Data-Driven Discovery of Partial Differential Equations

Zihan Zhou

Undergraduate Research Support Scheme
University of Warwick

Partial differential equations (PDEs) are used to describe a physical or biological system. Knowledge of PDEs provide a basis for obtaining analytical solutions or conducting numerical simulation for the system.

Partial Differential Equations - Governing a System

Partial differential equations (PDEs) are used to describe a physical or biological system. Knowledge of PDEs provide a basis for obtaining analytical solutions or conducting numerical simulation for the system.

Examples:

- The **Navier-Stokes equations** in fluid dynamics:

$$\rho(\mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u}) = -\nabla p + \rho \mathbf{F} + \mu \nabla^2 \mathbf{u}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0. \tag{2}$$

Partial Differential Equations - Governing a System

Partial differential equations (PDEs) are used to describe a physical or biological system. Knowledge of PDEs provide a basis for obtaining analytical solutions or conducting numerical simulation for the system.
Examples:

- The **Navier-Stokes equations** in fluid dynamics:

$$\rho(\mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u}) = -\nabla p + \rho \mathbf{F} + \mu \nabla^2 \mathbf{u}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0. \tag{2}$$

- The **wave equation**, can be adapted to describe seismic wave propagation

$$u_{tt} = c^2(u_{xx} + u_{yy}) \tag{3}$$

Partial differential equations (PDEs) are used to describe a physical or biological system. Knowledge of PDEs provide a basis for obtaining analytical solutions or conducting numerical simulation for the system.
Examples:

- The **Navier-Stokes equations** in fluid dynamics:

$$\rho(\mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u}) = -\nabla p + \rho \mathbf{F} + \mu \nabla^2 \mathbf{u}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0. \tag{2}$$

- The **wave equation**, can be adapted to describe seismic wave propagation

$$u_{tt} = c^2(u_{xx} + u_{yy}) \tag{3}$$

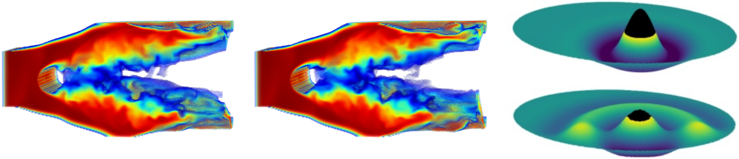


Figure: Left: the 3D Navier-Stokes Equation. Right: the 2D wave equation

How are PDEs obtained?

How are PDEs obtained?

- Traditionally, PDEs are deducted using physical principles: mass/momentum balance, Newton's law, ... etc.

How are PDEs obtained?

- Traditionally, PDEs are deducted using physical principles: mass/momentum balance, Newton's law, ... etc.
- The novel data-driven method: using machine learning to discover PDEs directly from experiment or simulation data. Proposed in the paper *"Data-Driven Identification of Parametric Partial Differential Equations"* (Rudy et al. 2019).

How are PDEs obtained?

- Traditionally, PDEs are deducted using physical principles: mass/momentum balance, Newton's law, ... etc.
- The novel data-driven method: using machine learning to discover PDEs directly from experiment or simulation data. Proposed in the paper *"Data-Driven Identification of Parametric Partial Differential Equations"* (Rudy et al. 2019).

Data-driven PDE discovery

- Suppose we have a dataset for a system regarding a certain quantity u, but the governing PDE for u is unknown. Assume the PDE is of the generic form

$$u_t = F(u, u_x, u_{xx}, \dots, x, \mu), \tag{4}$$

where $F(\cdot)$ **is an unknown, usually nonlinear function of** $u(x,t)$ **and its derivatives, parameterized by** μ. Goal: to discover F from the data.

How are PDEs obtained?

- Traditionally, PDEs are deducted using physical principles: mass/momentum balance, Newton's law, ... etc.
- The novel data-driven method: using machine learning to discover PDEs directly from experiment or simulation data. Proposed in the paper *"Data-Driven Identification of Parametric Partial Differential Equations"* (Rudy et al. 2019).

