PDE-Find using Truncated Least Squares with dynamic thresholding

Benedict Armstrong
benedict.armstrong@inf.ethz.ch

December 2024

Introduction

As part of this year's AI in the Sciences and Engineering course, I replicated and extended PDE-Find, as described in the paper by Rudy et al. [1]. The implementation is structured as a library, capable not only of identifying partial differential equations but also of solving them using SciPy's `solve_ivp` to verify the discovered equations. The notebooks, library code, and scripts for generating all plots and figures are available in this repository.

Implementation

The first step involved constructing a library of potential PDE terms. The main challenge was accurately estimating derivatives of the solution. Initially, I implemented a finite difference method but later transitioned to using `numpy`'s `np.gradient` for more efficient and reliable derivative estimation.

For solving the sparse linear system, I implemented a Truncated Least Squares with Dynamic Thresholding (TLS-DT) algorithm, based on the STRidge method detailed in the PDE-FIND paper [1].

The implementation, structured across three Jupyter notebooks (`pde1.ipynb`, `pde2.ipynb`, and `pde3.ipynb`), handles each dataset individually. Each file also contains a more detailed analysis for each problem. The core components for the Library, including code for generating the library of terms and the TLS-DT algorithm, are encapsulated in `lib/pde_find.py` and `lib/tlsq.py`.

Truncated Least Squares with dynamic thresholding (TLS-DT)

The TLS-DT algorithm extends STRidge [1] by dynamically adjusting the threshold for coefficient selection. The threshold is raised if the number of non-zero coefficients stays the same for a number of iterations and the coefficient contains more non-zero terms than allowed. This prevents the algorithm from becoming trapped in local minima, ensuring a more robust selection of sparse terms in the PDE. The full algorithm is outlined Algorithm 1.

Results

The accuracy of the reconstructed PDEs was validated by comparing solutions generated using `scipy.solve_ivp` with the original data. The results for each PDE are summarized below.

PDE 1

Initial inspection suggested a form resembling Burgers' equation. The term library included all second-order polynomial combinations of u and it's 1st, 2nd and 3rd order derivatives (see

Table 2). The algorithm converged rapidly to Equation 1.

$$\frac{\partial(u)}{\partial(t)} = -0.102u_{xx} - 0.997uu_x$$

Equation 1: Reconstructed solution for PDE 1

The relative L_2 error across the domain, of the reconstructed solution, compared to the given data was $5.304e^{-3}$. The reconstructed solution closely matched the reference data, as shown in Figure 1.

PDE 2

For dataset 2 the PDE was not immediately obvious (at least to the untrained eye). Using a similar term library (see Table 3),the algorithm initially identified a four-term solution resembling the *Korteweg-de Vries* equation. Refinement with adjusted `max_terms` value yields Equation 2, a two term solution with only a slight increase in the relative L_2 error and more desired sparsity.

$$\frac{\partial(u)}{\partial(t)} = -1.02u_{xxx} - 6.04uu_x$$

Equation 2: Reconstructed solution for PDE 2

The relative L_2 error was $6.023e^{-2}$. The reconstructed solution again closely matched the reference data, as shown in Figure 2.

PDE 3

Dataset 3 presented a convection-diffusion equation in two dimensions. Due to dataset size, spatial and temporal dimensions were scaled by a factor of 0.5. The term library was restricted to third-order polynomial terms of u

