Discovery of PDEs by Sequential Thresholding*

Natan Dominko Kobilica
Technische Universität München
(Dated: March 19, 2024)

We review the PDE-FIND algorithm, first introduced by Rudy et al. in 2017 [1]. We motivate it as an improvement to linear regression in terms of physical relevance. We discuss the implementation of the problem on a discrete grid in one spatial + one temporal dimension, though the generalization to higher dimensions is conceptually trivial. We point out practical stability and accuracy issues and preventive practices. We then test the algorithm on different mock datasets. We discuss our result along with possible further improvements.

I. INTRODUCTION

A. Linear Regression with Sequential Thresholding

Linear regression gives the best approximation of a time series data $\mathbf{y}(t_k)$ via some linear combination of arbitrary functions $\mathbf{f}_i(t)$. The loss function $\sum_{i,k} \|\xi_i \mathbf{f}_i(t_k) - \mathbf{y}(t_k)\|_2^2$ is minimized by some parameter values $\boldsymbol{\xi}_{\text{opt}}$. The approximation agrees more closely with the data when we increase the number of available functions $\mathbf{f}_i$. Too many functions, however, cause problems, such as overfitting.

A possible remedy is modifying the loss function to include a term $\|\boldsymbol{\xi}\|_0$, which counts the number of nonzero components of $\boldsymbol{\xi}$. The new $\boldsymbol{\xi}_{\text{opt}}$ will be more sparse and thus avoid overfitting better. Working with the 0-norm analytically is impossible and hence we turn to an approximate algorithmic solution.

Starting from the $\boldsymbol{\xi}_{\text{opt}}$ of the linear regression, we conduct *thresholding* i.e. setting all small coefficients to zero. Then, we solve the linear regression again, with the significantly reduced number of available functions $\mathbf{f}_i$. The procedure can then be repeated until the only remaining functions have coefficients above the threshold.

B. Identification of Dynamics

In physical systems it is often advantageous to learn the underlying dynamical law of a system. Supposing a system $\mathbf{y}$ follows a law of the form $\mathbf{y}'(t) = \mathbf{F}(\mathbf{y}(t), t)$, we can learn this law from data via sequential thresholding. We need only replace $\mathbf{y}(t_k)$ with $\mathbf{y}'(t_k)$ in the linear regression loss function. Since the observable $\mathbf{y}'$ is usually unavailable, we can use finite differences to estimate it.

In this setting, the importance of sequential thresholding becomes even more apparent: Most physical laws are simple and include at most a few terms. The estimation of the underlying law of dynamics combined with sequential thresholding is known as the Sparse Identification of Nonlinear Dynamics (SINDy) algorithm [2].

C. Spatially Dependent Dynamics

By including spatial scales to the dynamics, we can model partial differential equations (PDEs). Rudy et al. [1] introduced PDE-FIND, an algorithm that mirrors the SINDy algorithm with the addition of spatial operators such as the derivative.

In this monograph we provide a brief outline of the PDE-FIND procedure (Chapter II) to go along with the Julia implementation of the algorithm (see the accompanying GitHub repository). The repository also includes some examples of usage (see Chapter III).

We note that the algorithm implemented here differs slightly from the one introduced in [1]. In the provided examples, however, our implementation performed at the same level or better in terms of accuracy and speed.

It should also be noted that we have only implemented PDE-FIND in one spatial dimension and only for one component PDEs at the time of writing. Generalization to higher dimensions should be conceptually simple.

II. PROBLEM

A. Formulation and ansatz

Suppose we know that the function $u(t, x)$, that solves a PDE of the following form

$$u_t(t, x) = L(u, u_x, u_{xx}, \dots). \tag{1}$$

We assume that L can be described as a linear combination of some basis functions. A simple choice that describes most of the common PDEs is the polynomial basis $\{\Theta_i\}_i$. We label its entries as

$$\Theta_{i_1 i_2 \dots}(u, v, \dots) = u^{i_1} v^{i_2} \cdots. \tag{2}$$

which simplifies our ansatz for (1) to

$$u_t \approx \sum_i \xi_i \Theta_i(u, u_x, u_{xx}, \dots), \tag{3}$$

---

* Final project of the graduate course *Modeling of Dynamical Systems in Julia* by prof. dr. Maximilian Gelbrecht and dr. Alistair White at Technische Universität München
Accompanying GitHub repository: `https://github.com/dknatan/PDEfind.jl`

where we seek the optimal $\boldsymbol{\xi}$ which is also sufficiently sparse.

