Supplementary Materials for

Data-driven discovery of partial differential equations

Samuel H. Rudy, Steven L. Brunton, Joshua L. Proctor, J. Nathan Kutz

This PDF file includes:

1 Introduction

This supplementary document provides a detailed description of the methods and algorithms used to discover governing partial differential equations from time series data collected on a spatial domain. Methods for data-driven discovery of dynamical systems [1] include equation-free modeling [2], artificial neural networks [3], nonlinear regression [4], empirical dynamic modeling [5, 6], modeling emergent behavior [9], and automated inference of dynamics [10, 11, 12]. Other data-driven methods include normal form identification in climate [7], nonlinear Laplacian spectral analysis [8], modeling emergent behavior [9], Koopman analysis [26, 29, 38, 39], and automated inference of dynamics [10, 11, 12]; reference [11] provides an excellent review. Seminal contributions leveraging symbolic regression and an evolutionary algorithm [13, 14] were capable of directly determining nonlinear dynamical system from data. More recently, sparsity [15] has been used to robustly determine, in a highly efficient computational manner, the governing dynamical system [17, 27]. Both the evolutionary [14] and sparse [27] symbolic regression methods avoid overfitting by selecting parsimonious models that balance model accuracy with complexity via Pareto analysis. More generally, sparsity has recently been used for data-driven discovery of dynamical systems [24, 27, 28, 37, 42, 44, 45, 47, 50].

To be more precise about the various contributions, we highlight some of the technical achievements of the above referenced works and their application context. Most of the methods are regression frameworks for modeling the system dynamics. The majority of the work cited is specific to ODEs, including those derived from discretized PDEs [2, 3, 4, 5, 6, 11, 12]. The mathematical methodologies applied are extremely diverse and beyond the scope of the article to summarize in concise and precise manner. We again point to Ref. [11] since it provides a thorough review. Regression frameworks extend to building models (linear or nonlinear) that best represent the collection of time series data [7, 8, 26, 29, 38, 39]. Such methods go under the moniker of Dynamic Mode Decomposition, Koopman theory and/or diffusion maps. The seminal contribution from Lipson and co-workers [10, 13, 14] was a methodology, based upon a genetic algorithm search (i.e. random combinations of right hand side terms), for exploring a large, perhaps combinatorially so, number of potential ODEs. The method was capable of producing parsimonious and interpretable representations of the dynamics and *discovering* the correct ODEs that generated the data. A more efficient algorithm for achieving the parsimonious solution used sparsity promoting techniques to determine the governing ODEs [27]. However, the methods above have not addressed discovering principled, *parsimonious and interpretable governing PDE derivations*. The current work extends such methods to PDE systems. Note, the method finds the fundamental form of the PDE, not its discretized version.

2 PDE-FIND

The PDE-FIND algorithm discussed in this work is a method for discovering the governing equation for a discretized dataset which we assume to be the solution to a PDE taking the form

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

where subscripts denote partial differentiation and μ represents parameters in the system. It is assumed that the function N may be expressed as a sum of a small number of terms, which is certainly the case for the PDEs considered here and/or widely used in practice. We denote $\mathbf{U}$ to be a matrix containing the values of u and $\mathbf{Q}$ as a matrix containing additional information that may be relevant, such as dependencies on the absolute value of $|u|$, or another time varying function interacting with u.

While PDE-FIND does not have restrictions on the functional form of candidate terms in N, polynomial nonlinearities are common in many of the examples in this work and in many of the canonical models of mathematical physics. We may also consider data that is on a higher dimensional spatial domain, in which case we simply allow for derivatives with respect to each spatial dimension; u_x, u_y, u_{xy}, $u_x^2 u_{yy}$, etc.