Data-driven PDE discovery

- Suppose we have a dataset for a system regarding a certain quantity u, but the governing PDE for u is unknown. Assume the PDE is of the generic form

$$u_t = F(u, u_x, u_{xx}, \ldots, x, \mu), \tag{4}$$

where $F(\cdot)$ **is an unknown, usually nonlinear function of** $u(x,t)$ **and its derivatives, parameterized by** μ. Goal: to discover F from the data.

- **Input data** $\mathbf{U} \in \mathbb{C}^{mn}$ is **a matrix of values of** u collected at m time steps with time interval Δt and n spatial locations with space interval Δx given by

$$\mathbf{U} = \begin{bmatrix} u(0,0) & \ldots & u(n\Delta x, 0) \\ \vdots & & \vdots \\ u(0, m\Delta t) & \ldots & u(n\Delta x, m\Delta t) \end{bmatrix}. \tag{5}$$

$\mathbf{U}$ is flattened as a vector $[u(0,0) \ldots u(n\Delta x, 0) \ldots u(0, m\Delta t) \ldots u(n\Delta x, m\Delta t)]$ of length mn to serve as the input to our the machine learning algorithm.

Discretizing Target PDE and Learning Coefficients

Applying numerical methods (finite difference methods, or polynomial interpolation) to input data $\mathbf{U}$ yields **a dataset $\mathbf{U}_t \in \mathbb{R}^{mn}$ of time derivative** u_t. The discrete form of the target PDE $u_t = F(u, u_x, u_{xx}, \ldots, x, \mu)$ can thus be written as

$$\mathbf{U}_t = [u_t(0,0) \ldots u_t(n\Delta x, m\Delta t)]^T = \mathbf{\Theta}(\mathbf{U}, \mathbf{Q})\xi. \tag{6}$$

Here $\mathbf{Q} \in \mathbb{C}^{mn}$ denotes **additional input**, and $\mathbf{\Theta}(\mathbf{U}, \mathbf{Q}) \in \mathbb{C}^{mn \times D}$ is **a library of D candidate terms** and takes the form

$$\mathbf{\Theta}(\mathbf{U}, \mathbf{Q}) = [\mathbf{1} \quad \mathbf{U} \quad \mathbf{U^2} \ldots \quad \mathbf{Q} \quad \ldots \quad \mathbf{U}_x \quad \mathbf{U}\mathbf{U}_x \quad \ldots]. \tag{7}$$

$\xi \in \mathbb{C}^D$ **is the coefficient vector** containing coefficients of each of the candidate terms. We apply machine learning methods to obtain the correct coefficient vector ξ.

Discretizing Target PDE and Learning Coefficients

Applying numerical methods (finite difference methods, or polynomial interpolation) to input data $\mathbf{U}$ yields **a dataset $\mathbf{U}_t \in \mathbb{R}^{mn}$ of time derivative** u_t. The discrete form of the target PDE $u_t = F(u, u_x, u_{xx}, \ldots, x, \mu)$ can thus be written as

$$\mathbf{U}_t = [u_t(0,0) \ldots u_t(n\Delta x, m\Delta t)]^T = \mathbf{\Theta}(\mathbf{U}, \mathbf{Q})\xi. \tag{6}$$

Here $\mathbf{Q} \in \mathbb{C}^{mn}$ denotes **additional input**, and $\mathbf{\Theta}(\mathbf{U}, \mathbf{Q}) \in \mathbb{C}^{mn \times D}$ is **a library of D candidate terms** and takes the form

$$\mathbf{\Theta}(\mathbf{U}, \mathbf{Q}) = [\mathbf{1} \quad \mathbf{U} \quad \mathbf{U}^2 \ldots \quad \mathbf{Q} \quad \ldots \quad \mathbf{U}_x \quad \mathbf{U}\mathbf{U}_x \quad \ldots]. \tag{7}$$

$\xi \in \mathbb{C}^D$ **is the coefficient vector** containing coefficients of each of the candidate terms. We apply machine learning methods to obtain the correct coefficient vector ξ.

