Project 2: Reconstructing PDEs from Data Using PDE-Find

Wu, You
Department of Mathematics
youwuyou@ethz.ch

Theory: Data Driven Discovery of Dynamical Systems and PDEs

In this project, we implemented the **PDE Functional Identification of Nonlinear Dynamics (PDE-Find)** algorithm [1], which discovers the relevant terms of the governing PDEs using sparse regression. It is an extension of the SINDy algorithm [2] in which PDE-Find considers also partial derivatives as part of its feature library. Within the scope of this report, we aim to recover the symbolic expression from the solution u of evolution problems, where the physics is governed by some unknown PDE(s). The spatiotemporal domain is defined as $\tilde{\Omega} := \Omega \times (0, T)$, where T is some final observation time.

Overview

In the following section, we briefly illustrate our implementation of PDE-Find. Our main goal is to assemble a linear system of equations (LSE) from both the PDE solution data and approximation of derivatives of the solution, and then use sparse regression to solve the LSE. We consider a simplified setup, where all solution data is real-valued and the domain $\tilde{\Omega}$ is discretized into equidistant space-time grid of dimension $n \times m$, such that nodal values represents n spatial measurements at m time points. We can now rearrange the solution into a real-valued data matrix $U \in \mathbb{R}^{(n \cdot m) \times 1}$ which contains the discretization of the solution u across spatiotemporal domain. Similarly, the temporal derivative u_t is assembled into a column vector $U_t \in \mathbb{R}^{(n \cdot m) \times 1}$.

$$U_t = \Theta \cdot \xi \tag{1}$$

In equation (1), the first-order temporal derivative u_t of the discretized solution is set as the LHS of the equation. We call the matrix $\Theta \in \mathbb{R}^{(n \cdot m) \times D}$ the **feature library**, where D represents the number of candidate functions in the library. The feature candidates (column vectors of Θ) are discretized solution u and its mixed partial derivatives (u_t, u_x, uu_x ...). With careful selection of feature candidate functions as column vectors in the matrix Θ, we can assume that the matrix is an overcomplete library such that the PDE can be represented by linear and nonlinear-combinations of the feature candidates.[1] Once we obtain the solution $\xi \in \mathbb{R}^{D \times 1}$, we can use its coefficients to recover the dynamic system by:

$$u_t(x, y) := \sum_j \Theta_j \xi_j \tag{2}$$

This way, we can recover the governing equation where each non-zero entry $\xi_j \neq 0, \forall j \in \{1, ..., D\}$ of the solution ξ represents a term in the PDE.

[1]Under the assumption by the task hint, we assume the partial derivatives are only up to and include third order derivatives.

Numerical Evaluation of Derivatives

An essential part in the PDE-Find algorithm lies in computing the partial derivatives. In the original implementation of PDE-Find, simple **finite difference** is used for approximating the spatial and temporal derivatives on clean data. For noisy data, they utilized polynomial interpolation to overcome the additional difficulty.

In project 2, **we initially wanted to train simple feedforward neural networks (FNNs) on all three provided datasets**, and then we can use the trained FNN to approximate the dataset better and then perform automatic differentiation to compute derivatives. However, due to the high training cost, **we only used FNN + automatic differentiation for the first two systems**. **For the third coupled system, we simply perform second-order accurate central differences method** with first order estimates at the boundaries via `torch.gradient` directly on the dataset.

1D Implementation (FNN + automatic differentiation)

For affordable 1D datasets, we trained simple **feed-forward neural networks (FNNs)** to approximate spatiotemporal solutions of the PDE, solely based on the provided `X.npz` dataset. For $X \in \{1, 2\}$, we train a separate FNN as part of the feature library preparation process. For the **network architecture**, we used a simple feedforward neural network (FNN) with three linear layers of customizable width, each linear layer is followed by an nonlinear activation. Even this construct is simple, it allows us to nicely approximate the solution $u(x, t)$ across time, thanks to the universal approximation property of FNN [3]. Once we have successfully trained the FNN and verified its expressiveness by comparing it to the original dataset, we use the **automatic differentiation** utility of the `PyTorch` library, provided by `torch.autograd.grad`, for computing derivatives up to arbitrary order.