PDE-FIND creates a large library of candidate terms that may appear in N, including nonlinearities and partial derivatives, and then selects a sparse subset of *active* terms from this list. In general, we always assume that the time evolution of a complex-valued function may depend on it's magnitude. The first step is to take the derivatives of the data with respect to time and each spatial dimension. Derivatives are taken either using finite differences for clean data, or when noise is added, with polynomial interpolation. The derivatives and the function itself are then combined into a matrix $\mathbf{\Theta}(\mathbf{U}, \mathbf{Q})$

$$\mathbf{\Theta}(\mathbf{U}, \mathbf{Q}) = \begin{bmatrix} 1 & \mathbf{U} & \mathbf{U}^2 & \dots & \mathbf{Q} & \dots & \mathbf{U}_x & \mathbf{U}\mathbf{U}_x & \dots & \mathbf{Q}^2\mathbf{U}^3\mathbf{U}_{xxx} \end{bmatrix} \tag{2}$$

Each column of $\mathbf{\Theta}$ contains all of the values of a particular candidate function across all of the grid points on which data is collected. Therefore, if we have data on an $n \times m$ grid (e.g. a 256×100 grid represents 256 spatial measurements at 100 time points) and have 50 candidate terms in the PDE, then $\mathbf{\Theta} \in \mathbb{C}^{256 \cdot 100 \times 50}$. We also take the time derivative to compute $\mathbf{U}_t$ and reshape it into a column vector just like we did the columns of $\mathbf{\Theta}$. This gives a linear equation representing our PDE

$$\mathbf{U}_t = \mathbf{\Theta}(\mathbf{U}, \mathbf{Q})\xi \tag{3a}$$

$$\begin{bmatrix} u_t(x_0, t_0) \\ u_t(x_1, t_0) \\ u_t(x_2, t_0) \\ \vdots \\ u_t(x_{n-1}, t_m) \\ u_t(x_n, t_m) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & u(x_0, t_0) & u_x(x_0, t_0) & \dots & u^5 u_{xxx}(x_0, t_0) \\ 1 & u(x_1, t_0) & u_x(x_1, t_0) & \dots & u^5 u_{xxx}(x_1, t_0) \\ 1 & u(x_2, t_0) & u_x(x_2, t_0) & \dots & u^5 u_{xxx}(x_2, t_0) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & u(x_{n-1}, t_m) & u_x(x_{n-1}, t_m) & \dots & u^5 u_{xxx}(x_{n-1}, t_m) \\ 1 & u(x_n, t_m) & u_x(x_n, t_m) & \dots & u^5 u_{xxx}(x_n, t_m) \end{bmatrix}}_{\text{Example } \mathbf{\Theta} \text{ for real valued function in one spatial dimension}} \begin{bmatrix} \xi \end{bmatrix} \tag{3b}$$

Note that if we assume $\mathbf{\Theta}$ is an over complete library, meaning $\mathbf{\Theta}$ has a sufficiently rich column space that the dynamics will be in it's range, then the PDE should be well-represented by Eq. (3a) with a sparse vector of coefficients ξ. This amounts to picking enough candidate functions that

the full PDE may be written as weighted sum. Each row of this linear system represents an obser-
vation of the dynamics at a particular point in time and space.

$$u_t(x, y) = \sum_j \Theta_j(u(x, t), q(x, t))\xi_j. \tag{4}$$

For an unbiased representation of the dynamics, we would simply solve the least squares problem
for ξ. However, even with the only error coming from numerical round-off, the least-squares solu-
tion may be inaccurate. In particular, ξ will have predominantly nonzero values suggesting a PDE
with every functional form contained in the library. Most importantly, for regression problems
similar to (4), the least squares problem is poorly conditioned. Error in computing the derivatives
(already an ill-conditioned problem with noise) will be magnified by numerical errors when in-
verting Θ. Furthermore, measurement error also affects the least-squares solution. Thus if least
squares is used, it can radically change the qualitative nature of the inferred dynamics. Instead,
we utilize sparse regression to approximate a solution of

$$\xi = arg\,min_{\hat{\xi}}\|\Theta\hat{\xi} - \mathbf{U}_t\|_2^2 + \lambda\|\hat{\xi}\|_0 \tag{5}$$

This assures that terms will only show up in the derived PDE if their effect on the error $\|\Theta\hat{\xi} - \mathbf{U}_t\|$
outweight their addition to $\|\hat{\xi}\|_0$. The ℓ^0 term makes this problem *np*-hard. In the next section we
discuss methods for approximating solutions to Eq. (5).