Example (Diffusion Equation)

Suppose the data $\mathbf{U}$ is governed by the 1D heat equation $u_t = u_{xx}$, then we may build a library of $D = 6$ candidate terms

$$\mathbf{\Theta}(\mathbf{U}, \mathbf{Q}) = [\mathbf{1} \quad \mathbf{U} \quad \mathbf{U}^2 \quad \mathbf{U}_x \quad \mathbf{U}_{xx} \quad \mathbf{U}\mathbf{U}_x],$$

and if our method is successful, we will get a coefficient vector $\xi \in \mathbb{C}^{D=6}$ that takes the value

$$\xi = [0\ 0\ 0\ 0\ 1\ 0]^T.$$

Sparse Regression: Ridge, LASSO, and Elastic Net

Our goal is to find the optimal coefficient vector $\xi \in \mathbb{C}^D$ satisfying

- $\mathbf{U}_t \approx \mathbf{\Theta}(\mathbf{U}, \mathbf{Q})\xi$,
- ξ is sparse, so that only the most relevant terms are included.

To achieve this, it is natural to consider applying the sparse regression models

Sparse Regression: Ridge, LASSO, and Elastic Net

Our goal is to find the optimal coefficient vector $\xi \in \mathbb{C}^D$ satisfying

- $\mathbf{U}_t \approx \mathbf{\Theta}(\mathbf{U}, \mathbf{Q})\xi$,
- ξ is sparse, so that only the most relevant terms are included.

To achieve this, it is natural to consider applying the sparse regression models

Three Types of Sparse Regression

- **Ridge Regression**: aims to find $\hat{\xi}$ that minimizes the loss function

$$\hat{\xi} = \text{argmin}_{\xi} L(\xi), \text{where } L(\xi) = \|\mathbf{\Theta}\xi - \mathbf{U}_t\|_2^2 + \lambda\|\xi\|_2, \tag{8}$$

  where $\|\xi\|_2 = \sqrt{\xi_1^2 + \cdots + \xi_D^2}$ is the L_2 norm and λ a constant coefficient

- **LASSO Regression**: similar structure, but uses L_1 norm instead of the L_2 norm

$$L(\xi) = \|\mathbf{\Theta}\xi - \mathbf{U}_t\|_2^2 + \alpha\|\xi\|_1, \tag{9}$$

  where $\|\xi\|_1 = |\xi_1|_+ \cdots + |\xi_D|$ is the L_1 norm.

- **Elastic Net Regression** simply combines the two regularization terms above:

$$L(\xi) = \|\mathbf{\Theta}\xi - \mathbf{U}_t\|_2^2 + \lambda\|\xi\|_2 + \alpha\|\xi\|_1. \tag{10}$$

Minimizing $\|\mathbf{\Theta}\xi - \mathbf{U}_t\|_2^2$ fits ξ to the data $\mathbf{U_t}$, and minimizing $\|\xi\|_1$ and/or $\|\xi\|_2$ shrinks coefficients of irrelevant terms.

However, PDE discovery requires an output where only relevant terms are nonzero.

Sequential Threshold Ridge Regression

However, PDE discovery requires an output where only relevant terms are nonzero. The optimal $\hat{\xi}$ should minimize the loss function

$$\hat{\xi} = \text{argmin}_{\xi} L(\xi), \text{where } L(\xi) = \|\boldsymbol{\Theta}(\mathbf{U}, \mathbf{Q})\xi - \mathbf{U}_t\|_2^2 + \epsilon\kappa(\boldsymbol{\Theta}(\mathbf{U}, \mathbf{Q}))\|\xi\|_0. \tag{11}$$

$\|\cdot\|_0$ is the L_0 **norm that counts the number of nonzero elements** in ξ, $\kappa(\boldsymbol{\Theta}(\mathbf{U}, \mathbf{Q}))$ is the **condition number** of the matrix $\boldsymbol{\Theta}$.