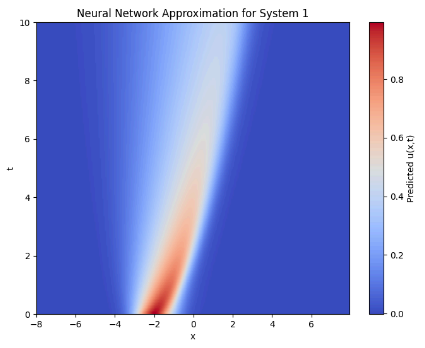


Figure 1: Heatmap of FNN-approximated solution across spatio-temporal grid for **system 1**

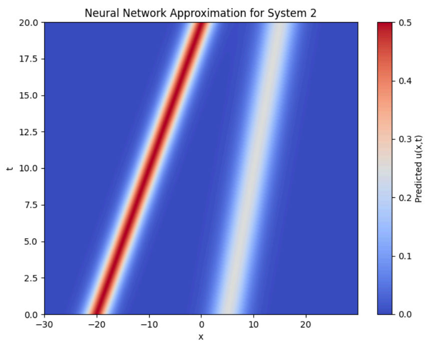


Figure 2: Heatmap of FNN-approximated solution across spatio-temporal grid for **system 2**

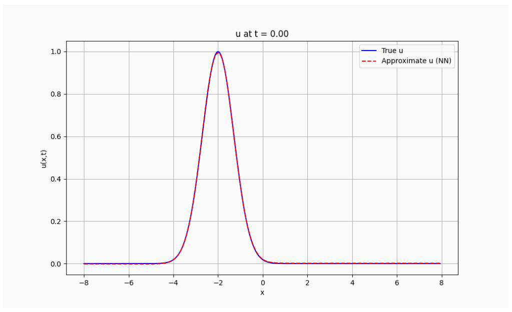


Figure 3: Comparison of ground truth and FNN-approximated solution of **system 1** at $t = 0$

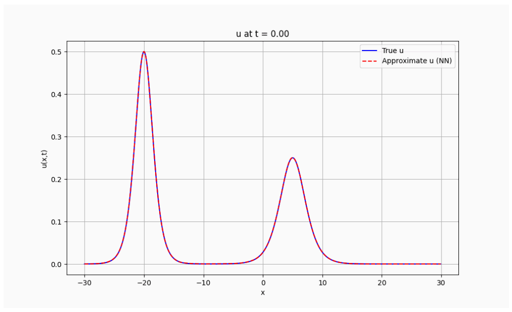


Figure 4: Comparison of ground truth and FNN-approximated solution of **system 2** at $t = 0$

2D Implementation (Direct `torch.gradient`)

For applying PDE-Find on 2D datasets, we did not train a FNN, but performed the `torch.gradient` directly on the dataset as we stated earlier. Under the hood, it performs second-order accurate central differences method with first order estimates at the boundaries.

Building Libraries of Candidate Terms

To build the feature library Θ as in (1), we use the numerical approximation of derivatives as introduced in the previous section. We used automatic differentiation on FNN on 1D data to obtain derivatives, and direct finite difference on 2D data, as already emphasized in the previous section.

Then for each system X for $X \in \{1, 2, 3\}$, we need to make careful decision for selecting suitable candidates accordingly. Since it is give by project hand-out that the highest order of derivatives is 3, we set the upper limit to order of derivatives to 3. In **Appendix A**, we explicitly list out candidates used for each system in our experiments.

An example assembled LSE as shown below (3) is taken from the supplementary material of PDE-Find [4] for some spatio-temporal grid $\tilde{\Omega}$ with n spatial grid points and m time snapshots, and D indicates the number of candidates in the library.