2.1 Sparse Regression for PDE-FIND

The convex relaxation of the ℓ^0 optimization problem in (5) is

$$\xi = arg\,min_{\hat{\xi}}\|\Theta\hat{\xi} - \mathbf{U}_t\|_2^2 + \lambda\|\hat{\xi}\|_1 \tag{6}$$

We originally utilized the least absolute shrinkage and selection operator (LASSO) to solve this
convex optimization problem defined by PDE-FIND [15]. However, LASSO tends to have diffi-
culty finding a sparse basis when the data matrix Θ has high correlations between columns, which
is the case in many of our examples [21].

An alternative method for sparse regression was utilized in [27], called sequentially thresh-
olded least squares (STLS). In STLS, a least squares predictor is obtained and a hard threshold is
performed on the regression coefficients. The process is repeated recursively on the remaining
nonzero coefficients. This is illustrated in algorithm 1 when $\lambda = 0$. STLS outperforms LASSO
in some cases but does not avoid the challenge of correlation in the data. Using a regularizer for
the least squares problem can help avoid problems due to correlations. Ridge regression is an ℓ^2
regularized variation of least squares corresponding to the maximum a posteriori estimate using
a Gaussian prior [40]. It is defined by

$$\begin{aligned}\hat{\xi} &= argmin_{\xi}\|\Theta\xi - \mathbf{U}_t\|_2^2 + \lambda\|\xi\|_2^2 \\ &= \left(\Theta^T\Theta + \lambda I\right)^{-1}\Theta^T\mathbf{U}_t\end{aligned} \tag{7}$$

We substitute ridge regression for least squares in STLS and call the resulting algorithm Sequential
Threshold Ridge regression (STRidge), outlined in algorithm 1. Note that for $\lambda = 0$ this reduces
to STLS. STRidge had the best empirical performance for PDE-FIND of any sparse regression
algorithm tested in this work.

    Since each value of the threshold tolerance will give a different level of sparsity in the final
solution, we also used a separate method to find the best tolerance. Predictors are trained at

varying tolerances and their performance on a holdout set, taking into account an ℓ^0 penalty, is used to find the best tolerance. We set the ℓ^0 penalty to be proportional to the condition number of Θ, enforcing sparsity in the case of highly correlated and ill-conditioned data. A multiplier of 10^{-3} was used based on empirical evidence. Our method for searching for the optimal tolerance is outlined in algorithm 2. Arguments passed into the search algorithm include Θ, $\mathbf{U}_t$, λ, and STR_iters which are passed directly to STRidge as well as d_tol and tol_iters. d_tol tells the search algorithm how large of a step to take while looking for the optimal tolerance and tol_iters indicates how many times the algorithm will refine it's guess at the best tolerance.

As a preprocessing step, each column of Θ is normalized to unit variance. This is especially useful if the function is not roughly $\mathcal{O}(1)$ so that higher powers are either very large or small. In all of the examples presented in our paper, columns of Θ are normalized to unit length before solving for the sparse vector of coefficients ξ. A final prediction of ξ is obtained by regressing the non-normalized data onto the identified terms. The only instance in which this was less successful than STRidge without normalization was for identifying the advection diffusion equation from a biased random walk.