However, PDE discovery requires an output where only relevant terms are nonzero. The optimal $\hat{\xi}$ should minimize the loss function

$$\hat{\xi} = \text{argmin}_{\xi} L(\xi), \text{where } L(\xi) = \|\boldsymbol{\Theta}(\mathbf{U}, \mathbf{Q})\xi - \mathbf{U}_t\|_2^2 + \epsilon\kappa(\boldsymbol{\Theta}(\mathbf{U}, \mathbf{Q}))\|\xi\|_0. \tag{11}$$

$\|\cdot\|_0$ is the L_0 **norm that counts the number of nonzero elements** in ξ, $\kappa(\boldsymbol{\Theta}(\mathbf{U}, \mathbf{Q}))$ is the **condition number** of the matrix $\boldsymbol{\Theta}$.
To get the above $\hat{\xi}$, sparse regression is applied with sequential thresholding:

Algorithm 3: Sequential Threshold Ridge Regression:STRidge($\boldsymbol{\Theta}, \mathbf{U}_t, \lambda, tol$, iters, num_big)

$\hat{\xi} \leftarrow \text{argmin}_{\xi}(\|\boldsymbol{\Theta}\xi - \mathbf{U}_t\|_2^2 + \lambda\|\xi\|_2)$ `#standard ridge regression`

biginds $\leftarrow \{j : |\hat{\xi}_j| \geq tol\}$ `#select indices of large coefficients`

`#Check if number of big coefficients changed from last time`

if *num_big != len(biginds)* **then**

  num_big $\leftarrow$ len(biginds) `#Updates number of big coefficients`

  $\hat{\xi}[\sim \text{biginds}] \leftarrow 0$ `#set small coefficients to 0`

  $\hat{\xi}[\text{biginds}] \leftarrow \text{STRidge}(\boldsymbol{\Theta}[:, \text{biginds}], \mathbf{U}_t, \lambda, tol, \text{iters} - 1, \text{num\_big})$

  `#recursive call with fewer coefficients`

else

  return $\hat{\xi}$

end

Generating Simulation Data with Finite Difference Method

Let $u_t(x,t) = F(u, u_x, \dots), x \in [0, L], t \in [0, T]$ be a PDE, with initial condition $u(x,0) = f(x)$, boundary conditions $u(0,t) = g(t)$, $u(L,t) = h(t)$. A simulation dataset for this PDE can be generated using **finite difference methods (FDM)**.

Finite Difference Methods: FTCS and Crank-Nicholson

- **Forward difference in time, central difference for space (FTCS):**

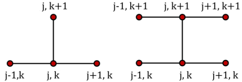
$$U_t \approx \frac{U_j^{k+1} - U_j^k}{\Delta t},\ U_x = \frac{U_{j+1}^k - U_{j-1}^k}{2\Delta x},\ U_{xx} = \frac{U_{j+1}^k - 2U_j^k + U_{j+1}^k}{2(\Delta x)^2} \tag{12}$$

- **Crank-Nicholson method,** which uses the same formulation of U_t and U_x but

$$UU_x = \frac{1}{2}\left(U_j^k \frac{U_{j+1}^{k+1} - U_{j-1}^k}{2\Delta x} + U_j^{k+1} \frac{U_{j+1}^k - u_{j-1}^k}{2\Delta x} \right) \tag{13}$$

$$U_{xx} \approx \frac{1}{2}\left(\frac{U_{j+1}^{k+1} - 2U_j^{k+1} + U_{j-1}^{K+1}}{2\Delta x} + U_j^{k+1} \frac{U_{j+1}^k - U_{j-1}^k}{2\Delta x} \right) \tag{14}$$

$$\mathbf{U} = \begin{bmatrix} u(0,0) & \dots & u(n\Delta x, 0) \\ \vdots & & \vdots \\ u(0, m\Delta t) & \dots & u(n\Delta x, m\Delta t) \end{bmatrix} = \begin{bmatrix} U_0^0 & \dots & U_n^0 \\ \vdots & & \vdots \\ U_0^m & \dots & U_n^m, \end{bmatrix}$$



Testing results: Linear Advection Equation

We now generate data set with FDM and test the accuracy of the regression algorithm.