$$\underbrace{\begin{bmatrix} u_t(x_0,t_0) \\ u_t(x_1,t_0) \\ u_t(x_2,t_0) \\ \vdots \\ u_t(x_{n-1},t_m) \\ u_t(x_n,t_m) \end{bmatrix}}_{U_t \in \mathbb{R}^{(n\cdot m)\times 1}} = \underbrace{\begin{bmatrix} 1 & u(x_0,t_0) & u_x(x_0,t_0) & \dots & u^5 u_{xxx}(x_0,t_0) \\ 1 & u(x_1,t_0) & u_x(x_1,t_0) & \dots & u^5 u_{xxx}(x_1,t_0) \\ 1 & u(x_2,t_0) & u_x(x_2,t_0) & \dots & u^5 u_{xxx}(x_2,t_0) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & u(x_{n-1},t_m) & u_x(x_{n-1},t_m) & \dots & u^5 u_{xxx}(x_{n-1},t_m) \\ 1 & u(x_n,t_m) & u_x(x_n,t_m) & \dots & u^5 u_{xxx}(x_n,t_m) \end{bmatrix}}_{\Theta \in \mathbb{R}^{(n\cdot m)\times D}} \underbrace{\begin{bmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \vdots \\ \xi_D \end{bmatrix}}_{\xi \in \mathbb{R}^{D\times 1}} \tag{3}$$

Sparse Regression

Once we have built the candidate library $\Theta \in \mathbb{R}^{(n\cdot m)\times D}$ and computed the temporal derivative of the solution u_t, we can solve for the unknown ξ in the LSE (1). Note that since the resulting Θ **is not** a square matrix, instead, it is a *tall* rectangular system matrix with $(n \cdot m) > D$. For such **overdetermined system, where we have more equations than the unknowns**, we need to seek for the **least squares solution**. Moreover, we are motivated to use **sparse regression** because we know in advance that the solution vector ξ will have sparse coefficients. If we used plain least square methods without regularizations, we would end up in densely populated solution vector ξ, which is not desirable for what we aim for.

Therefore, we implemented the *Algorithm 1. STRidge* with ridge regression. Recall that each ridge regression is to solve a linear least squares problem with l2 regularization, in our context, we aim to minimizes the following objective function:

$$\arg\min_{\xi \in \mathbb{R}^D} \|U_t - \Theta\xi\|_2^2 + \lambda \|\xi\|_2^2 \tag{4}$$

where λ is used to controls the regularization strength and penalize when the found coefficient ξ is too large. In our implementation, we use `sklearn.linear_model.Ridge` with `fit_intercept = False` and set `tol=1e-5` and `max_iter=500`.

In addition to the ridge regression (4), **STRidge recursively performs ridge regression on the augmented LSE** and seek for some $\hat{\xi}$ that minimizes the regularized least squares problem. After each ridge regression solve, we augment the LSE to be solved by applying a predefined hard threshold τ and zero out all coefficients and set $\xi_i = 0$ in the solution vector for those that are larger than the threshold $\xi_i > \tau$. To gradually refine the selection of the best tolerance τ used in algorithm 1, we also implemented the *Algorithm 2. TrainSTRidge*. This algorithm performs the STRidge for a fixed number of time. Among all solves, it finds the optimal predictor ξ_{best} by optimizing the STRidge performance on a selected split of subset that serves as validation from the original dataset. Both algorithm 1 and 2 can be found in the supplementary material of the original PDE-Find paper [1], we also supplement them in **Appendix C** for completeness.

Results: Predict the Governing Equations

We represent results on three different systems, using provided datasets named by `X.npz` for $X \in \{1, 2, 3\}$. For each system we report the discovered equation and the relative L2 error between the LHS and the RHS. The parameters used in sparse regression with `TrainSTRidge` are reported in the associated table for each dataset. For conciseness, we only report the number of candidate terms D used for each system. We refer readers to **Appendix A** to see the full expression of the used candidates, as well as the analysis of each least squares problem based on the condition number of the Θ matrix.