Algorithm 1: STRidge($\Theta, \mathbf{U}_t, \lambda, tol, \text{iters}$)

$\hat{\xi} = arg\,min_\xi \|\Theta\xi - \mathbf{U}_t\|_2^2 + \lambda\|\xi\|_2^2$ # ridge regression

bigcoeffs = $\{j : |\hat{\xi}_j| \geq tol\}$ # select large coefficients

$\hat{\xi}[\sim \text{bigcoeffs}] = 0$ # apply hard threshold

$\hat{\xi}[\text{bigcoeffs}] = \text{STRidge}(\Theta[:, \text{bigcoeffs}], \mathbf{U}_t, tol, \text{iters} - 1)$

         # recursive call with fewer coefficients

return $\hat{\xi}$

Algorithm 2: TrainSTRidge($\Theta, \mathbf{U}_t, \lambda, d_{tol}, \text{tol\_iters}, \text{STR\_iters}$)

First split the data into training and testing sets

$$\left.\begin{array}{l} \Theta \to [\Theta^{train}, \Theta^{test}] \\ \mathbf{U}_t \to [\mathbf{U}_t^{train}, \mathbf{U}_t^{test}] \end{array}\right\} \ 80/20 \text{ split}$$

Set an appropriate ℓ^0-penalty. The following worked well empirically
$\eta = 10^{-3}\kappa(\Theta)$

Get a baseline predictor
$\xi_{best} = \left(\Theta^{train}\right)^{-1}\mathbf{U}_t^{train}$
$\text{error}_{best} = \|\Theta^{test}\xi_{best} - \mathbf{U}_t^{test}\|_2^2 + \eta\|\xi_{best}\|_0$

Now search through values of tolerance to find the best predictor
$tol = d_{tol}$
for $iter = 1, \ldots, \text{tol\_iters}$:

>   # Train and evaluate performance
> $\xi = \text{STRidge}(\Theta^{train}, \mathbf{U}_t^{train}, \lambda, tol, \text{STR\_iters})$
> $\text{error} = \|\Theta^{test}\xi - \mathbf{U}_t^{test}\|_2^2 + \eta\|\xi\|_0$
>
>   # Is the error still dropping?
>   if $\text{error} \le \text{error}_{best}$:
>   > $\text{error}_{best} = \text{error}$
>   > $\xi_{best} = \xi$
>   > $tol = tol + d_{tol}$
>
>   # Or is tolerance too high?
>   else:
>   > $tol = \max([0, tol - 2d_{tol}])$
>   > $d_{tol} = \frac{2d_{tol}}{\text{tol\_iters} - iter}$
>   > $tol = tol + d_{tol}$

return ξ_{best}

Several other methods exist for finding sparse solutions to least squares problem. Greedy algorithms have been shown to exhibit good performance on sparse optimization problems including PDE-FIND but in some cases were less reliable than STRidge [20]. While STRidge with normalization works well on almost all of the examples we tested (advection diffusion being the exception), we do not make the claim that it is optimal. The elastic-net algorithm, which has been shown to show advantages over LASSO, was also tested and found to be less effective for sparse regression than STRidge. [21]. If additional information regarding the PDE is known, for instance if we know one of the terms is nonzero, then this may be incorporated into the penalty on the coefficients.

2.2 Numerical Differentiation

When using PDE-FIND on clean data from numerical simulations we take derivatives with second order finite differences [35, 36]. Numerically differentiating noisy data is considerably more challenging. If one uses finite difference techniques on a grid with spacing $\mathcal{O}(h)$ and noise with amplitude $\mathcal{O}(\epsilon)$ then the dth derivative will have noise approximately of $\mathcal{O}(\frac{\epsilon}{h^d})$, which will result in numerical derivatives being dominated by the effects of noise. We consider four other methods

for numerical differentiation when there was noise added to the solution.

A simple variation of finite differences is to use a smoothing technique on the noisy data such as interpolation with a spline or convolving with a smoothing kernel. We tried the latter, using a Gaussian smoothing kernel on noisy data prior to taking derivatives with finite differences. While the derivatives we obtained seemed to be free of noise, they were sufficiently biased to create problems with identifying the dynamics. Convolving with a Gaussian has the effect of nearly eliminating higher frequency components of a signal, as it is equivalent to multiplying the spectral representation by a Gaussian. This smooths out sharp inflections in the curve. Tikhonov differentiation finds a numerical derivative $\hat{f}'$ for a function f by balancing the closeness of the integral of $\hat{f}'$ to f with the smoothness of $\hat{f}'$ [22]. The continuous problem and discrete version used in practice are given by

$$\hat{f}'(x) = argmin_g \left\| \int_{x_0}^{x} g(s)\, ds - (f(x) - f(x_0)) \right\|_2^2 + \lambda \left\| \frac{dg}{dx} \right\|_2^2$$
$$\hat{f}' = argmin_g \|Ag - (f - f_0)\|_2^2 + \lambda \|Dg\|_2^2$$

