compared with actual output y [19].
Suppose the layer $k-1$ has n nodes in a dense network , the value of i^{th} node in the k^{th} layer, $o_i^{(k)}$, is given by:

$$o_i^{(k)} = f\left(\sum_{j=1}^{n} w_{ji}^{(k-1)} o_j^{(k-1)} + b_i^{(k-1)}\right)$$

where f is the appropriate activation function.

2.4.4 Backpropagation

Backpropagation is an algorithm for optimisation of weights and biases of neural networks using stochastic gradient descent or batch gradient descent algorithms. Given a neural network and a cost or error function, this algorithm calculates the gradient of the cost/ error function with respect to the weights and biases of the network. The error is directly related to the weights and biases of the last layer. But the weights and biases of the last layer are dependent on those of the second last layer. Since the network is woven is such a fashion that the weights and biases of any layer is dependent on those of its previous layer, the error function is indirectly dependent on all the weights and biases.

The gradients of are initially calculated with respect to the weights of the last layer. Then the gradients of the weights are calculated for the interior layers. They are calculated by using the product rule. The gradients of one layer are used in calculation of gradients in its previous

layers and this process continues till the input layer. [19, 23].

Getting the optimum values of weights and biases is achieved using the backpropagation algorithm. Learning rate parameter α is selected appropriately and the weights and biases (Θ) are optimised following the gradient descent algorithm as shown below:

$$\Theta^{t+1} = \Theta^t - \alpha \frac{\partial E}{\partial \Theta} [23]$$

where Θ^t denotes the parameters of the neural network at epoch t in gradient descent. So each and every weight and bias are updated using its gradient w.r.t. to the error. Since the hidden layers has no actual output to get the error, the gradients are calculated with the main error E using chain rule.

2.4.5 Activation Function

Activation function are functions applied to the value of each node when it is transfered to the next layer. They take input and outputs a value in a restricted range. They are used to scale down the output values. These are non-linear which add non linearity to the network and hence helps in complex tasks [26]. Common activation functions are:

- Sigmoid
- ReLU
- Leaky ReLU
- tanh
- ELU

Leaky ReLU has been used as the activation function. It is given by :

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ ax & \text{otherwise} \end{cases} \tag{1}$$

where a is a constant. It is actually non-differentiable at $x = 0$, but is made differentiable by making $f'(0) = 0$

2.4.6 Stochastic Gradient Descent

Normal gradient descent algorithm calculates the error function and its gradient taking into account all the training examples and update the parameters/ weights using that. Stochastic gradient descent only the other hand calculates the error gradient using a single, randomly chosen, training example and updates the weights. This process is carried for all the data points. It can be given by the formula [14]:

$$\Theta^{(k+1)} = \Theta^{(k)} - \alpha \nabla J(\Theta^{(k)}, x^{(j)}, y^{(j)})$$

where $x^{(j)}, y^{(j)}$ are the input and output j^{th} training example respectively. It is achieved randomly shuffling the input data and then selecting the data points sequentially.

The error function is not always convex and may contain many local minima. During the process of optimisation using gradients, if a local minima is attained, the algorithm can get stuck at that point and the global/ true minima of the function is not attained. The main advantage of this method is that we are not stuck at any local minima of the error function. This is due

to the stochastic nature of the algorithm, a random point next selected which might not be in the same region, hence the iteration is moved away from a local minima. Apart form this, it is computationally faster and less expensive than normal gradient descent.

2.4.7 Gradient/ Norm Clipping

In case of neural networks, the error function is dependent on all the weights of all the layers. This makes it a complex non-convex function. In certain cases, the curvature of the error function is such that gradients at certain points are very high. As the weights are updated using the gradients, the updated weights can also be very high and can also overflow in certain cases. This will hence result in non-convergence and high computational cost.
To eliminate this, gradient norm clipping is used. The gradient is used to update the weights because of its direction. The magnitude of the gradient is just used as the distance to travel in the required direction. The direction of any vector is conserved even after normalising it. Normalisation of a vector is the scalar multiplication of the vector by the reciprocal of its magnitude. It transforms the vector into a unit vector in the same direction. So to eliminate the problem of exploding gradients, the gradient is normalised when it crosses a threshold value, thereby reducing the exploding magnitude of the gradient and preserving the direction [9].
If the vector $v = (v_1, v_2 \cdots v_n)$, then normalisation of v is given by :

$$\overline{v} = \frac{v}{\sqrt{\sum_{i=1}^{n} v_i^2}}$$

Suppose the selected threshold for gradient is c, then if the magnitude of the gradient increases this threshold, it will be scaled down following the below given criterion.

$$\text{If } ||\nabla J|| \geq c \implies \nabla J = \frac{c}{||\nabla J||} \nabla J$$

2.5 Regularisation

In machine learning, regularisation is the process of making a solution simpler and interpretable. Regularisation can be explicit and implicit. Explicit regularisation refers to addition of extra terms such as penalties, constraints etc. to the optimisation function. These terms penalises wrong or complex solutions thereby achieving simplicity. All other types of regularisation are implicit. They may include processes such as discarding outliers, stopping early in optimisation, etc.

One major problem of machine learning countered by regularisation is overfitting. Overfitting is the phenomenon when the machine learning model fits the training data very accurately. This results in a model not being accurate when predicting for new/ unseen example thereby defeating the entire purpose. Overfitting results in high generalisation error i.e. poor prediction and high error for unseen data. So to combat overfitting, the algorithm must reduce generalisation error without reducing the training error. This can be done effectively using regularisation.

Machine learning algorithms like regression are generally regularised using three types of regularisation:

- Ridge regression or L2 Regularisation
- LASSO regression or L1 Regularisation
- Elastic Net i.e. combination of Ridge and LASSO

Each of these have a general structure that is used to define the error term or the loss function of a regularised regression scheme. It can be given by :

$$\text{Error}_{regularised} = \text{Error}_{least\ squares} + \text{Penalty term} \quad (2)$$

The Penalty term used in the above equation determines the type of regularisation used.

2.5.1 LASSO regression

LASSO is the acronym for Least Absolute Shrinkage and Selection Operator. Shrinkage refers to shrinking or reducing the weights of the features thus avoiding overfitting. Selection means selecting only those features for the model that are important for prediction and rejecting others. In selection process, the weights are unimportant features is reduced to zero which implies they do not contribute to prediction task. Thus LASSO performs the combined task of regularisation and feature selection simultaneously. It gives a more interpretable model and sparse regression can also be achieved. Sparse models refers to models where the weights or coefficients of a lot of features are zero [27, 21].

LASSO uses summation L1 norms of each weight of the feature as the penalty term. The error term is given by :

$$E(\Theta)_{\text{LASSO}} = \sum_{i=1}^{m}(y^{(i)} - \sum_{j=1}^{p}\theta_j x_j^{(i)})^2 + \lambda \sum_{j=1}^{p}|\theta_j|$$

λ is the regularisation/ tuning parameter which must be optimally determined. θ_s are the weights/ coefficients.
As we can see that the penalty term is the sum of absolute value of weights or L1 norm of the weight vector. The presence of this L1 norm helps in inducing the sparsity. The set of all vector with a given fixed L1 norm is in shape of a diamond with "corners". The LASSO uses L1 norm of weight vector, so there are "corners" in the constraint corresponding to the diamond. The least square error has an L2 norm which means the vectors having same norm forms an ellipse. If the ellipse i.e. least square error "hits" / equals the diamond i.e. penalty term at the corner i.e. on the axis of any feature, it means the weight of that feature shrinks to zero. This gives sparsity to the model. It can be seen from the figure which shows the same phenomenon in 2 dimensions:

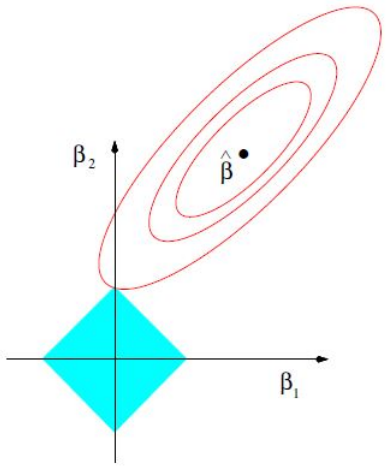

Figure 1: Pictorial representation of sparsity induced by LASSO regression

Here β_1, β_2 are the two weights/ coefficients of the features.
L1 norm is inherently non - differentiable. Due to this gradients do not exist at certain points, hence normal gradient descent algorithm cannot be used for optimising the error function. Coordinate descent algorithm have to be used in this case.

2.5.2 Subgradients

Gradients have the property that they lower bound convex functions. It can be expressed as :

$$f(b) \geq f(a) + \nabla f(a)(b - a)$$

for two points a, b and $b > a$ and convex function f.

Subgradients generalise gradients at non differentiable points. A subgradient of the function at any point a is any plane that lower bounds the function at that point. If V is one of the subgradients of function g at point a i.e. $V \in \partial g(a)$ then:

$$f(b) \geq f(a) + V(b - a)$$

$\partial g(a)$ is the set of all subgradients of g at point a

2.5.3 Coordinate descent algorithm

Coordinate descent algorithms are used to solve optimization problems by successively performing approximately minimizing the objective function along each of the coordinate direction. Thus in case of LASSO, the error function is sequentially minimised along each of the feature. Due to this we can approximate the derivative of the error function, especially the penalty term by using the subgradient instead of gradients [29].