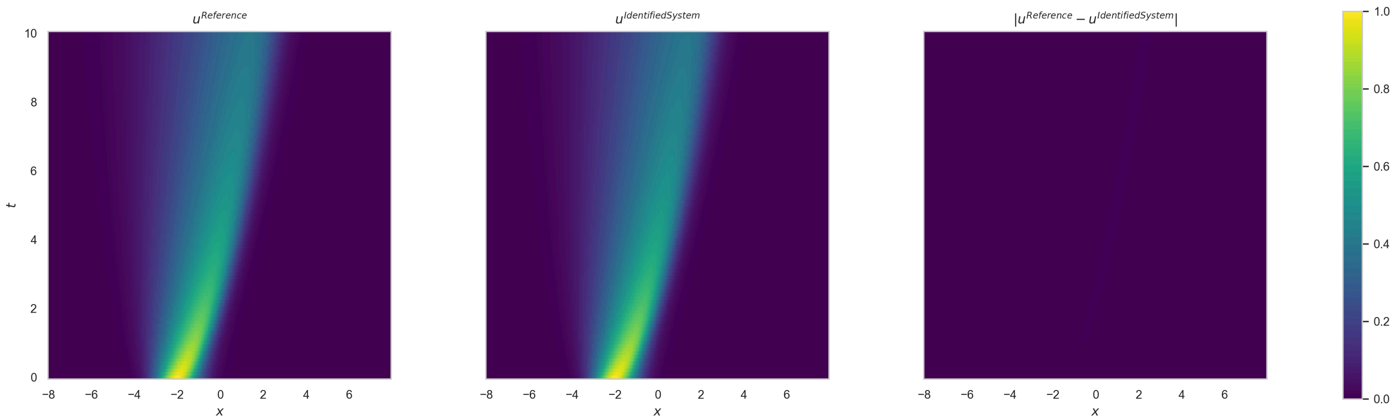

Figure 1: Reconstructed solution for PDE 1

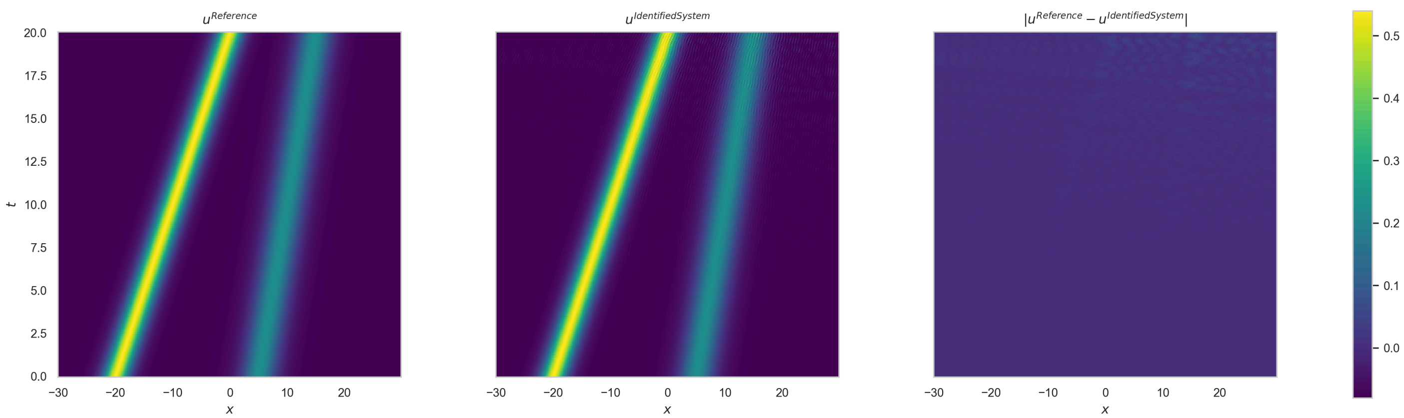

Figure 2: Reconstructed solution for PDE 2

and v with a maximum of one derivative of any order (see Table 4). The algorithm converged to Equation 3 with a relative L_2 error of $4.173e^{-1}$ for both u and v. This error is significantly higher than for the previous two PDEs, but might be due to the larger domain and higher order terms. The hyperparameter used were: `max_terms`=10 and `cutoff`=$1e^{-4}$.

A visualization (`pde3.gif`) over time is available in the `figures` folder.

PDE	Relative L_2 error
PDE 1	$5.304e^{-3}$
PDE 2	$6.023e^{-2}$
PDE 3	$4.173e^{-1}$

Table 1: Relative L_2 error for each PDE

Conclusion & Improvements

The PDE-FIND implementation successfully identified plausible PDEs for all datasets. Future improvements include:

- Extending the library to support other differentiation methods such as polynomial interpolation.
- Addressing boundary condition issues observed in PDE 3.
- Improve sub-sampling: the current implementation samples the data along entire rows/columns, random sub-sampling could improve the algorithm's performance and robustness to noise.
- Add measures to make solvers more robust to noise in the data.
- Benchmarking against other sparse regression algorithms.
- Extending the library to support non-polynomial terms.
- Decoupling solver components to allow easy integration with alternative sparse solvers.