In the discrete problem, A is a trapezoidal approximation to the integral and D is a finite difference approximation to the derivative. The problem has closed form solution given by

$$\hat{f}' = \left(A^T A + \lambda D^T D\right)^{-1} A^T f$$

Tikhonov differentiation, similar to using a smoothing kernel, results in a far smoother numerical derivative than finite differences but also introduces a small amount of bias, particularly for functions with large higher order derivatives.

When the data is on a periodic domain, the best method for taking the dth derivative may be via the discrete Fourier transform and multiplication by $(ik)^d$ in the frequency domain. To combat the effects of noise one could use a filter in the frequency domain. Doing so, however, would require a principled method for deciding exactly how to threshold high frequency terms without distorting the shape of the curve. Furthermore, we cannot always assume a period spatial domain or use the fourier transform to differentiate with respect to time. Spectral differentiation was not implemented in the examples in this manuscript. We suspect this would be a promising tool for data considered on a periodic domain, or a sufficiently wide domain that the data goes to zero at the boundaries.

The method we found to be the easiest to implement and most reliable for noisy data was polynomial interpolation [22]. For each datapoint where we compute a derivative, a polynomial of degree p was fit to greater than p points and derivatives of the polynomial were taken to approximate those of the numerical data. Points close to the boundaries where it was difficult to fit a polynomial were not used in the regression. This method is far from perfect; data close to the boundaries was difficult to differentiate and the result of PDE-FIND depended strongly on the degree of polynomial and number of points used to fit it. For a more principled but involved approach to polynomial differentiation, see [23].

2.3   Subsampling data

For large datasets such as those with more than one spatial dimension, PDE-FIND can be used on subsampled data. We randomly select a set of spatial points and evenly sample the data in time at a lower frequency than data is collected, resulting in the use of only a fraction of the dataset. Mathematically, this amounts to ignoring a fraction of the rows in the linear system $\mathbf{U}_t = \boldsymbol{\Theta}(\mathbf{U}, \mathbf{Q})\xi$ as illustrated in fig. S1 panels 2a and 2b.
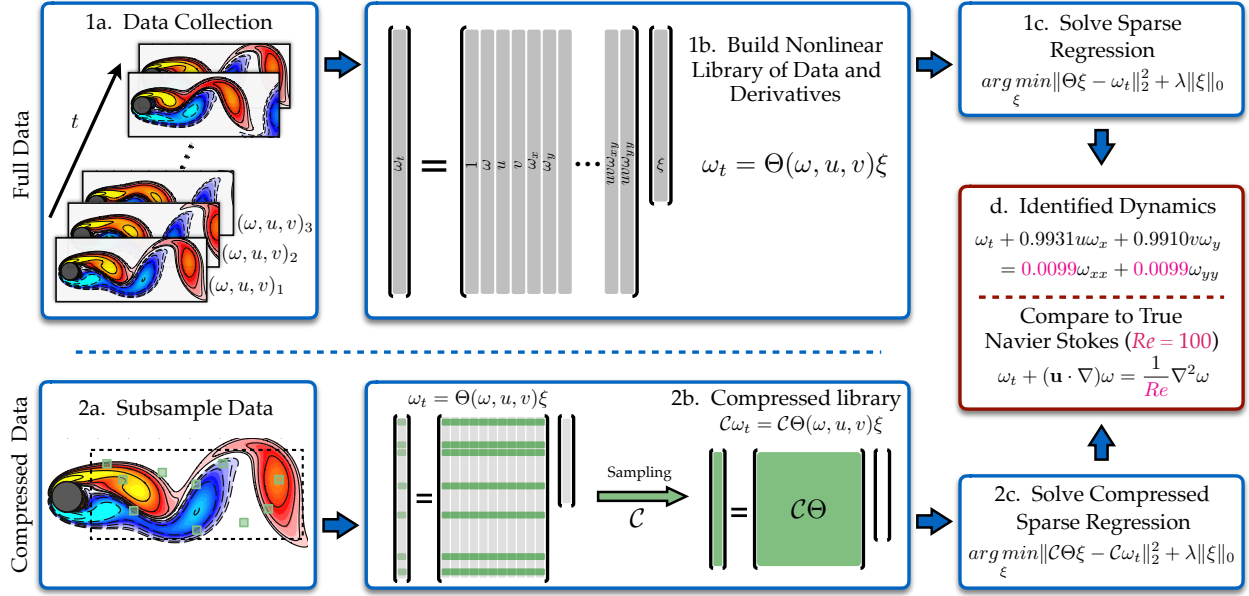
fig. S1. Steps in the PDE functional identification of nonlinear dynamics (PDE-FIND) algorithm, applied to infer the Navier-Stokes equation from data. **1a.** Data is collected as snapshots of a solution to a PDE. **1b.** Numerical derivatives are taken and data is compiled into a large matrix Θ, incorporating candidate terms for the PDE. **1c.** Sparse regressions is used to identify active terms in the PDE. **2a.** For large datasets, sparse sampling may be used to reduce the size of the problem. **2b.** Subsampling the dataset is equivalent to taking a subset of rows from the linear system in (3). **2c.** An identical sparse regression problem is formed but with fewer rows. **d.** Active terms in ξ are synthesized into a PDE.