The advection equation

The **advection equation** is a hyperbolic equation that represents how a scalar quantity u (such as temperature, concentration, etc.) is transported by a fluid moving with a constant velocity c in a single spatial dimension. For our example we set $c = 1$ and use **Gaussian initial condition** and **periodic boundary condition**.

$$u_t(x,t) = -cu_x(x,t), \quad x \in [-L/2, L/2], t \in [0, T] \tag{15}$$

$$u(x,0) = \exp(-x^2) \quad , \tag{16}$$

$$u(-L/2,t) = u(L/2,t) \quad . \tag{17}$$

Testing results: Linear Advection Equation

We now generate data set with FDM and test the accuracy of the regression algorithm.

The advection equation

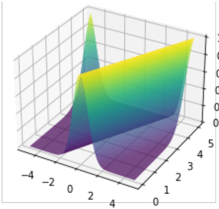
The **advection equation** is a hyperbolic equation that represents how a scalar quantity u (such as temperature, concentration, etc.) is transported by a fluid moving with a constant velocity c in a single spatial dimension. For our example we set $c = 1$ and use **Gaussian initial condition** and **periodic boundary condition**.

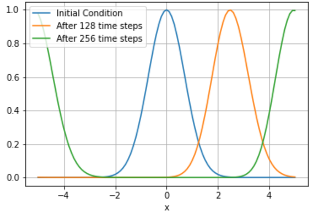
$$u_t(x,t) = -cu_x(x,t), \quad x \in [-L/2, L/2], t \in [0,T] \tag{15}$$

$$u(x,0) = \exp(-x^2) \quad , \tag{16}$$

$$u(-L/2,t) = u(L/2,t) \quad . \tag{17}$$

The simulation results of $u_t = -u_x$ from Crank-Nicholson is given by the figure below.





Testing results: Linear Advection Equation

We now generate data set with FDM and test the accuracy of the regression algorithm.

The advection equation

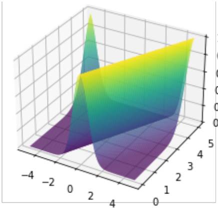
The **advection equation** is a hyperbolic equation that represents how a scalar quantity u (such as temperature, concentration, etc.) is transported by a fluid moving with a constant velocity c in a single spatial dimension. For our example we set $c = 1$ and use **Gaussian initial condition** and **periodic boundary condition**.

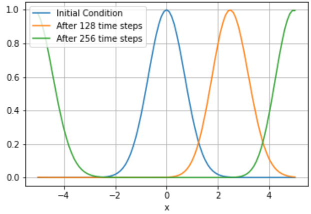
$$u_t(x,t) = -cu_x(x,t), \quad x \in [-L/2, L/2], t \in [0, T] \tag{15}$$

$$u(x,0) = \exp(-x^2) \quad , \tag{16}$$

$$u(-L/2,t) = u(L/2,t) \quad . \tag{17}$$

The simulation results of $u_t = -u_x$ from Crank-Nicholson is given by the figure below.





The PDE discovered by the data-driven method from this dataset is $u_t = -0.999448u_x$.

Testing results: Diffusion Equation

The Diffusion Equation

The **diffusion equation**, also known as **the 1D heat equation**, describes how heat u (or similar diffusive scalar quantity) is distributed over time in a one dimensional system. The heat equation deals with the spreading or dissipation of heat due to diffusion.

$$u_t = du_{xx}, \quad x \in [-L/2, L/2], t \in [0, T]. \tag{18}$$

For our example we set $d = 0.1$ and apply the same Gaussian initial condition and periodic boundary condition.