For each of the weights, the penalty term is non-differentiable at point $\theta_i = 0$ for each i. Three different planes/ subgradients are used to approximate the gradient of the penalty term. They are:

$$\partial|\theta_i| = \begin{cases} -1 & \text{if } \theta_i < 0 \\ 0 & \text{if } \theta_i = 0 \\ 1 & \text{if } \theta_i > 0 \end{cases}$$

where $\partial|\theta_i|$ is the subgradient of $|\theta_i|$ which approximates the gradient $\frac{\partial|\theta_i|}{\partial\theta_i}$.
Partially differentiating the error function $E(\Theta)$ w.r.t each θ_k gives:

$$\frac{\partial E(\Theta)}{\partial \theta_k} = -2\sum_{i=1}^{m}(y^{(i)} - \sum_{j=1}^{p}\theta_j x_j^{(i)})x_k^{(i)} + \begin{cases} -\lambda & \text{if } \theta_k < 0 \\ 0 & \text{if } \theta_k = 0 \\ \lambda & \text{if } \theta_k > 0 \end{cases}$$

Since we are minimising the error, we equate the equation to zero i.e. $\frac{\partial E(\Theta)}{\partial \theta_k} = 0$ to get the optimum value of θ_k.

$$\implies -2\sum_{i=1}^{m}(y^{(i)} - \sum_{j=1}^{p}\theta_j x_j^{(i)})x_k^{(i)} + \begin{cases} -\lambda & \text{if } \theta_k < 0 \\ 0 & \text{if } \theta_k = 0 \\ \lambda & \text{if } \theta_k > 0 \end{cases} = 0$$

If each feature is normalised, then for each feature k :

$$\sum_{i=1}^{m}(x_k^{(i)})^2 = 1$$

So the above equation can be re-written as:

$$-2\sum_{i=1}^{m}(y^{(i)} - \sum_{\substack{j=1 \\ j\neq k}}^{p}\theta_j x_j^{(i)})x_k^{(i)} + 2\sum_{i=1}^{m}\theta_k(x_k^{(i)})^2 + \begin{cases} -\lambda & \text{if } \theta_k < 0 \\ 0 & \text{if } \theta_k = 0 = 0 \\ \lambda & \text{if } \theta_k > 0 \end{cases}$$

Let the term ρ_k denote:

$$\rho_k = \sum_{i=1}^{m}(y^{(i)} - \sum_{\substack{j=1 \\ j\neq k}}^{p}\theta_j x_j^{(i)})x_k^{(i)}$$

So the above equation can be rewritten as:

$$-2\rho_k + 2\theta_k + \begin{cases} -\lambda & \text{if } \theta_k < 0 \\ 0 & \text{if } \theta_k = 0 = 0 \\ \lambda & \text{if } \theta_k > 0 \end{cases}$$

$$\implies \begin{cases} -2\rho_k + 2\theta_k - \lambda & \text{if } \theta_k < 0 \\ [-2\rho_k - \lambda, -2\rho_k + \lambda] & \text{if } \theta_k = 0 = 0 \\ -2\rho_k + 2\theta_k + \lambda & \text{if } \theta_k > 0 \end{cases}$$

Rearranging the above equation we get:

$$\theta_k = \begin{cases} \rho_k + \frac{\lambda}{2} & \text{if } \rho_k < -\frac{\lambda}{2} \\ 0 & \text{if } \rho_k \in [-\frac{\lambda}{2}, \frac{\lambda}{2}] \text{ [29]} \\ \rho_k - \frac{\lambda}{2} & \text{if } \rho_k > \frac{\lambda}{2} \end{cases}$$

Thus directly get the optimum value of θ_k using this equation. We repeat this step for all θ_k till convergence.

2.6 Data Smoothing

Data collected experimentally is never ideal and is affected by noise/ garbage. This may be due to various reasons such as non-ideal condition for carrying out experiments, inaccurate measurements and instruments, inefficient practices and largely because of the fact that no process in the real world is ideal and it is bound to be affected by external randomness. This results in errors in the actual values of the property to be measured. In order to successfully and efficiently use that data, it must be treated to reduce the impact of noise. Thus the data must be filtered in such a way that the noise is filtered and close to ideal, smooth data is obtained [13].

Noise is generally of high frequency, much higher than the data/ original signal. In order to reduce or eliminate that a low-pass filter must be used. As the name suggests, a low-pass filter allows the low - frequency components of the signal to pass through and filters out all the high frequency components of the signal. Thus treating the signal with a low-pass filter will greatly eliminate the high frequency noise components from the signal and clean the signal. Thus the processed signal is now smooth, without spikes or purterbances [13].
The following image depicts a signal smoothed from a noisy signal.

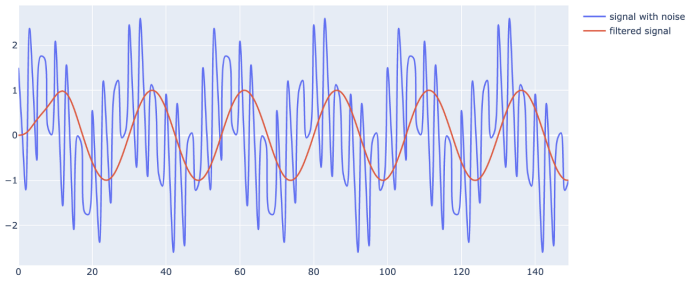

Figure 2: Smoothing of Noisy Data

2.6.1 Savitzky Golay Filter

It is a low-pass filter that uses the idea of polynomial approximation using neighbouring points. The idea is to fit a polynomial of a higher degree to data points within a window of certain points. A window is a set of neighbouring points. Then the value of the center point of the window is approximated by the value of the polynomial at that point. Suppose f is the original function that generates the signal. The point at the center of the window is suppose x_k, and the polynomial is $p_k(x)$, then [20]:

$$f(x_k) \approx p_k(x_k)$$

The smoothened value of the signal at point x_k is $p_k(x_k)$. The polynomial function is fitted using least square. That is, if the polynomial is of degree n, then the $n+1$ coefficients of the polynomial are obtained using the method of least squares. The window is then moved one step ahead with its center now being x_{k+1}. A new polynomial p_{k+1} is fitted to the points in that window using the method of least squares. And the smoothened value of $f(x_{k+1})$ is $p_{k+1}(x_{k+1})$. The window is moved till all the points are covered. The number of points in the window must be odd [20].
This procedure results in smoothing action and the high frequency noise is reduced to a great extent. This is the working principle of Savitzky Golay filter. The image below depicts a smoothing of noisy signal using Savitzky Golay filter.

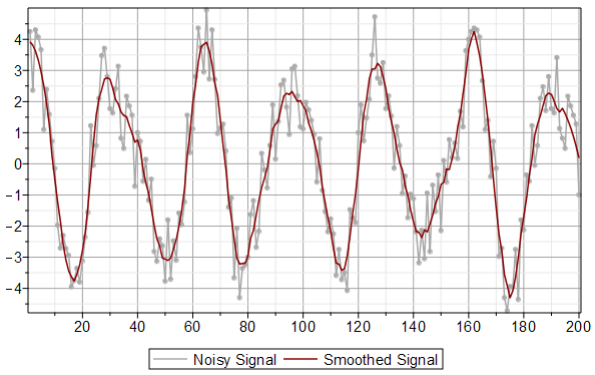

Figure 3: Savitzky Golay Filter for smoothing noisy signal

2.7 Legendre Polynomials

Legendre Polynomials are the set or system of polynomials that are orthogonal to each other. It is understood that any point in an n dimensional vector space can be expressed as linear combination of n basis vectors which are orthogonal to each other. Similarly since the Legendre polynomials are orthogonal to each other, they can be used as basis vectors for the functional space. Thus any n degree polynomial can be expressed as a linear combination of 1 to n degree Legendre polynomials. [12]
If $P_n(x)$ and $P_m(x)$ are Legendre polynomials of degree n and m respectively then the condition for orthogonality can be given as:

$$\int_{-1}^{1} P_n(x)P_m(x)dx = 0$$

The generating function for $P_n(x)$ is given by:

$$\frac{1}{\sqrt{1-2xt+t^2}} = \sum_{n=0}^{\text{inf}} P_n(x)t^n$$

[12] Thus the coefficients of t^n in the above equation are the Legendre polynomials $P_n(x)$.
The figure below depicts Legendre polynomial of degree 0 to 5 from $x = -1$ to $x = 1$

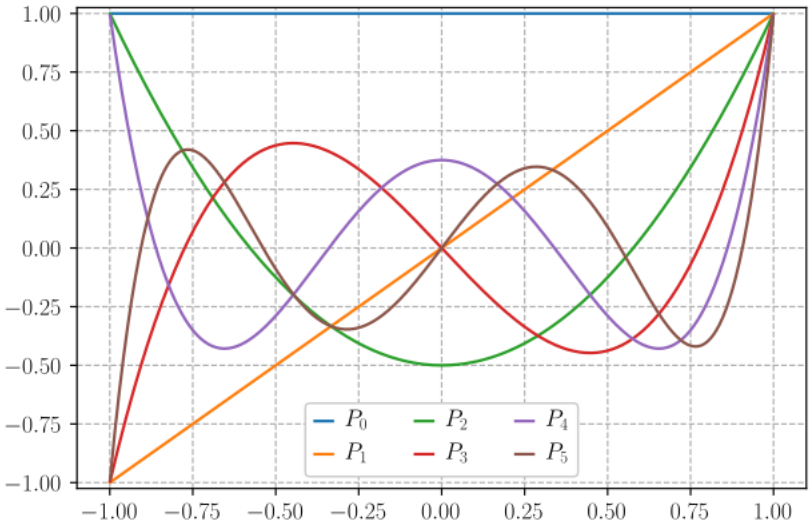

Figure 4: Legendre Polynomials

3 Getting data for training and testing

Every model needs different data sets for its training i.e. for getting the optimal weights and biases and θ_s and for testing i.e. check how the model performs while dealing with new data. Both these datasets are generated. The advection equation and Burgers' equation are solved numerically using forward difference technique for certain initial and boundary conditions [18]. The **boundary condition used is periodic** in all the cases.