Though we only use a small fraction of the spatial points in the linear system, nearby points are needed to evaluate the derivative terms in the library. The derivatives were computed via polynomial interpolation, using a small number of points close to the point in question to fit to a polynomial. Therefore, while subsampling uses only a small fraction of the points in the regression, we are using local information around each of the measurement.

2.4 PDE-FIND for a Fokker-Planck equation

Under a fairly nonrestrictive set of assumptions regarding a stochastic trajectory, a PDE may also be derived for the distribution function of the future position from just the single single trajectory, also known as the Fokker-Planck equation [32]. Let $X(t)$ be the position of a particle undergoing a random walk. We assume the trajectory follows the rule that the displacement of the particle over an interval of time t, $X(t+\tau) - X(\tau)$, may be predicted according to a probability distribution which at time zero is a point mass and which does not depend on τ or $X(\tau)$. That is

$$X(t+\tau) - X(\tau) \sim u(x,t) \text{ where } u(x,0) = \delta(x) \tag{8}$$

The important point here is that we assume enough to justify splitting the time series into pieces that all follow the same PDE, independent of their location in time and space. PDE-FIND looks for the relation $u_t = N(u, u_x, \ldots)$ by approximating the distribution function u using histograms. In the paper we demonstrate this as a method for computationally deriving the diffusion equation from a single trace of Brownian motion. In the supplemental code it is also demonstrated for deriving an advection diffusion equation from a biased random walk.

We start with a single time series consisting of evenly spaced measurements of the stochastic trajectory, $X = (X_0, X_1, \ldots, X_n)$. This time series is split into many shorter series, H_j of length p (we used p = 5)

$$H_j = (X_{j+1}, X_{j+2}, \ldots, X_{j+p}) - X_j \tag{9a}$$

$$= (H_j^1, H_j^2, \ldots, H_j^p) \qquad \text{for } j = 1, \ldots, n - p \tag{9b}$$

For each of the p timesteps, it is possible to build a histogram across all of the H_j time series. These binned histograms approximate the discretized version of our probability density u on a grid of size $n \times p$. We may then compute spatial and temporal derivatives for use in PDE-FIND.