Testing results: Diffusion Equation

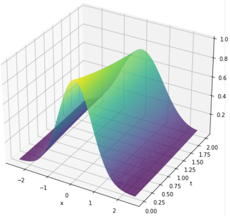
The Diffusion Equation

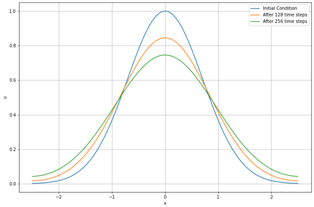
The **diffusion equation**, also known as **the 1D heat equation**, describes how heat u (or similar diffusive scalar quantity) is distributed over time in a one dimensional system. The heat equation deals with the spreading or dissipation of heat due to diffusion.

$$u_t = du_{xx}, \quad x \in [-L/2, L/2], t \in [0, T]. \tag{18}$$

For our example we set $d = 0.1$ and apply the same Gaussian initial condition and periodic boundary condition.

The simulation results of $u_t = 0.1u_{xx}$ from Crank-Nicholson is given by below.





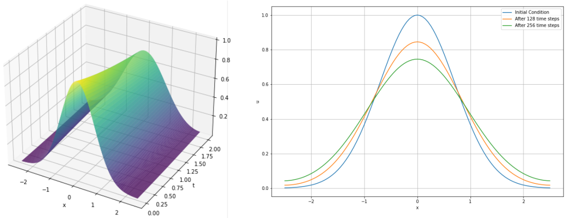
The Diffusion Equation

The **diffusion equation**, also known as **the 1D heat equation**, describes how heat u (or similar diffusive scalar quantity) is distributed over time in a one dimensional system. The heat equation deals with the spreading or dissipation of heat due to diffusion.

$$u_t = du_{xx}, \quad x \in [-L/2, L/2], t \in [0, T]. \tag{18}$$

For our example we set $d = 0.1$ and apply the same Gaussian initial condition and periodic boundary condition.

The simulation results of $u_t = 0.1u_{xx}$ from Crank-Nicholson is given by below.



The PDE discovered by data-driven method from this dataset is $u_t = 0.100003u_{xx}$.

Burger's Equation

Burgers' equation is a fundamental PDE from the field of fluid mechanics and can be used for studying shock waves. The equation is given by

$$u_t = -uu_x + \nu u_{xx}, \quad x \in [-L/2, L/2], t \in [0, T], \tag{19}$$

where u represents speed in a fluid flow context, and ν is viscosity.

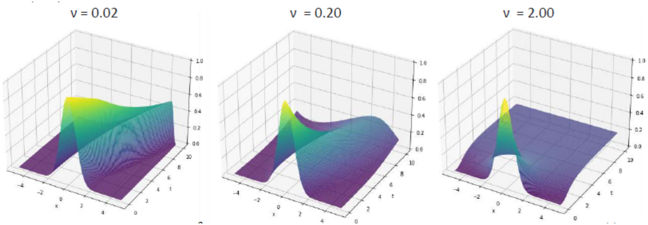
Burger's Equation

Burgers' equation is a fundamental PDE from the field of fluid mechanics and can be used for studying shock waves. The equation is given by

$$u_t = -uu_x + \nu u_{xx}, \quad x \in [-L/2, L/2], t \in [0, T], \tag{19}$$

where u represents speed in a fluid flow context, and ν is viscosity.