We know that the partial derivatives can be written as :

$$\frac{\partial u}{\partial t} \approx \frac{u(x, t+\Delta t) - u(x,t)}{\Delta t} [11,\ 18]$$

$$\frac{\partial u}{\partial x} \approx \frac{u(x+\Delta x, t) - u(x-\Delta x,t)}{2\Delta x} [11,\ 18]$$

where Δx and Δt are small changes or step size in x and t respectively.

Since we are given the initial condition we attempt to find the value of the velocity field for the next time stamp from the previous time stamp. So we find an equation for $u(x, t_k+\Delta t) = u_{x,t_{k+1}}$ representing the velocity field at spatial point x and time stamp $t+1$ for some given t_k.

For the advection equation, one dimensional Burgers' inviscid equation and one dimensional Burgers' equation, the data used is simulated using numerical techniques which is pure but gaussian noise is added to the numerically simulated data for 2 dimensional Burgers' equation.

3.1 Advection equation

The advection equation is :

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0$$

the value of c is taken as one.
The initial condition for training data sets is :

$$u(x,0) = \sin(x)$$

So we can write :

$$\frac{\partial u}{\partial t} = -\frac{\partial u}{\partial x}$$

$$\implies \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} = -\frac{u(x+\Delta x,t) - u(x-\Delta x,t)}{2\Delta x}$$

$$\implies \frac{u(x,t+\Delta t)}{\Delta t} = \frac{u(x,t)}{\Delta t} - \frac{u(x+\Delta x,t) - u(x-\Delta x,t)}{2\Delta x}$$

$$\implies u(x,t+\Delta t) = u(x,t) - \frac{\Delta t}{2\Delta x}\left(u(x+\Delta x,t) - u(x-\Delta x,t)\right)$$

$$\implies u_{x_j,t_{k+1}} = u_{x_j,t_k} - \frac{\Delta t}{2\Delta x}(u_{x_{j+1},t_k} - u_{x_{j-1},t_k}) [11,\ 18]$$

We use the above equation to generate the data for advection equation and given initial and boundary condition. Here the boundary and step sizes are is :

$$x_{left} = 0, x_{right} = 2\pi, \Delta x = 0.01$$

$$t_{start} = 0, t_{end} = 1, \Delta t = 0.001$$

The graph for **training data** for $u(x,0) = \sin(x)$ is given here shows data :

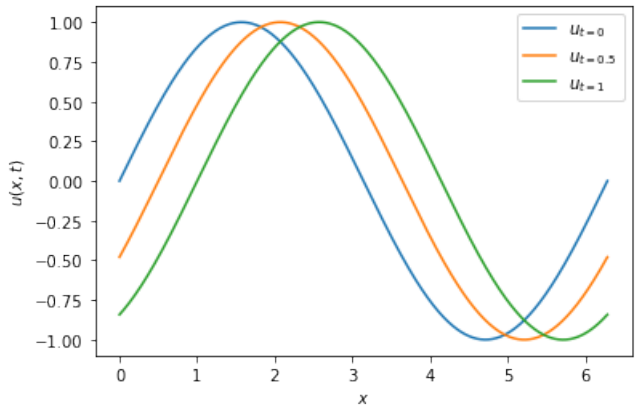

Figure 5: Training data for advection equation

The graph for **testing data** for $u(x, 0) = \cos(x)$ is given here shows data :

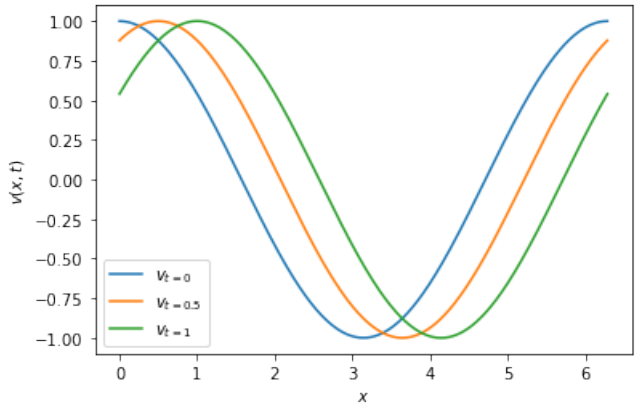

Figure 6: Testing data for advection equation

3.2 Burgers' inviscid equation

The advection equation is :

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0$$

So we can write :

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x}$$

$$\implies \frac{u(x, t+\Delta t) - u(x,t)}{\Delta t} = -u(x,t)\left(\frac{u(x+\Delta x, t) - u(x-\Delta x, t)}{2\Delta x}\right)$$

$$\implies \frac{u(x, t+\Delta t)}{\Delta t} = \frac{u(x,t)}{\Delta t} - u(x,t)\left(\frac{u(x+\Delta x, t) - u(x-\Delta x, t)}{2\Delta x}\right)$$

$$\implies u(x, t+\Delta t) = u(x,t) - \frac{\Delta t}{2\Delta x}(u(x,t)\,(u(x+\Delta x, t) - u(x-\Delta x, t))$$

$$\implies u_{x_j, t_{k+1}} = u_{x_j, t_k} - \frac{\Delta t}{2\Delta x} u_{x_j, t_k}(u_{x_{j+1}, t_k} - u_{x_{j-1}, t_k})$$

$$\implies u(x, t + \Delta t) = u(x, t) - \frac{\Delta t}{2\Delta x}(u(x, t)\,(u(x + \Delta x, t) - u(x - \Delta x, t))\,[17]$$

The above equation is used to generate the data for Burgers' equation and given initial and boundary condition. Here the boundary and step sizes are is :

$$x_{left} = 0, x_{right} = 2\pi, \Delta x = 0.025$$

$$t_{start} = 0, t_{end} = 1, \Delta t = 0.0025$$

The graph for **training data** for $u(x, 0) = \sin(x)$ is given here shows the data :

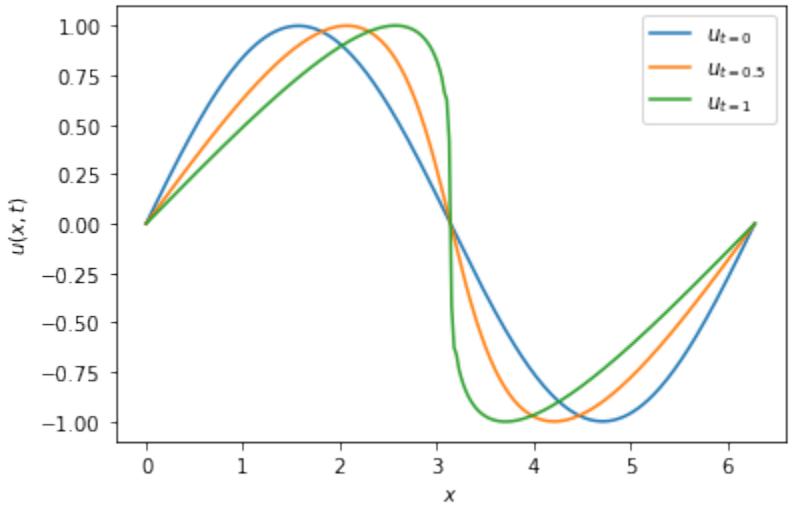

Figure 7: Training data for inviscid Burgers' equation

The graph for **testing data** for $u(x, 0) = \cos(x)$ is given here shows the data:

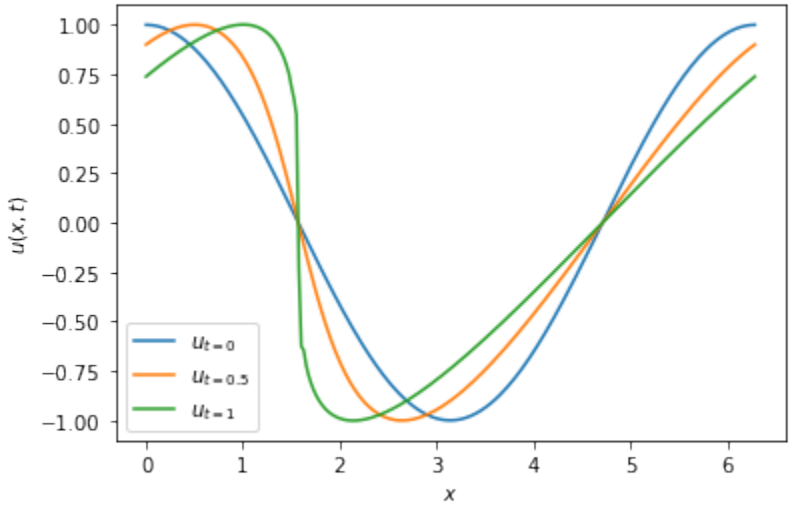

Figure 8: Testing data for inviscid Burgers' equation

3.3 Burgers' Equation for clean data

This equation models advection and diffusion process and has double degree partial differentials. Therefore, normal forward difference method cannot be employed for numerically simulating the data.