B. Discretization

Suppose that the solution function $u(t,x)$ is given for some uniformly discretized t and x. Denoting $j \in \{0,1,\dots,m\}$ as the time index and $k \in \{0,1,\dots n\}$ as the space index we have in total $(m+1)(n+1) =: N$ data points labeled $u_{jk} := u(t_j, x_k)$. We construct the time derivatives $(u_t)_{jk}$ and spatial derivatives $(u_x)_{jk}$ from u_{jk} (see chapter II E).

The discretized equation (3) reads

$$(u_t)_{jk} \approx \sum_i \Theta_i(u_{jk}, (u_x)_{jk}, (u_{xx})_{jk}, \dots)\xi_i, \qquad (4)$$

or in matrix notation

$$\mathbf{u}_t \approx \boldsymbol{\Theta}\boldsymbol{\xi}. \qquad (5)$$

Which is a linear regression problem with loss function

$$\|\boldsymbol{\Theta}\boldsymbol{\xi} - \mathbf{u}_t\|_2^2, \qquad (6)$$

Notice that the number of terms in (4) grows very quickly when we include higher derivatives and higher degree polynomials. We assume the dynamics is **simple** i.e. $\boldsymbol{\xi}$ is sparse. Therefore, we seek the minimizer of

$$\|\boldsymbol{\Theta}\boldsymbol{\xi} - \partial_t\mathbf{u}\|_2^2 + \eta\|\boldsymbol{\xi}\|_0^2, \qquad (7)$$

with $\|\boldsymbol{\xi}\|_0$ denoting the number of nonzero components of $\boldsymbol{\xi}$ and η some parameter balancing the contribution of the two terms.

C. Ridge Regularization

Problem (7) is NP-hard [1] and can only be solved approximately. As in the case of the SINDy algorithm [2] we conduct sequential thresholding. As a precaution step we include a *ridge regularization* term, that ensures all components of $\boldsymbol{\xi}$ are well behaved.

$$\boldsymbol{\xi}_{\text{opt}} = \text{argmin}_{\boldsymbol{\xi}} \tfrac{1}{N}\|\boldsymbol{\Theta}\boldsymbol{\xi} - \mathbf{u}_t\|_2^2 + \lambda\|\boldsymbol{\xi}\|_2^2, \qquad (8)$$

$$= \tfrac{1}{N}(\tfrac{1}{N}\boldsymbol{\Theta}^\top\boldsymbol{\Theta} + \lambda\,\text{id})^{-1}\boldsymbol{\Theta}^\top\mathbf{u}_t. \qquad (9)$$

Note that the prefactor of $\frac{1}{N}$ was added just to ensure better scaling when changing the discretization. This method will, however, allow for small nonzero components of $\boldsymbol{\xi}$. This issue is solved (as before) by repeatedly setting smaller components of $\boldsymbol{\xi}$ to zero in what is known as the *Sequential Thresholded Ridge Regression* (STRidge) algorithm.

The pseudocode of the STRidge algorithm:

```
STRidge(theta, iters, ...)
  xi = ridge(theta, ...)     # see equation (9)
  mask = |xi| > th           # big coefficients
  if iters == 1:
    return xi[mask]
  else:
    return STRidge(theta[:,mask], iters-1, ...)
```

In the algorithm above, we need to specify a threshold. To circumvent this, we conduct a simple optimal threshold search.

The pseudocode for STRidge at optimal threshold:

```
TrainSTRidge()
  th = big_th # start with large threshold
  xi_best = STRidge(th, ...)
  e_best = sparse_loss(xi_best) # see eq. (7)
  for i = 1:max_tol_iters
    th /= (1 + d)
    xi = STRidge(th, ...)
    if len(xi) - len(xi_best) > 1
    th *= (1 + d)
    d /= 2
      else
        e = sparse_loss(xi)   # see eq. (7)
        if e > e_best
          break
        else
          e_best = e
          xi_best = xi
```

Algorithm `TrainSTRidge` rigourosly finds the first local minimum and the corresponding minimizer $\boldsymbol{\xi}_{\text{opt}}$ of equation (7). It is more robust than the one introduced in [1] because it takes advantage of the fact that the sparse loss landscape changes incrementally when $\|\boldsymbol{\xi}\|_0$ changes. The algorithm is also more efficient, since it initializes at high threshold and adds terms to the RHS one by one.

The above can easily be generalized to more spatial dimensions $x, y, \dots$ and to multiple components $u^1, u^2, \dots$ of the PDE (1) in question.