The simulation results of $u_t = -uu_x + \nu u_{xx}$ by Crank-Nicholson are given below



ν	Discovered PDE
0.02	$u_t = -1.06uu_x + 0.29u^2u_x - 0.26u^3u_x + 0.02u_{xx}$
0.20	$u_t = -uu_x + 0.20u_{xx}$
2.00	$u_t = 0.04u - 0.30u^2 - 0.46u^3 + \dots$

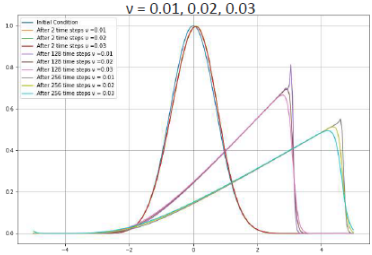
Effect of Smoothness

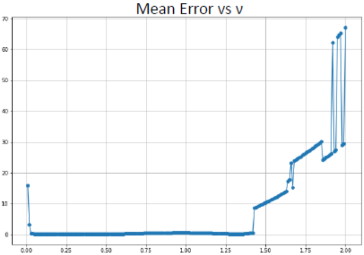
Previous results show that our method appears to only work for a certain range of viscosity ν. We tested the method on $\nu = 0.01, 0.02, \ldots, 2.00$ to investigate the cut-off points.

Effect of Smoothness

Previous results show that our method appears to only work for a certain range of viscosity ν. We tested the method on $\nu = 0.01, 0.02, \ldots, 2.00$ to investigate the cut-off points.

v	PDE	Mean parameter error	Std. of parameter error
0.01	u_t = (0.050140 +0.000000i)u + (-0.163713 +0.000000i)u^2 + (0.100025 +0.000000i)u^3 + ...	16.009310%	14.174193%
0.02	u_t = (-1.057625 +0.000000i)uu_{x} + (0.283156 +0.000000i)u^2u_{x} + ...	3.319493%	2.442960%
0.03	u_t = (-0.995775 +0.000000i)uu_{x} + (0.029581 +0.000000i)u_{xx}	0.420690%	0.001847%
0.04	u_t = (-0.997450 +0.000000i)uu_{x} + (0.039687 +0.000000i)u_{xx}	0.283914%	0.028941%
	...		
1.41	u_t = (-0.995066 +0.000000i)uu_{x} + (1.409464 +0.000000i)u_{xx}	0.514596%	0.021155%
1.42	u_t = (-0.994757 +0.000000i)uu_{x} + (1.419375 +0.000000i)u_{xx}	0.574683%	0.050394%
1.43	u_t = (-0.998046 +0.000000i)uu_{x} + (1.447058 +0.000000i)u_{xx} + (-0.312211 +0.000000i)uu_{xx} + ...	8.626920%	8.431511%
1.44	u_t = (-0.997805 +0.000000i)uu_{x} + (1.457516 +0.000000i)u_{xx} + (-0.320027 +0.000000i)uu_{xx} + ...	8.867872%	8.648358%
	...		
1.99	u_t = (-0.034076 +0.000000i)u + (0.260786 +0.000000i)u^2 + (-0.391764 +0.000000i)u^3 + ...	29.460429%	16.816766%
2.00	u_t = (0.000286 +0.000000i) + (-0.039955 +0.000000i)u + (0.296344 +0.000000i)u^2 +...	67.171925%	20.811909%





- $\nu < 0.02$: a shock is formed, does not identify the correct terms.
- $0.02 \le \nu \le 1.42$: retrieves the original PDE.
- $\nu \ge 1.43$: diffusing too quickly, does not reflect the dynamics, gets inaccurate results.

Effect of Regression Type

The original paper does not test the model with LASSO and Elastic Net. To fill the blank, I modified the STRidge code to develop the algorithms STLasso and STElasticNet for further exploration.

Effect of Regression Type

The original paper does not test the model with LASSO and Elastic Net. To fill the blank, I modified the STRidge code to develop the algorithms STLasso and STElasticNet for further exploration.