The equation is given by :

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial^2 x}$$

. It can be also written as:

$$u_t + u \cdot u_x = \nu u_{xx}$$

. Integrating this equation w.r.t x from limits $x_{j-\frac{1}{2}}$ to $x_{j+\frac{1}{2}}$:

$$\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u_t\,dx + \int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u \cdot u_x\,dx = \nu\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u_{xx}\,dx[17]$$

Each term can be written as

$$\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u_t\,dx = u_t(x_j,t)(\Delta x)$$

$$\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u \cdot u_x\,dx = \frac{1}{2}[u^2]_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} = \frac{1}{2}(u(x_{j+\frac{1}{2}},t)^2 - u(x_{j-\frac{1}{2}},t)^2)$$

$$\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u_{xx}\,dx = \nu[u_x]_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} = \nu\left(\frac{u(x_j,t)-u(x_j-\Delta x,t)}{\Delta x} - \frac{u(x_j+\Delta x,t)-u(x_j,t)}{\Delta x}\right) =$$

$$\nu\frac{u(x_j+\Delta x,t) - 2u(x_j,t) + u(x_j-\Delta x,t)}{\Delta x}$$

So the equation can be written as :

$$\frac{\partial u(x_j,t)}{\partial t}(\Delta x) = \nu\frac{u(x_j+\Delta x,t) - 2u(x_j,t) + u(x_j-\Delta x,t)}{\Delta x} - -\frac{1}{2}(u(x_{j+\frac{1}{2}},t)^2 - u(x_{j-\frac{1}{2}},t)^2)$$

$$\boxed{u(x_j,t+\Delta t) = u(x_j,t) + \Delta t\left(\nu\frac{u(x_j+\Delta x,t) - 2u(x_j,t) + u(x_j-\Delta x,t)}{(\Delta x)^2} - \frac{1}{2}\frac{(u(x_{j+\frac{1}{2}},t)^2 - u(x_{j-\frac{1}{2}},t)^2}{\Delta x}\right)}[17]$$

Here $u(x_{j\pm\frac{1}{2}},t)$ is the average value of $u(x_j \pm \Delta x,t)$ and $u(x_j,t)$.
The above equation is used to get the values of velocity at any point x_j for a given time stamp, from the velocity value of previous times stamps.

The above equation is used to generate the data for Burgers' equation and given initial and boundary condition. Here the boundary and step sizes are is :

$$x_{left} = 0, x_{right} = 4\pi, \Delta x = 0.05$$

$$t_{start} = 0, t_{end} = 12, \Delta t = 0.005$$

The viscosity is taken to be $\nu = 0.2$.

The graph for **training data** for $u(x,0) = \cos(x)$ is given here shows the data :

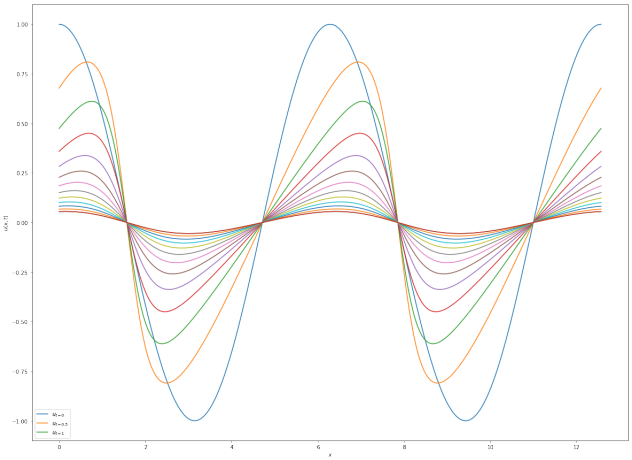

Figure 9: Training data of Burgers' equation

3.4 Burgers' equation for noisy data

Since the equation is same, the same numerical scheme is used to to generate the data. The generated data is then corrupted using Gaussian noise. Gaussian noise is added to the velocity u values at each grid point.
Thus the numerical scheme is :

$$u(x_j, t+\Delta t) = u(x_j,t) + \Delta t \left(\nu \frac{u(x_j+\Delta x, t) - 2u(x_j,t) + u(x_j - \Delta x, t)}{(\Delta x)^2} - \frac{1}{2} \frac{(u(x_{j+\frac{1}{2}},t)^2 - u(x_{j-\frac{1}{2}},t)^2}{\Delta x} \right) \text{[17]}$$

The initial condition used to generate the data is : $u(x,0) = 0.3\sin(3x) - 0.2\cos(2x)$ *The boundary condition is periodic.*
the boundary and step sizes are is :

$$x_{left} = 0, x_{right} = 10\pi, \Delta x = 0.05$$

$$t_{start} = 0, t_{end} = 60, \Delta t = 0.005$$

The viscosity is taken to be $\nu = 0.1$.
The generated data is shown in the image given below:

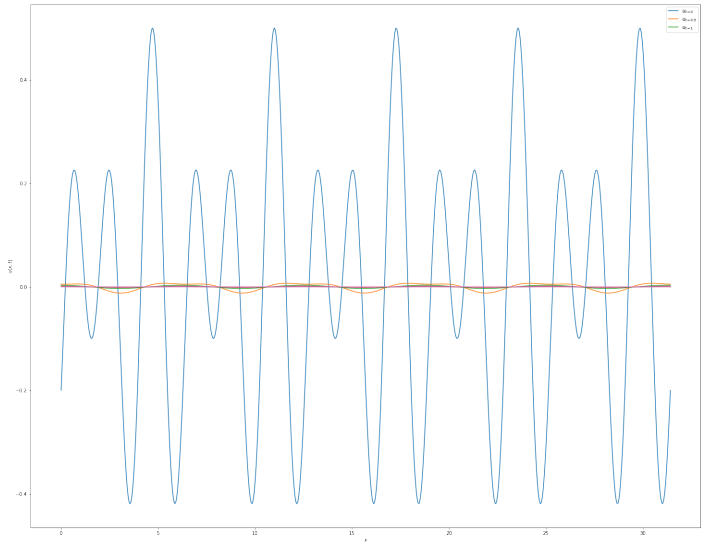

Figure 10: Smooth data of Burgers' equation

Now to this data, Gaussian noise is added using the formula given below:

$$\tilde{u}(x_j, t_i) = u(x_j, t_i) + 0.05 \cdot u(x_j, t_i) \cdot \epsilon_{j,i}$$

Here $\tilde{u}(x_j, t_i)$ is the corrupted/ noisy velocity value, $u(x_j, t_i)$ is the value generated from the numerical scheme mentioned above at point(x_j, t_i) and $\epsilon_{j,i}$ is the Gaussian error/ noise for that point.
$\epsilon_{j,i}$ is selected randomly from a Gaussian distribution with mean 0 and variance is 1. Each ϵ is independent.
The figure representing the noisy/ corrupted data is shown below:

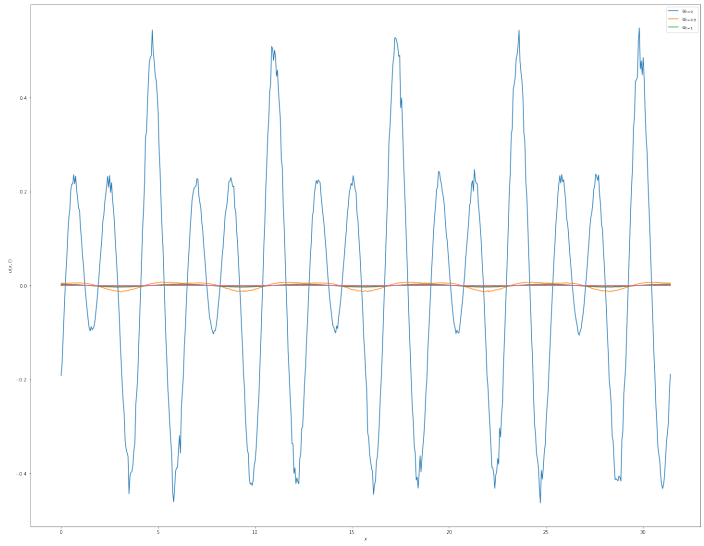

Figure 11: Noisy data of Burgers' equation

4 NN Model to predict velocity for advection and inviscid Burgers' equation

4.1 Procedure for training and testing the model

Generally training and testing follow the same procedure of finding the error of the predicted value from the actual value. And the actual data is given as an input in both the cases
Also , the training and testing are done on parts of the same dataset. But a different procedure is followed here.