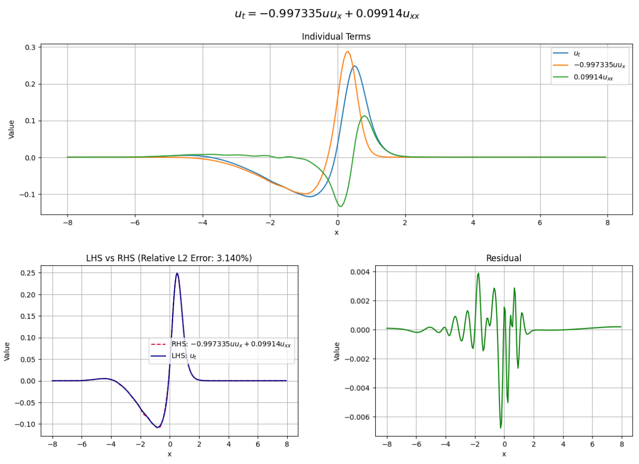
During our implementation, we found the supplementary materials of the original paper [1] to be fruitful and recognized that the provided data may be part of it. Moreover, by visualizing on the original data set, we also could identify the first system to emerge from Burgers' equation, second system to be the Korteweg–De Vries equation (KdV equation) and the third system to depict the reaction-diffusion process. Thus, during our experiments, we did not fully blindly optimize for performance, but followed a Keplerian paradigm, **combining observations and assumptions with our modeling**. We confirm our assumptions on all three systems in the following section.

System 1 (FNN + automatic differentiation)

- *Discovered Equation:*

$$u_t = -0.997335 \cdot uu_x + 0.099140 \cdot u_{xx} \tag{5}$$

- *Relative L2 Error of (5) :* 3.140%



Library Size D	Ridge Penalty λ	STRidge Tolerance	STRidge Iterations
31 Terms	1e-6	5e-3	10
Ridge Iterations	**Train Split**	**Train Iterations**	**Train Penalty η**
500	0.5	50	1e-3

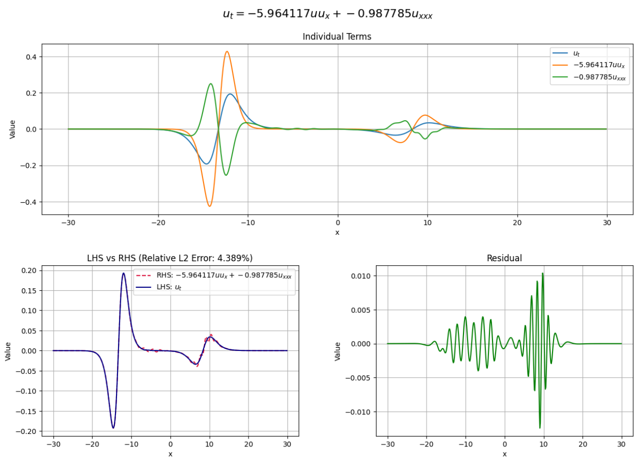
Table 1: Sparse regression parameter for system 1.

System 2 (FNN + automatic differentiation)

- *Discovered Equation:*

$$u_t = -5.964103 \cdot uu_x - 0.987782 \cdot u_{xxx} \tag{6}$$

- *Relative L2 Error of (6) :* 4.389%



Library Size D	Ridge Penalty λ	STRidge Tolerance	STRidge Iterations
13 Terms	1e-6	1e-2	10
Ridge Iterations	**Train Split**	**Train Iterations**	**Train Penalty η**
500	0.7	50	1e-3

Table 2: Sparse regression parameter for system 2.

Success & Failures

As shown above, we were not able to find an approximation of the governing equations, but the relative L2 error between the LHS and RHS is quite low $3-4\%$ for both 1D systems. **This first experiment indicates the success of our implementation.** However, unlike the rather straight forward success for system 1, where we were able to use 31 terms for the Θ library assembly, **we were not able to use more candidates for system 2.**

As shown in Figure 2 and Figure 4, system 2 consists of two non-interacting traveling waves with different amplitudes to the KdV equation. **The similarity of the underlying physics to another system poses significant**

difficulty in symbolic regression, as the nature of the one-way wave equation $u_t + cu_x = 0$ is highly similar with system 2. In the original paper, an approach for disambiguation is proposed in which the least squares problem is solved in a block-wise fashion combining two sets of observations. **We could potentially further improve our method by adding the disambiguation for system 2.**

System 3 (Direct `torch.gradient`)