When using histograms to approximate the density function it is important to choose values of n (number of bins) and p (number of time steps) that are sufficiently high to be able to accurately differentiate the density function, but not so high that we over fit the distribution. For example, in the diffusion example we expect the density at each time step to be a Gaussian. If n is too low, then we will not be able to compute spatial derivatives well but if it too high then we will not have enough data to adequately approximate the density in each bin. When choosing p, we need sufficiently many time steps to evaluate temporal derivatives but since the density function spreads out we cannot choose p too high or else we will not be able to approximate the very wide distribution that results.

2.5 Filtering noise via singular value decomposition

For some datasets we may be able to denoise via exploiting low dimensional structures in the data. The singular value decomposition [49] is utilized to discover low-energy directions in the data corresponding to additive noise. Applied to spatial-temporal datasets, this is often referred to as the proper orthogonal decomposition (POD). Modes with larger singular values correspond to recurrent structures in the data. Typically, only a few of these modes are required to reconstruct the dynamics with low error [25, 27, 43]. Principled truncations methods for denoising via the SVD for square matrices are explained in detail in [33].

Adding noise to a spatio-temporal dataset erases features corresponding to low singular values while leaving coherent structures largely unaffected. We truncate the SVD according the optimal hard threshold criterion [33]. The result is a low dimensional approximation of the noisy dataset that we assume to be less noisy than the original while maintaining all of the important dynamics. We employed an SVD filter for noise in the examples for Navier Stokes and the reaction diffusion equation. Each equation was identified from a low dimensional subspace recovered from its most important singular vectors.

3 Examples

PDE-FIND was tested on each of the examples listed below. In each case that the dynamics were specified in an Eulerian frame of reference, PDE-FIND was used both on clean data using finite differences to take derivatives, and on noisy data using polynomial interpolation. For the artificial noise, we used white noise with magnitude equal to 1% of the standard deviation of the solution function. For complex functions, both real and imaginary noise was added, each having magnitude $1/\sqrt{2}$% of the real and complex components of the function, respectively. Details on numerical methods for creating training data are also given. Where not specified, the numerical solution was obtained via spectral differentiation and a Runge-Kutta-45 ODE solver.

table S1. Summary of regression results for a wide range of canonical modes of mathematical physics. In each example, the correct model structure is identified using PDE-FIND. The spatial and temporal sampling used for the regression is given along with the error produced in the parameters of the model for both no noise and 1% noise. In the reaction-diffusion (RD) system, 0.5% noise is used. For Navier Stokes and Reaction Diffusion, the percent of data used in subsampling is also given.

PDE	Form	Error (no noise, noise)	Discretization
KdV	$u_t + 6uu_x + u_{xxx} = 0$	$1\% \pm 0.2\%, 7\% \pm 5\%$	$x \in [-30, 30], n=512, t \in [0, 20], m=201$
Burgers	$u_t + uu_x - \epsilon u_{xx} = 0$	$0.15\% \pm 0.06\%, 0.8\% \pm 0.6\%$	$x \in [-8, 8], n=256, t \in [0, 10], m=101$
Schrödinger	$iu_t + \frac{1}{2}u_{xx} - \frac{x^2}{2}u = 0$	$0.25\% \pm 0.01\%, 10\% \pm 7\%$	$x \in [-7.5, 7.5], n=512, t \in [0, 10], m=401$
NLS	$iu_t + \frac{1}{2}u_{xx} + |u|^2 u = 0$	$0.05\% \pm 0.01\%, 3\% \pm 1\%$	$x \in [-5, 5], n=512, t \in [0, \pi], m=501$
KS	$u_t + uu_x + u_{xx} + u_{xxxx} = 0$	$1.3\% \pm 1.3\%, 52\% \pm 1.4\%$	$x \in [0, 100], n=1024, t \in [0, 100], m=251$
Reaction Diffusion	$u_t = 0.1\nabla^2 u + \lambda(A)u - \omega(A)v$ $v_t = 0.1\nabla^2 v + \omega(A)u + \lambda(A)v$ $A^2 = u^2 + v^2, \omega = -\beta A^2, \lambda = 1 - A^2$	$0.02\% \pm 0.01\%, 3.8\% \pm 2.4\%$	$x, y \in [-10, 10], n=256, t \in [0, 10], m=201$ subsample 1.14%
Navier Stokes	$\omega_t + (\mathbf{u} \cdot \nabla)\omega = \frac{1}{Re}\nabla^2\omega$	$1\% \pm 0.2\%, 7\% \pm 6\%$	$x \in [0, 9], n_x=449, y \in [0, 4], n_y=199,$ $t \in [0, 30], m=151$, subsample 2.22%