4.1.1 Training

In this case the training is done on the data obtained by using the initial condition $u(x, 0) = \sin(x)$.
The training of both the model is done in the original fashion of given data points as inputs and checking the accuracy of prediction and optimising parameters based on the error of prediction.

$$\implies (u_{pred})_{x,t_{k+1}} = g(u_{x,t_k}) \qquad \forall \;\; k$$

4.1.2 Testing

The testing is done on the data obtained using $u(x, 0) = \cos(x)$.
But the testing is done in a slightly different way. It is as follows:

- The data from the initial condition is used for predicting the value of the first time stamp, i.e. u_{x,t_1} is predicted from $u_{x,0}$.

$$\implies (u_{pred})_{x,t_1} = g(u_{x,0})$$

- For all the next time stamps, the value u predicted for the exact previous time stamp is used for prediction, instead of the original data.

$$\implies (u_{pred})_{x,t_{k+1}} = g((u_{pred})_{x,t_{k+1}}) \qquad \forall \;\; k > 1$$

4.2 Features, architecture and hyperparameters

4.2.1 Features

After trying a lot of different features, finally three features are selected both for advection equation and Burgers' Equation.

Advection equation
The three features selected for advection equation are :

$$\boxed{u_{x-1,t-1}, \qquad u_{x,t-1}, \qquad u_{x+1,t-1}}$$

Burgers' equation
The three features selected for advection equation are :

$$\boxed{u_{x,t-1} \cdot u_{x-1,t-1}, \qquad u_{x,t-1}, \qquad u_{x,t-1} \cdot u_{x+1,t-1}}$$

The output for both the cases as described earlier is $u_{x,t}$

4.2.2 Architecture and hyperparameters

Architecture of a machine learning model refers to the type of model used. The features and outputs, the number of hidden layers and number of nodes in case of a neural network.

In machine learning, a hyperparameter is a parameter whose value is used to control the learning process. Hyperparameters are selected initially before training of model. They affect speed and performance of the model and thus appropriate selection of hyperparameters is very essential.

Advection equation
A neural network is used in case of advection equation. It fits the data well. It has :

- 1 input layer with 3 nodes
- 2 hidden layers with 3 nodes and 1 nodes respectively
- 1 output layer with 1 node

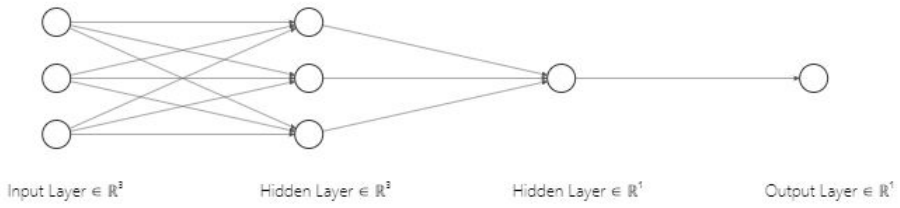

Figure 12: Neural Network architecture for advection equation

Invisid Burgers equation
A neural network is used in case of invisid Burgers' equation. It fits the data well. It has :

- 1 input layer with 3 nodes
- 2 hidden layers with 5 nodes and 3 nodes respectively
- 1 output layer with 1 node

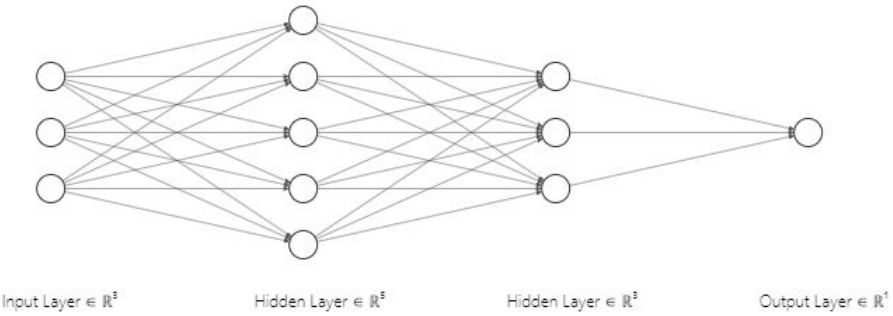

Figure 13: Neural Network architecture for inviscid Burgers' equation

4.3 Model Implementation

It is a neural network model. The neural network is developed from scratch using only numpy arrays in python. There is a neural network class, layers class and a weights and biases class The neural network class has arrays of layer class objects and similar weights and biases class objects in accordance to the number of hidden layers, nodes in each layer and input and output condition.
The schema of the classes is :

- The weight class has weight matrix and biases vector as its attributes.
- The layers class has a layer vector as its attribute with an appropriate activation function.
- The network class is the main class. Along with layers and weights and biases objects it has functions for forward propagation, backpropagation, adding layers and calculating gradients.

To form a neural network following steps are performed:

- Instantiating an object of the neural network class. It needs the number of input nodes, number of output nodes and the learning rate to instantiate.
- Calling add layers function of that object for adding appropriate number of layers to the model. The have to be passed with the number of nodes that are present.

Note : The add layers function must be used in the order of layers in the network.
Then the function is run for several epochs and the accuracy of the training prediction is measured to check if the decided number epochs are sufficient. Each epoch is a combination of one forward propagation step followed by a backpropagation step for each of the data points that are in randomly shuffled. The forward propagation changes the values of nodes in the hidden layers and output layer. The backpropagation changes the weights and biases between each layers using stochastic gradient descent.
Initially the data for training is generated using forward propagation mechanism. All the weights and biases are iniitally random.

4.3.1 Advection equation

The training is done keeping:

- Learning rate $\alpha = 0.0025$
- Epochs = 150
- Activation function = ELU

After 50 epochs very good efficiency is achieved. The model is then tested against the testing set following the pre-mentioned procedure.

4.3.2 Inviscid Burgers' equation

For this equation the training part is hindered by **exploiding gradients** problem. Due to this in the gradient calculation process, gradient norm clipping is employed. The training is done keeping:

- Learning rate $\alpha = 0.005$
- Epochs = 150
- Activation function = ReLU

After 150 epochs very good efficiency is achieved. The model is then tested against the testing set following the pre-mentioned procedure.

4.4 Results

Results of the experiment are gauged in terms of how the model performs when using the testing data as the input. The prediction of the training data and its accuracy is visualised using graphs that shows the actual value of the testing output and predicted value of the output through the model.

4.4.1 Advection Equation

The following figure shows the values of $u_{calculated}$ i.e. actual output and $u_{predicted}$ plotted verses x for 4 time stamps

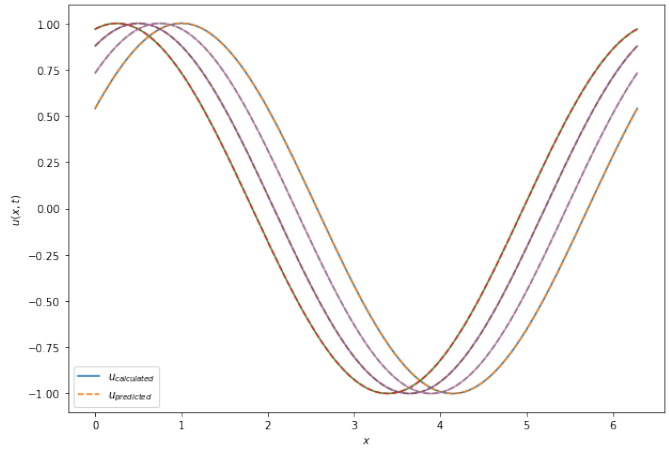

Figure 14: Neural network test results for advection equation

4.4.2 Burgers' Equation

The following figure shows the values of u and u_{pred} i.e. the predicted output plotted verses x for 3 time stamps

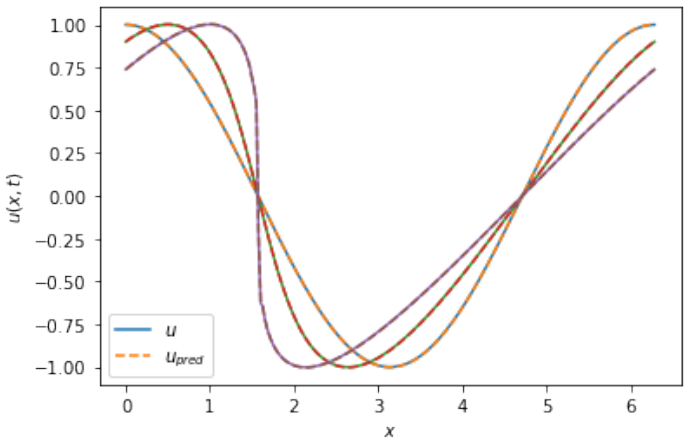

Figure 15: Neural network test results for inviscid Burgers' equation

Thus it can be concluded that such simple fluid systems can be approximately modelled using neural network machine learning models.