- *Discovered system:*

$$\begin{aligned} u_t &= 0.978845 \cdot u + 0.099731 \cdot u_{xx} + 0.139817 \cdot u_{yy} \\ &\quad -0.952589 \cdot u^3 + 0.993481 \cdot v^3 - 0.955152 \cdot u \cdot v^2 + 0.992257 \cdot u^2 \cdot v \end{aligned} \tag{7}$$

$$\begin{aligned} v_t &= -0.987519 \cdot u^3 + 1.175099 \cdot v + 0.106914 \cdot v_{xx} \\ &\quad +0.152016 \cdot v_{yy} - 1.159284 \cdot v^3 - 0.999191 \cdot u \cdot v^2 - 1.160521 \cdot u^2 \cdot v \end{aligned} \tag{8}$$

with relative L2 Error of (7) : 11.802% and *of (8) :* 12.239%

Library Size D	Ridge Penalty λ	STRidge Tolerance	STRidge Iterations
23 Terms	1e-6	5e-3	10
Ridge Iterations	**Train Split**	**Train Iterations**	**Train Penalty η**
500	0.8	60	1e-3

Table 3: Sparse regression parameter for system 3.

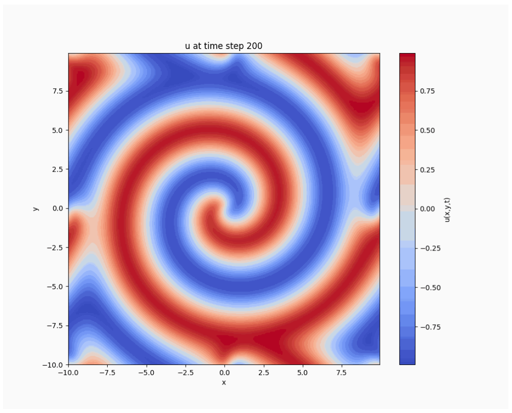


Figure 5: Heatmap plot of the **original** 2D data u at final time $t = T$

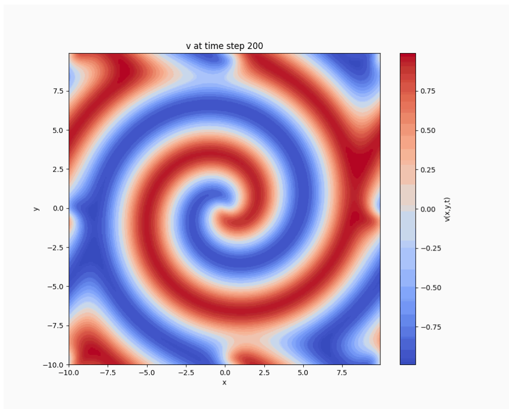


Figure 6: Heatmap plot of the **original** 2D data v at final time $t = T$

Success & Failures

For system 3, we noticed that the sparse regression does not converge on the full LSE. We randomly select 10,000 rows of the Θ matrix and the corresponding rows of the RHS vector with the sampling operator $\mathcal{C}$. Thus, **we used a compressed library** $\mathcal{C}(\Theta)$ and the associated compressed RHS $\mathcal{C}(u_t)$ or $\mathcal{C}(v_t)$ respectively for system 3. **This effectively resolves the issue and gives reasonable results (with $\sim 11 - 12\%$ of relative L2 error) for the coupled system we obtained.** However, we also observed that the quality of the selected rows of Θ largely influence our final results. **In future, improvements could be done to select a high-quality sample collection which captures the most important dynamics of the system**, or one could optionally implement a routine similar to algorithm 2 that performs regression on a wide range of randomly selected $(\mathcal{C}(\Theta), \mathcal{C}(u_t))$ pairs and find the optimal indices for sample selection. Moreover, we also noticed that the 2D problem in system 3 exhibits larger errors compared to system 1 & system 2. We assume this is mainly due to the fact that there are more terms involved in the actual solution.

Appendix A: Library Candidates

In this appendix section, we supplement information about the detailed selection of candidates used for building the library $\Theta \in \mathbb{R}^{(n \cdot m) \times D}$ for each system. We denote D as the size of the library, it equals the number of candidates in the library. n denotes the number of spatial points and m the number of time points on the $n \times m$ equidistant spatio-temporal grid $\tilde{\Omega}$.