D. Practical Implementation Advice

1. *Function Library Normalization:* To ensure stability, we normalize each entry Θ_i to unit variance before using equation (9).
2. *Train-Test Split:* In algorithm `TrainSTRidge` we optimize with some subset of data $\boldsymbol{\Theta}_{\text{train}}, (\mathbf{u}_t)_{\text{train}}$ and estimate the error with the remaining data $\boldsymbol{\Theta}_{\text{test}}, (\mathbf{u}_t)_{\text{test}}$.
3. *Choice of λ:* The ridge regression parameter λ should be large enough to ensure numerical stability of the inverse in equation (9), but small enough not to significantly change the predicted dynamics.

4. *Choice of η:* The sparse regression parameter η determines the loss function landscape w.r.t. the tolerance τ. Contrary to [1], we already used a slightly modified sparse loss function and hence ended up using $\eta = 10^{-6} \times$ condition number($\mathbf{\Theta}$) unless stated otherwise[3].

E. Numerical Derivatives

Mock data only include noise at computer precision level and it suffices to calculate temporal and spatial derivatives with finite difference methods of specified order.

For noisy "real-world" data one would need to use more robust methods of taking derivatives, which we have not yet implemented.

III. EXAMPLES

A. Diffusion equation

We prepare test data by solving the diffusion equation $u_t = u_{xx}$ on $t \in [0, 0.3]$, $x \in [0, 1]$ with boundary conditions $u(t, 0) = u(t, 1) = 0$ and initial condition $u(0, x) = u_0(x)$.

An important side note: if we prepared the initial condition as $u_0(x) = \sin(\pi x)$ we run into a problem since $u_{xx}(x, t) = -\pi^2 u(x, t)$. Hence the columns Θ_{100} and Θ_{001} are co-linear and hence equally represented in the solution ($|\xi_{100}| = |\xi_{001}|$). This can always occur if the initial condition is an eigenfunction of any of the Θ_i.

We proceed by setting $u_0(x) = \sin(\pi x) + \frac{1}{2}\sin(3\pi x)$ and prepare the data on a grid with $\mathrm{d}t = 2 \times 10^{-3}$ and $\mathrm{d}x = 4 \times 10^{-3}$. If we limit ourselves to spatial derivatives of maximum degree 2 and polynomials of maximum degree 2 the RHS of (3) is given as

$$\begin{aligned} u_t = \xi_{000} + \xi_{100} u + \xi_{010} u_x + \xi_{001} u_{xx} + \\ + \xi_{200} u^2 + \xi_{110} u u_x + \xi_{101} u u_{xx} + \\ + \xi_{020} (u_x)^2 + \xi_{011} u_x u_{xx} + \xi_{002} u_{xx}^2. \end{aligned} \tag{10}$$

With unknown coefficients ξ_i.

At $\lambda = 1 \times 10^{-3}$ and threshold 0.1 the algorithm determines the coefficient $\xi_{001} = 1$ with a precision of 10^{-4}.

The interested reader is invited to further explore the notebook on the GitHub repository, where we show a plot of the error landscape as a combination of the l^2 and l^0 norm and investigate the behaviour of the algorithm at different values of λ.

B. Advective Diffusion

We now add a second term, which is much stronger:

$$u_t = -6 u_x u + 0.1 u_{xx},$$

on $t \in [0, 1], x \in [0, 1]$ with periodic boundary conditions and a centered Gaussian of width 0.1 as the initial condition.

In this case, the algorithm correctly predicts the dynamics with 10^{-2} precision. Similar aspects are explored.

C. Many Terms

We also included an example which has many terms

$$u_t = u - 3 u_x u + 0.1 u_{xx} - 0.4 u_x,$$

again on $t \in [0, 1], x \in [0, 1]$ with periodic boundary conditions and a centered Gaussian of width 0.1 as the initial condition. Even in this case, the algorithm performed at 10^{-2} precision.

IV. DISCUSSION AND OUTLOOK

In the described tests, the algorithm performs as expected. Notable changes to [1] are in the `TrainSTRidge` algorithm.

Most importantly, we chose $\|\boldsymbol{\xi}\|_0^2$ instead of $\|\boldsymbol{\xi}\|_0$ to promote sparsity. This choice is made by arguments of proportionality scaling of the l_2 and l_0 terms.

Secondly, we initiated the optimal threshold search from the high-threshold side and terminate the search at the first local minimum of the sparse error function. Heuristically, if one should want more terms included in the dynamics one can lower the η multiplier and rerun the algorithm.

As mentioned, many features remain to be implemented, namely higher spatial dimensions, multi component dynamics and noise-robust numerical derivatives.

[1] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, Data-driven discovery of partial differential equations, Science Advances **3**, 10.1126/sciadv.1602614 (2017).

[2] S. L. Brunton, J. L. Proctor, and J. N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proceedings of the National Academy of Sciences **113**, 3932–3937 (2016).

[3] The choice of both are explored in the supplementary `examples` notebooks available in the GitHub repository.