5 Reconstruction of governing equations from data

The velocity data for a fluid for given initial and boundary conditions is collected. From that data, the goal is to reconstruct the PDE that governs the velocity and behaviour of the fluid. This PDE can be used to predict the future behaviour of the fluid.

5.1 Problem Formulation

To find the PDE or a system of PDE's governing the system from the data, appropriate features, input and output must be used. PDE's are implicit function of velocity, its different powers, its partial differentials of different degrees w.r.t space variables and time, and their product terms. Since the flow of fluids changes with time, the rate of change of velocity with time is an appropriate output or LHS of the equation, as expect for steady state condition, the temporal derivative is non-zero. All other possible partial differentials, velocity and its power terms and their products can be used as features. Thus the time derivative of velocity is expressed as a function of all other terms. The goal is to find this function. The function is assumed to a linear combination of the input terms i.e. features.
A linear combination can be expressed as:

$$Y = X\Theta$$

Y has the values of output feature, X has the values of different input features and Θ represents the coefficients that define the linear combination.

Since the equation is only one dimensional in space domain, the equation can be written as:

$$u_t = f(u, u^2 \cdots, u_x, u_{xx} \cdots, u_{tt}, \cdots, u.u_x, u.u_t, u.u_{xx}, u^2.u_x, u^2.u_x x \cdots)$$

The selected input features are :

$$\left(u, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}, u\frac{\partial u}{\partial x}, u\frac{\partial u}{\partial t}, u\frac{\partial^2 u}{\partial x^2}, u^2, u^2\frac{\partial u}{\partial x}, u^2\frac{\partial^2 u}{\partial x^2}\right)$$

These are taken considering the nature of equations that are generally encountered when working with fluid systems. These can vary depending on the problem and type of data at hand. In this case, the terms of the linear combination denote:

- Y is 1D vector of values of u_t.
- X is matrix having columns as values of different features as discussed .It can be viewed as library of features.
- Θ represents the 1D coefficients/ weights vector.

The goal is to find appropriate value of Θ.

5.2 Processing raw data to generate input and output for the model

To train and test the model the data must be processed and the values of input features and the output feature must be calculated. In this model the input and output features used are the partial derivatives and their products with the velocity terms and its powers. Thus to generate the input and output data, these derivatives must be computed.

5.2.1 Burgers' equation with clean data

Since the data i.e. velocity (u) values are simulated directly from the equation using the numerical scheme mentioned, thus the error in those values is negligible. As the error in the velocity values is negligible, it is safe to use **central difference** to calculate the derivatives. The error in calculating the derivatives is tolerable.
Each data point of the input is the 9 dimensional vector of value of the input features at each grid point (x_k, t_i) for all $j = 1, 2, \cdots N$ and $i = 1, 2, \cdots M$. Where N is the number of points in x dimension and M is the number of points in t dimension. The values of the product terms are the product of individual term at each grid point. Same is the case with output value for that input vector. So for a grid point (x_p, t_q) the value of input vector and output are:

$$\text{output} \implies \frac{\partial u}{\partial t}(x_p, t_q)$$

$$\text{input} \implies \left(u(x_p, t_q), \frac{\partial u}{\partial x}(x_p, t_q), \frac{\partial^2 u}{\partial x^2}_{(x_p,t_q)}, u(x_p, t_q)\frac{\partial u}{\partial x}_{(x_p,t_q)}, u(x_p, t_q)\frac{\partial u}{\partial t}_{(x_p,t_q)}, \right.$$
$$\left. u(x_p, t_q)\frac{\partial^2 u}{\partial x^2}_{(x_p,t_q)}, u^2(x_p, t_q), u^2(x_p, t_q)\frac{\partial u}{\partial x}_{(x_p,t_q)}, u^2(x_p, t_q)\frac{\partial^2 u}{\partial x^2}_{(x_p,t_q)} \right)$$

Since there are N grid points in x dimension and M grid points in t dimension, the number of different input and output values are $M \times N$.

5.2.2 Burgers' equation with noisy data

A Guassian noise of comparable magnitude is added to each raw data point obtained using previous procedure, which generates velocity u values with noise. This data has spikes everywhere and calculating derivatives for such data will lead to errors of very high magnitude. Thus the data must be cleaned and smoothened as much as possible. The smoothened data still contains small perturbances hence calculating the partial derivatives using simple central difference techniques will give considerable errors. In such a case a new method must be used to calculate those [13].
Here also we assume there are N grid points in x dimension and M grid points in t dimension. The following procedure is used to smoothen the data.

- The Savitzky Golay filter is used to smoothen the raw data.
- 17 neighbouring points are used to generate a order 3 polynomial for each data point.
- The value of this polynomial at that point is stored in a vector of N dimension that approximates the original data **all x values** for **each t value**.
- This process generates M vectors one for each t value.
- The same procedure is repeated again for each of this vector to smoothen it further.
- Finally there M vectors of smoothened values each containing N entries. This forms a smooth-u matrix of $M \times N$ dimension.
- The smooth-u are used as smoothened values.
- This serves as a cleaner version of the original data on which further calculations are carried out.

The figure below shows the smoothened u values after following the above mentioned procedure.

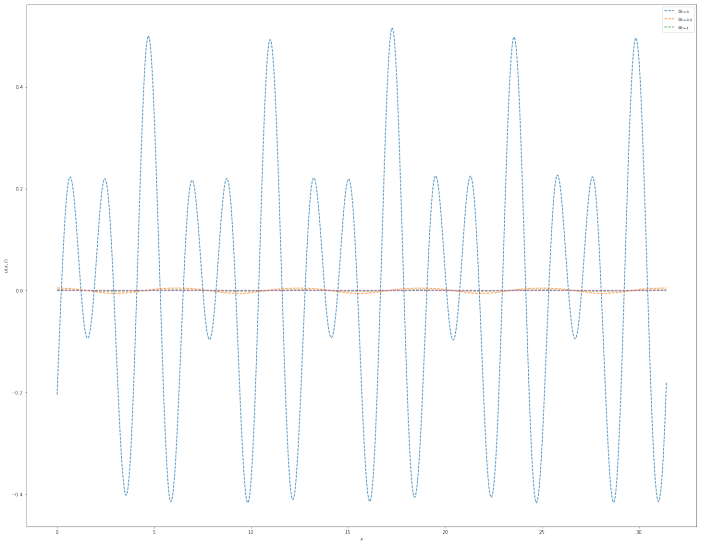

Figure 16: Smoothened Burger's training data

Now a new procedure is used to calculate the partial derivative values and populate the X matrix [11, 15, 8, 10]. It is as described below.

- Polynomial interpolation is used to interpolate the function that describes a certain set/ range of data.
- To calculate the partial derivatives with respect to x the following steps are taken and repeated for each t value to calculate the derivatives for all points:
    1. A window of 11 points is taken to calculate the derivatives for the point at the center of the window.
    2. Using those 11 points, a polynomial of degree 5 is fitted to the points in the window.
    3. The polynomial ($p(x)$) is constructed using **Legendre Polynomials** as the orthogonal polynomial basis.
    4. The derivatives of this polynomial ($p(x)$) are calculated.
    5. These derivatives $(\frac{dp}{dx})_{x_j}$, $(\frac{d^2p}{dx^2})_{x_j}$ serves as $\frac{\partial u}{\partial x}$, $\frac{\partial^2 u}{\partial x^2}$ respectively at all points (x_j, t_i), and are calculated keeping the value of t constant.
    6. The value of derivatives of $p(x)$ for the center point of the window serves as the partial derivative values that used in matrix X.
    7. The window is then shifted one point forward and the same procedure is repeated to calculate the partial derivative values for that point.
    8. This way the window is moved till all the points in x direction for that particular t are covered.
    9. This gives partial derivative values for all points in x direction except the first and last 5 points.
    10. The above procedure is repeated for all the t values.
- The value of derivatives for all the points of the grid except 5 boundary points in each direction are thus calculated and used to populate the matrix X.

- A similar procedure is followed for calculating $\frac{\partial u}{\partial t}$ values at all grid points.
- In that case a polynomial $q(t)$ of degree 3, taking into consideration 11 point - sized window, is calculated for all values of t for a given x value. And the same procedure is repeated for all x grid points.
- Thus $(\frac{dq}{dt})_{t_i}$ approximates the value of $\frac{\partial u}{\partial t}$ at point (x_j, t_i) and is used as to populate the vector Y.

The partial derivatives calculated in this fashion are much better approximations of the actual values but they still contains small spikes and irregularities due to the error in interpolation. To counter this, the partial derivatives are smoothened further. To do this the Savitzky Golay filter is used again. Each of the partial derivatives are smoothened using the filer. The procedure to smmothen them is mentioned below.