System 1

- Size of library $D = 31$, we used 31 candidates to assemble the library Θ
- Generalized condition number of Θ is 653973.8125. Found optimal tolerance: 0.080.
- Domain consists of n = 256 spatial points and m = 101 time snapshots ($n \times m = 256 \times 101 = 25856$)
  - $\Theta \in \mathbb{R}^{25856 \times 31}$
  - $u_t \in \mathbb{R}^{25856 \times 1}$

$$\begin{aligned}\mathrm{col}(\Theta) := \mathrm{span}\{&1, u, u^2, u^2u_x, u^2u_x^2, u^2u_x^3, u^2u_{xx}, u^2u_{xx}^2, u^2u_{xx}^3,\\ &u^3, u^3u_x, u^3u_x^2, u^3u_x^3, u^3u_{xx}, u^3u_{xx}^2, u^3u_{xx}^3,\\ &uu_x, uu_x^2, uu_x^3, uu_{xx}, uu_{xx}^2, uu_{xx}^3,\\ &u_x, u_x^2, u_x^3, u_{xx}, u_{xx}^2, u_{xx}^3,\\ &u_{xxx}, u_{xxx}^2, u_{xxx}^3\}\end{aligned} \tag{9}$$

System 2

- Size of library $D = 13$, we used 13 candidates to assemble the library Θ
- Generalized condition number of Θ is 3415.815185546875. Found optimal tolerance: 0.500.
- Domain consists of n = 512 spatial points and m = 201 time snapshots ($n \times m = 512 \times 201 = 102912$)
  - $\Theta \in \mathbb{R}^{102912 \times 13}$
  - $u_t \in \mathbb{R}^{102912 \times 1}$

$$\Theta(u) := \begin{bmatrix} 1 & u & uu_t & uu_{tt} & uu_x & uu_{xx} & uu_{xxx} & u_tu_x & u_{tt} & u_{tt}u_{xx} & u_x & u_{xx} & u_{xxx} \end{bmatrix} \tag{10}$$

System 3

- Size of library $D = 23$, we used 23 candidates to assemble the library Θ
- Domain consists of $n = 256 \times 256 = 65536$ spatial points and $m = 201$ time snapshots ($n \times m = 65536 \times 201 = 13172736$)
  - $\Theta \in \mathbb{R}^{13172736 \times 13}$
  - $u_t \in \mathbb{R}^{13172736 \times 1}$

Note that in system 3, we apply the sampling operator $\mathcal{C}$ on the candidate library Θ and the associated rows from RHS ξ and solve for a smaller least squares problem as follows:

$$\arg \min_{\xi \in \mathbb{R}^D} \|\mathcal{C}(u_t) - \mathcal{C}(\Theta)\xi\|_2^2 + \lambda\|\xi\|_2^2 \tag{11}$$

- Generalized condition number of $\mathcal{C}(\Theta)$ is 101.93910217285156.
- Found optimal tolerance for u_t: 0.098; optimal tolerance for v_t: 0.107
- $\mathcal{C}(\Theta) \in \mathbb{R}^{10000 \times 23}$
- $\mathcal{C}(u_t) \in \mathbb{R}^{10000 \times 1}$
- $\mathcal{C}(v_t) \in \mathbb{R}^{10000 \times 1}$

$$\begin{aligned}&\Theta(u, v)\\ &:= \begin{bmatrix} u & u_x & u_y & u_{xx} & u_{yy} & u_{xy} & u^4 & u^3 & u^2 & v & v_x & v_y & v_{xx} & v_{yy} & v_{xy} & v^4 & v^3 & v^2 & uv & uv^2 & u^2v & uv^3 & u^3v \end{bmatrix}\end{aligned} \tag{12}$$