Bibliography

[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, p. e1602614, 2017.

$$\frac{\partial(u)}{\partial(t)} = 0.828u + 0.251v - 0.807u^3 + 0.71v^3 + 0.711u^2v - 0.809uv^2 + 0.106u_{xx} + 0.103u_{yy}$$

$$\frac{\partial(v)}{\partial(t)} = -0.251u + 0.828v - 0.71u^3 - 0.807v^3 - 0.809u^2v - 0.711uv^2 + 0.103v_{xx} + 0.106v_{yy}$$

Equation 3: Reconstructed solution for PDE 3

Appendix

Algorithm 1: Truncated Least Squares with dynamic thresholding (TLS-DT)

```
X - Matrix of shape (n_samples, n_features)
y - Vector of shape (n_samples,)
cutoff - Initial cutoff value for coefficients
max_iterations - Maximum number of iterations
num_term_limit - Limit for the number of terms

coeffs = LeastSquares (X, y)
num_terms = []
original_cutoff = cutoff

for iteration in max_iterations:
    if the number of terms is consistent across two iterations and ≤ num_term_limit:
        break

    if the number of terms is consistent and ≥ num_term_limit:
        cutoff ∗ = 2

    if the number of terms differs between iterations:
        cutoff = original_cutoff

    # Identify indices of coefficients greater than the current cutoff
    indices = TRUE WHERE (coeffs > cutoff)

    if SUM (indices) = 0:
        Set the indices of the num_term_limit smallest coefficients to TRUE
    else:
        Set coefficients smaller than the cutoff to zero

    # Recalculate coefficients using only the non-zero terms
    coeffs[:, indices] = LeastSquares (X[:, indices], y)

    # Append the count of non-zero coefficients to num_terms
    num_terms append (SUM (indices))

return the final coeffs vector
```

Algorithm 1: Truncated Least Squares with dynamic thresholding (TLS-DT)

$$1, u, u_{xxx}, u_{xxx}u_{xxx}, u_{xxx}u_{xx}, u_{xxx}u_{x}, u_{xx}, u_{xx}u_{xx}, u_{xx}u_{x}, u_{x}, u_{x}u_{x}, uu, uu_{xxx}, uu_{xx}, uu_{x}$$

Table 2: Term library used for PDE 1

$$1, u, u_{xxx}, u_{xxx}u_{xxx}, u_{xxx}u_{xx}, u_{xxx}u_{x}, u_{xx}, u_{xx}u_{xx}, u_{xx}u_{x}, u_{x}, u_{x}u_{x}, uu, uu_{xxx}, uu_{xx}, uu_{x}$$

Table 3: Term library used for PDE 2

$$\begin{gathered}1, u, u_{xx}, u_{xx}v, u_{xx}vv, u_{xy}, u_{xy}v, u_{xy}vv, u_{x}, u_{x}v, u_{x}vv,\\ u_{yy}, u_{yy}v, u_{yy}vv, u_{y}, u_{y}v, u_{y}vv, uu, uu_{xx}, uu_{xx}v, uu_{xy},\\ uu_{xy}v, uu_{x}, uu_{x}v, uu_{yy}, uu_{yy}v, uu_{y}, uu_{y}v, uuu, uuu_{xx}, uuu_{xy},\\ uuu_{x}, uuu_{yy}, uuu_{y}, uuv, uuv_{xx}, uuv_{xy}, uuv_{x}, uuv_{yy}, uuv_{y},\\ uv, uv_{xx}, uv_{xy}, uv_{x}, uv_{yy}, uv_{y}, uvv, uvv_{xx}, uvv_{xy}, uvv_{x}, uvv_{yy},\\ uvv_{y}, v, v_{xx}, v_{xy}, v_{x}, v_{yy}, v_{y}, vv, vv_{xx}, vv_{xy}, vv_{x}, vv_{yy}, vv_{y},\\ vvv, vvv_{xx}, vvv_{xy}, vvv_{x}, vvv_{yy}, vvv_{y}\end{gathered}$$

Table 4: Term library used for PDE 3