- $\frac{\partial u}{\partial x}$ is smoothened by applying the filter twice, first on the original data and then on the filter smoothened data. In both the cases, 57 points are used and a degree 3 polynomial is fitted.
- $\frac{\partial u}{\partial t}$ is also smoothened by applying the filter twice in the same fashion as $\frac{\partial u}{\partial x}$. But here, in both the cases, 157 points are used and a degree 3 polynomial is fitted.
- $\frac{\partial^2 u}{\partial x^2}$ is also smoothened twice using different set of points. But in this case the set of points considered while smoothening differs with the value of t grid location. The polynomial remains of degree 3.
- The number of points to be included and the degree of polynomial are selected experimentally.

Finally these smoothened values of derivatives and the velocity u are used as input and output data used for training. X and Y matrices have this smoothened values.

The input and output features remains the same as used in Burgers' equation for clean data.

5.3 Model Implementation

The problem is finding the value of Θ which best fits the model $Y = X\Theta$. A simple approach is to use simple multi-linear regression which solves this optimisation problem. But it has a problem. Multi-linear regression cannot be used for sparse regression. It assigns some weights to all the features and no feature is assigned a zero weight. Zero weight to a feature indicates that the feature is not at all important for prediction and hence can be neglected from the final equation. Thus zero weights to features promotes sparsity as it indicates that only a few of the features are relevant for the problem.

PDE's are equations that model physical phenomenon. They are interpretable i.e. they have very few terms which has some physical meaning. Thus normal regression cannot be used as it gives a lengthy equation with all the terms as mentioned in matrix/ library X.

Thus LASSO which is a sparse regression technique is used. The value of Θ obtained from this has many zeroes. This means importance to only a few terms is given for prediction and thus gives an interpretable PDE.

5.3.1 One Dimensional Burgers' Equation

LASSO algorithm is used for training the model from the data generated as shown earlier. The output values and the input values are calculated using normal numerical differentiation formula. The optimum value of weights Θ depends on the value of hyper-parameter λ.
The following are the parameters that used while training the model:

- Maximum iterations = 10^9
- Tolerance = 10^{-11}

To find the optimum value of λ, the following procedure is followed:

- Different values of λ over a certain range of maximum and minimum values are selected.
- For all the different values of λ, the model is trained.
- Number of non-zero θ_s and the mean square error(MSE) is calculated for each of those λ values.
- The λ value with the least MSE and the optimum number of non - zero θ_s is selected.

For different values of λ the following data is calculated:

λ	**MSE**	**Non-zero θ_s**
10	0.009935	0
5	0.009935	0
1	0.009935	0
0.5	0.009935	0
0.1	0.009935	0
0.05	0.009935	0
0.01	0.005772	1
0.005	0.004166	2
0.001	0.000216	3
0.0005	5.427e-05	3
0.0001	2.227e-06	3
0.00005	6.015e-07	3
0.00001	6.677e-08	3
0.000005	4.357e-08	4
0.000001	1.620e-08	5

Table 1: Mean square error and Number of non zero θ values for each λ for clean Burgers' equation data

It is clear from the table that $\boxed{\lambda = 0.00001}$ gives optimum result.

5.3.2 Burgers' equation for noisy data

LASSO algorithm is used for training the model from the data generated as shown earlier. The output values and the input values are calculated using the procedures mentioned earlier. The optimum value of weights Θ depends on the value of hyper-parameter λ.
The following are the parameters that used while training the model:

- Maximum iterations = 10^{10}
- Tolerance = 10^{-12}

To find the optimum value of λ, the same procedure followed for clean data is followed. The following table depicts the value of mean squared error (MSE) and non-zero weights (θ_i) for different values of λ.

λ	**MSE**	**Non-zero θ_s**
10	0.0004447	0
5	0.0004447	0
1	0.00044475	0
0.5	0.0004447	0
0.1	0.0004447	0
0.05	0.0004447	0
0.01	0.0004447	0
0.005	0.0004447	0
0.001	8.978e-05	1
0.0005	6.937e-05	1
0.0001	6.283e-05	1
0.00005	3.036e-05	2
0.00001	9.466e-06	2
0.000005	8.593e-06	4
0.000001	8.040e-06	6
0.0000005	1.620e-06	6
0.0000001	8.015e-06	8
0.00000005	7.950e-06	8
0.00000001	7.933e-06	9

Table 2: Mean square error and Number of non zero θ values for each λ for noisy Burgers' equation data

It is clear from the table that $\boxed{\lambda = 0.000005}$ gives optimum result.

5.4 Results

5.4.1 Burgers' equation for clean data

For this value of λ the coefficient vector Θ is :

$$\boxed{\Theta = (0, \quad 0, \quad 0.2004, \quad 0, \quad -0.9998, \quad 0, \quad 0, \quad 0, \quad 0, \quad -0.0009)}$$

Substituting the value of Θ in the prediction equation $Y = X\Theta$. The PDE's reconstructed is :

$$\boxed{\frac{\partial u}{\partial t} = -0.9998\, u \frac{\partial u}{\partial x} + 0.2004\, \frac{\partial^2 u}{\partial x^2} + -0.0009\, u^2 \frac{\partial^2 u}{\partial x^2}}$$

It is similar to the original 1D Burgers' equation with slight change in coefficients, the value of ν is predicted to be 0.2004 which is close to the actual value of 0.2 and an extra $u^2 \frac{\partial^2 u}{\partial x^2}$ term is introduced with a very small coefficient.

The numerical scheme to solve this equation is similar to the original scheme with small modifications. The new scheme is :

$$u(x_j, t+\Delta t) = u(x_j, t) + \Delta t \left((0.2004 + 0.0009u^2) \frac{u(x_j+\Delta x, t) - 2u(x_j, t) + u(x_j - \Delta x, t)}{(\Delta x)^2} \right)$$

$$-\Delta t \left(\frac{0.9998}{2} \left(\frac{u(x_{j+\frac{1}{2}}, t)^2 - u(x_{j-\frac{1}{2}}, t)^2}{\Delta x} \right) \right)$$

This numerical scheme can be easily derived from the original scheme used to generate the data.

Both this new equation and the original equation is solved using the following conditions. The initial condition used for solving it is :

$$u(x, 0) = \sin(x)$$

The boundary values and grid size is:

$$x_{left} = 0, x_{right} = 4\pi, \Delta x = 0.05$$

$$t_{start} = 0, t_{end} = 12, \Delta t = 0.005$$

The viscosity is taken to be $\nu = 0.2$.

The actual data and predicted is shown in the given graph:

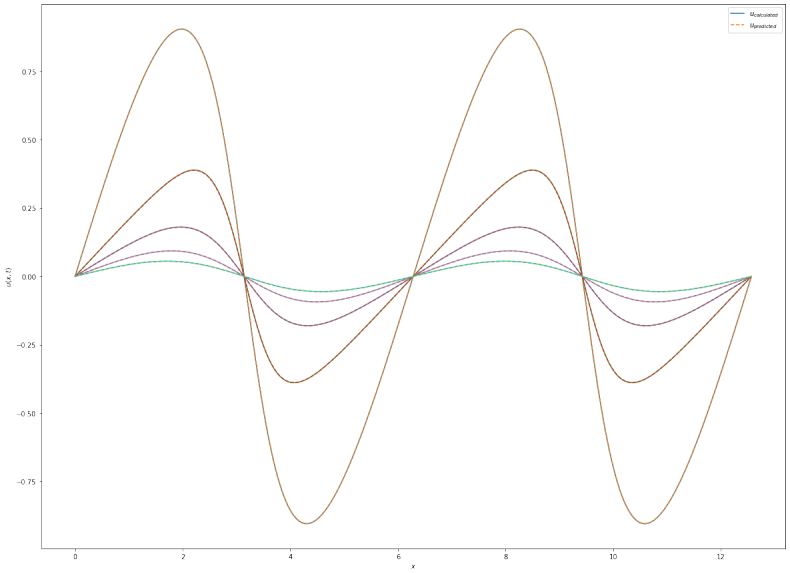

Figure 17: Test results after LASSO regression on clean Burgers' equation data

The solid line represents the actual data and the dotted line represents the data predicted from the new PDE.

5.4.2 Burgers' equation for noisy data

For this value of λ the coefficient vector Θ is :

$$\Theta = (0, \quad 0, \quad 0.1068\ , \quad -0.7876\ , \quad 0\ , \quad 0.0019\ , \quad 0\ , \quad 0\ , \quad -0.0015)$$

Substituting the value of Θ in the prediction equation $Y = X\Theta$. The PDE's reconstructed is :

$$\boxed{\frac{\partial u}{\partial t} = -0.7876\, u\frac{\partial u}{\partial x} + 0.1068\,\frac{\partial^2 u}{\partial x^2} + 0.0019\, u\frac{\partial^2 u}{\partial x^2} - 0.0015\, u^2\frac{\partial^2 u}{\partial x^2}}$$

This numerical scheme can be easily derived from the original scheme used to generate the data. The value of ν is predicted to be 0.1068 which is close to the actual value of 0.1 and extra $u\frac{\partial^2 u}{\partial x^2}$ and $u^2\frac{\partial^2 u}{\partial x^2}$ term is introduced with a very small coefficients.