Appendix B: Example Library Candidates

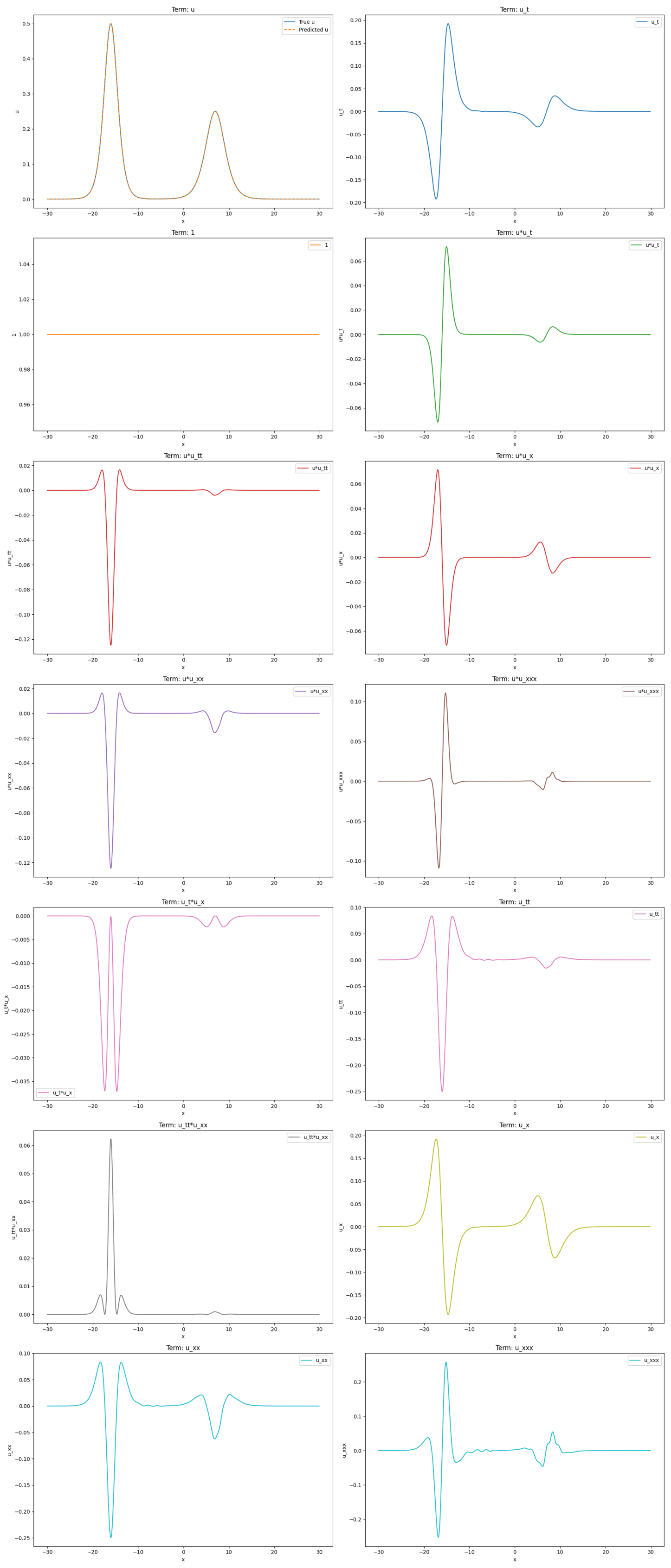


Figure 7: Visualization of u_t and all candidates used for building Θ of system 2. Computation of derivatives is done via automatic differentiation on a trained FNN for system 1 & 2. For system 3 direct finite difference with `torch.gradient` is used.

Appendix C: Sparse Regression Algorithms

The following algorithms can be found in the supplementary materials of the original PDE-Find paper [1]. Here we include them in this appendix section just for completeness. There are a series of hyperparameters one could tune when using both algorithms, in the main report one can found the values used for each system in its associated parameter table.