ν	Regression Type	Discovered PDE	Runtime (s)	Mean Error (%)	Std Error(%)
2.00	Ridge	$u_t = 2.221u^2u_{xx} - 1.751u^3u_{xx} - 0.009u_{xxx} + 0.060uu_{xxx} \ldots$	1.90	67.2	20.8
2.00	LASSO	$u_t = 2.047u_{xx} - 0.750uu_{xx} + 2.284u^2u_{xx} - 1.792u^3u_{xx} \ldots$	37.88	29.9	17.1
2.00	Elastic Net	$u_t = 2.047u_{xx} - 0.750uu_{xx} + 2.284u^2u_{xx} - 1.792u^3u_{xx} \ldots$	8.61	29.9	17.1
0.20	Ridge	$u_t = -1.00uu_x + 0.200u_{xx}$	**1.2**	0.1	0.1
0.20	LASSO	$u_t = -1.00uu_x + 0.200u_{xx}$	**7.0**	0.1	0.1
0.20	Elastic Net	$u_t = -1.00uu_x + 0.200u_{xx}$	**8.9**	0.1	0.1
0.02	Ridge	$u_t = -0.040u + 0.296u^2 - 0.457u^3 + 0.057u_x - 1.880uu_x + \ldots$	3.0	742.9	2429.8
0.02	LASSO	$u_t = -0.035u + 0.266u^2 - 0.402u^3 - 1.128uu_x + 1.411u^2u_x \ldots$	18.8	701.4	2437.6
0.02	Elastic Net	$u_t = -0.035u + 0.266u^2 - 0.402u^3 - 1.128uu_x + 1.411u^2u_x \ldots$	18.7	701.4	2437.6

Conclusion

- The data-driven method works well when the data is smooth and no shock waves are formed.
- Ridge regression is the most efficient compared to the other two sparse regressions, and accuracy does not seem to differ.
- (Not included in this presentation) Accuracy suffers when a certain level of noise is introduced.

Conclusion

- The data-driven method works well when the data is smooth and no shock waves are formed.
- Ridge regression is the most efficient compared to the other two sparse regressions, and accuracy does not seem to differ.
- (Not included in this presentation) Accuracy suffers when a certain level of noise is introduced.

Related Works

- More on machine learning for PDE-discovery:
  - Brunton et al., *"Discovering governing equations from data by sparse identification of nonlinear dynamical systems"* (2016)
  - Baddoo et al., *"Physics-informed dynamic mode decomposition"* (2023)
- Physics-Informed Neural Networks (PINN) for PDE-discovery:
  - Raissi et al., *"Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations"* (2019)
  - Chen et al., *"Physics-informed learning of governing equations from scarce data"* (2021)
- PDE-discovery for climate science and geophysics:
  - Lai et al., *"Machine learning for climate physics and simulations"* (2024)
  - Cheng and Alkhalifah, *"Robust data driven discovery of a seismic wave equation"* (2024)
  - Zanna and Bolton, *"Data-driven equation discovery of ocean mesoscale closures"*, (2020)

Reference I

- Baddoo, Peter J et al. (2023). "Physics-informed dynamic mode decomposition". In: *Proceedings of the Royal Society A* 479.2271, p. 20220576.
- Brunton, Steven L, Joshua L Proctor, and J Nathan Kutz (2016). "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the national academy of sciences* 113.15, pp. 3932–3937.
- Chen, Zhao, Yang Liu, and Hao Sun (2021). "Physics-informed learning of governing equations from scarce data". In: *Nature communications* 12.1, p. 6136.
- Cheng, Shijun and Tariq Alkhalifah (2024). "Robust data driven discovery of a seismic wave equation". In: *Geophysical Journal International* 236.1, pp. 537–546.
- Lai, Ching-Yao et al. (2024). "Machine learning for climate physics and simulations". In: *Annual Review of Condensed Matter Physics* 16.
- Raissi, M., P. Perdikaris, and G.E. Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378, pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Reference II

Rudy, Samuel et al. (Jan. 2019). "Data-Driven Identification of Parametric Partial Differential Equations". In: *SIAM Journal on Applied Dynamical Systems* 18, pp. 643–660. DOI: 10.1137/18M1191944.

Zanna, Laure and Thomas Bolton (2020). "Data-driven equation discovery of ocean mesoscale closures". In: *Geophysical Research Letters* 47.17, e2020GL088376.