The numerical scheme to solve this equation is similar to the original scheme with small modifications. The new scheme is :

$$u(x_j, t+\Delta t) = u(x_j, t) + \Delta t\left((0.1068 + 0.0019u - 0.0015u^2)\frac{u(x_j+\Delta x, t) - 2u(x_j, t) + u(x_j - \Delta x, t)}{(\Delta x)^2}\right)$$

$$-\Delta t\left(\frac{0.7876}{2}\left(\frac{u(x_{j+\frac{1}{2}}, t)^2 - u(x_{j-\frac{1}{2}}, t)^2}{\Delta x}\right)\right)$$

Both this new equation and the original equation is solved using the following conditions. The initial condition used for solving it is :

$$u(x, 0) = 2\,\cos(x)$$

The boundary values and grid size is:

$$x_{left} = 0, x_{right} = 4\pi, \Delta x = 0.05$$

$$t_{start} = 0, t_{end} = 12, \Delta t = 0.005$$

The viscosity is taken to be $\nu = 0.1$.

The actual data and predicted is shown in the below given graph:

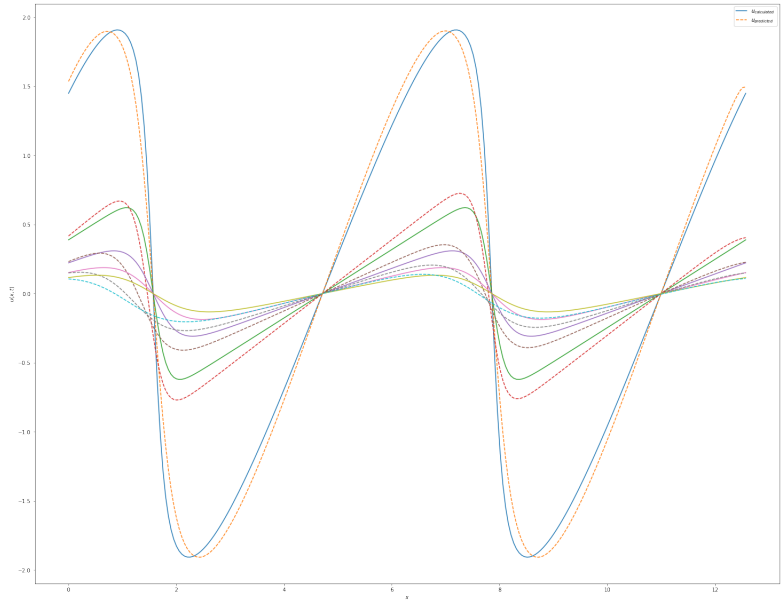

Figure 18: Test results after LASSO regression on noisy Burgers' equation data

The solid line represents the actual data and the dotted line represents the data predicted from the new PDE.

6 Conclusion

The thesis aims at developing methods and models that could be used to forecast the behaviour of dynamical systems, i.e. systems that shows change in its characteristics over time. Fluid systems are the best examples of dynamical systems and thus an attempt is made to predict its course using machine learning models. For simpler equations such as advection and inviscid Burgers' equation, a simple neural network model with appropriate features can be used to predict the velocity of fluids confirming those equations. These were very simple models used for simpler systems.

The main goal of developing prediction models from the complex data of the system could be realised by uncovering the underlying differential equation that governs the system behaviour. Thus a framework is developed to accomplish this task.The partial derivatives with respect to spatial and temporal dimensions, the powers of system variables are used as input and outputs of the model. Thus the model predicts rate of temporal change of the system variables as a function other partial derivatives and power terms. Once the model is trained, it predicts the differential equation which governs the data ,i.e. system variables. This equation can then be used to predict of behaviour of the system for future time stamps and in different conditions. Sparse regression model is used to generate differential equations which are sparse and thus interpretable. Most of the differential equations governing natural systems have inherently very few terms that have physical significance. Thus a sparse regression technique has to be employed such that the differential equations generated have as fewer terms as possible.

A cleaner velocity data is first used to calculate the input and output features that are used to train and test the model for predict the Burgers' equation which the system follows. The equation thus reconstructed out of the data is very similar to the Burgers equation but has one extra partial derivative term with a very small coefficient. This depicts either the data is insufficient and the model training is slightly inaccurate to achieve the exact same equation or there is a presence of certain terms indicates that those terms might be latently present at certain levels.

The model is then implemented for a noisy and corrupted data. The data is filtered and smoothened using Saviltzky Golay filter and the partial derivatives are calculated using polynomial interpolation through Legendre polynomials. These derivatives are further smoothened using the Saviltzky Golay filter and are then used as input and output features. The sparse LASSO regression model is trained using this data and the optimisation results in the differential equation with two extra terms with small coefficients. The coefficients of the actual terms also vary a bit. These terms arises due to the high frequency noise present in the data, which amplifies the presence of those extra terms. These extra terms reasons from the combination of noise and inaccurate model prediction. Thus noisy real time data follows slightly different governing partial differential equation than cleaner ideal data. It can be concluded that a near to ideal governing equation can be recontructed from the data available.

This work could be further improved by using more advanced techniques for sparse optimisation and noise filtration. Also acquisition real time data can be improved and made more accessible for users worldwide. This type of model can be used in predicting behaviours of all kind of dynamical system ranging from biology to economics and must be used in those domains for further improvements. It can also be extended for evaluating real world systems in other domains, which are inherently complex and difficult to interpret and poses problems during its experimental evaluation; like biological processes or chemical reactions in the body. This method produces the underlying pattern which is interpretable and extendable.

References

[1] Pedro Pablo Cárdenas Alzate. A survey of the implementation of numerical schemes for linear advection equation. *Advances in Pure Mathematics*, 2014, 2014.

[2] Adrian Bejan. *Convection heat transfer*. John wiley & sons, 2013.

[3] Jens Berg and Kaj Nyström. Data-driven discovery of pdes in complex datasets. *Journal of Computational Physics*, 384:239–252, 2019.

[4] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

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

[6] Johannes Martinus Burgers. A mathematical model illustrating the theory of turbulence. *Advances in applied mechanics*, 1:171–199, 1948.

[7] Kathleen Champion, Peng Zheng, Aleksandr Y Aravkin, Steven L Brunton, and J Nathan Kutz. A unified sparse optimization framework to learn parsimonious physics-informed models from data. *IEEE Access*, 8:169259–169271, 2020.

[8] Rick Chartrand. Numerical differentiation of noisy, nonsmooth data. *International Scholarly Research Notices*, 2011, 2011.

[9] Xiangyi Chen, Steven Z Wu, and Mingyi Hong. Understanding gradient clipping in private sgd: A geometric perspective. *Advances in Neural Information Processing Systems*, 33:13773–13782, 2020.

[10] William S Cleveland and Susan J Devlin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American statistical association*, 83(403):596–610, 1988.

[11] Jane Cullum. Numerical differentiation and regularization. *SIAM Journal on numerical analysis*, 8(2):254–265, 1971.

[12] John Heading. *Mathematical methods in science and engineering*. Elsevier Publishing Company, 1970.

[13] JF Kaiser and WA Reed. Data smoothing using low-pass digital filters. *Review of Scientific Instruments*, 48(11):1447–1457, 1977.

[14] Nikhil Ketkar. Stochastic gradient descent. In *Deep learning with Python*, pages 113–132. Springer, 2017.

[15] Ian Knowles and Robert J Renka. Methods for numerical differentiation of noisy data. *Electronic Journal of Differential Equations*, 21:235–246, 2014.

[16] John H Lagergren, John T Nardini, G Michael Lavigne, Erica M Rutter, and Kevin B Flores. Learning partial differential equations for biological transport models from noisy spatio-temporal data. *Proceedings of the Royal Society A*, 476(2234):20190800, 2020.

[17] Mikel Landajuela. Burgers equation. *BCAM Internship-summer*, 2011.

[18] PD Lax. Numerical solution of partial differential equations. *The American Mathematical Monthly*, 72(sup2):74–84, 1965.

[19] Kishan Mehrotra, Chilukuri K Mohan, and Sanjay Ranka. *Elements of artificial neural networks*. MIT press, 1997.

[20] William H Press and Saul A Teukolsky. Savitzky-golay smoothing filters. *Computers in Physics*, 4(6):669–672, 1990.

[21] J Ranstam and JA Cook. Lasso regression. *Journal of British Surgery*, 105(10):1348–1348, 2018.

[22] Michael Renardy and Robert C Rogers. *An introduction to partial differential equations*, volume 13. Springer Science & Business Media, 2006.

[23] Raul Rojas. The backpropagation algorithm. In *Neural networks*, pages 149–182. Springer, 1996.

[24] Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science advances*, 3(4):e1602614, 2017.

[25] Hayden Schaeffer. Learning partial differential equations via data discovery and sparse optimization. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2197):20160446, 2017.

[26] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017.

[27] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[28] H Wang, Z Lei, X Zhang, B Zhou, and J Peng. Machine learning basics. *Deep learning*, pages 98–164, 2016.

[29] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[30] Hao Xu, Haibin Chang, and Dongxiao Zhang. Dl-pde: Deep-learning based data-driven discovery of partial differential equations from discrete and noisy data. *arXiv preprint arXiv:1908.04463*, 2019.