Algorithm 1: STRidge (Sequential Threshold Ridge Regression)

```
1: **function** STRidge($\Theta$, $U_t$, $\lambda$, tol, iters)
2:     $\hat{\xi} \leftarrow \arg\min_{\xi} \|\Theta\xi - U_t\|_2^2 + \lambda\|\xi\|_2^2$
3:     bigcoeffs $\leftarrow \left\{j : |\hat{\xi}_j| \geq \text{tol}\right\}$
4:     $\hat{\xi}[\sim \text{bigcoeffs}] \leftarrow 0$
5:     $\hat{\xi}[\text{bigcoeffs}] \leftarrow \text{STRidge}(\Theta[:, \text{bigcoeffs}], U_t, \text{tol}, \text{iters} - 1))$
6:     **return** $\hat{\xi}$
```

Algorithm 2: TrainSTRidge

In algorithm 2, we scale the hyperparameter η proportionally to the condition number of the matrix Θ. In order to better target the linear least squares problems in the scope of our project, we use the generalized condition number of a matrix for monitoring the sensitivity of the systems to be resolved as in the lecture note by Hiptmair [5]. Given $A \in \mathbb{K}^{m,n}, m \geq n, \text{rank}(A) = n$, we define the **generalized Euclidean condition number** as:

$$\text{cond}_2(A) := \sqrt{\frac{\lambda_{\max}(A^H A)}{\lambda_{\min}(A^H A)}}. \qquad (13)$$

where $\lambda_{\min}(A)$ represents the smallest eigenvalue of some matrix A, and $\lambda_{\max}(A)$ the largest eigenvalue on the contrary.

```
1: **function** TrainSTRidge($\Theta$, $U_t$, $\lambda$, $d_{\text{tol}}$, tol_iters, STR_iters)
2:     $\{\Theta^{\text{train}}, \Theta^{\text{test}}\} \leftarrow \{80/20 \text{ split of } \Theta\}$
3:     $\{U_t^{\text{train}}, U_t^{\text{test}}\} \leftarrow \{80/20 \text{ split of } U_t\}$
4:     $\eta \leftarrow 10^{-3}\kappa(\Theta)$
5:     $\xi^{\text{best}} \leftarrow \left(\Theta^{\text{train}}\right)^{-1} U_t^{\text{train}}$
6:     $\text{error}_{\text{best}} \leftarrow \|\Theta^{\text{test}}\xi^{\text{best}} - u_t^{\text{test}}\|_2^2 + \eta\ \|\xi^{\text{best}}\|_0$
7:     $\text{tol} \leftarrow \text{dtol}$
8:     **for** iter $= 1, ..., \text{tol\_iters}$ **do**
9:         $\xi \leftarrow$ **TrainSTRidge** ($\Theta^{\text{train}}$, $U_t^{\text{train}}$, $\lambda$, tol, STR_iters)
10:        **if** error $\leq \text{error}_{\text{best}}$ **then**
11:            $\text{error}_{\text{best}} \leftarrow \text{error}$
12:            $\xi_{\text{best}} \leftarrow \xi$
13:            $\text{tol} \leftarrow \text{tol} + d_{\text{tol}}$
14:        **else**
15:            $\text{tol} \leftarrow \max([0, \text{tol} - 2d_{\text{tol}}])$
16:            $d_{\text{tol}} \leftarrow \frac{2d_{\text{tol}}}{\text{tol\_iters} - \text{iter}}$
17:            $\text{tol} \leftarrow \text{tol} + d_{\text{tol}}$
18:    **return** $\xi^{\text{best}}$
```

Project Tracking

Deliverables:

✓ 1. In your project report, state your guess of the PDE for each file.
✓ 2. Describe how your algorithm works, the size of the library D you use for each file.
✓ 3. What convergence issues you encounter.
✓ 4. Possible future extensions to improve the convergence and/or generality of your method.

References

[1] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Science Advances*, vol. 3, no. 4, 2017, doi: 10.1126/SCIADV.1602614/ASSET/B84AADFB-A619-444B-B64C-7FB89C5AA4E6/ASSETS/GRAPHIC/1602614-F3.JPEG.

[2] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data: Sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 113, no. 15, pp. 3932–3937, 2015, doi: 10.1073/pnas.1517384113.

[3] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks*, vol. 3, no. 5, pp. 551–560, 1990, doi: 10.1016/0893-6080(90)90005-6.

[4] S. L. Brunton and J. N. Kutz, "Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control," *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*, pp. 1–472, 2019, doi: 10.1017/9781108380690.

[5] R. Hiptmair, *Numerical Methods for Computational Science and Engineering*, 0th ed., vol. 0. Zürich: Seminar für Angewandte Mathematik, ETH Zürich, 2024.