Table S1 shows each example problem, error in identifying the PDE with and without noise, and the discretization used in each case. Ipython notebooks with code used for each example are available through GitHub at https://github.com/snagcliffs/PDE-FIND. Notebook names are given for each example here.

3.1 The KdV Equation

The KdV equation is an asymptotic simplification of Euler equations used to model waves in shallow water. It can also be viewed as Burgers' equation with an added dispersive term. Traveling waves in a solution to the KdV equation behave linearly when alone but exhibit nonlinear interactions. However, any solution with waves at multiple amplitudes will exhibit nonlinear behavior irregardless of interaction due to the amplitude dependance of wave speed.

3.1.1 Two soliton solution of KdV

Ipython notebook: PDE-FIND/Examples/TwoSolitonKDV.ipynb

PDE-FIND was tested on a very simple solution to the KdV equation having two non-interacting traveling waves with different amplitudes. The two soliton solution to the KdV equation was created using a spectral method with 512 spatial points and 200 timesteps. The initial condition was a superposition of functions of the form in Eq. (10) with offset centers and different amplitudes (fig. S2). Results are summarized in table S2.

table S2. Summary of PDE-FIND for identifying the KdV equation.

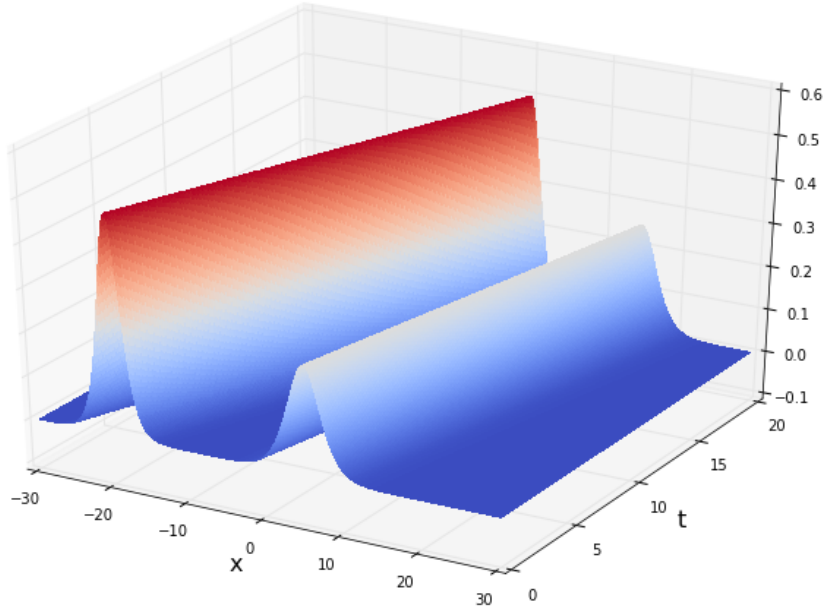| Correct PDE	$u_t + 6uu_x + u_{xxx} = 0$
Identified PDE (clean data)	$u_t + 5.956uu_x + 0.988u_{xxx} = 0$
Identified PDE (1% noise)	$u_t + 6.152uu_x + 1.124u_{xxx} = 0$

fig. S2. The numerical solution to the KdV equation plotted in space-time.

3.1.2 Disambiguating linear and nonlinear waves

Ipython notebook: PDE-FIND/Examples/KdVandAdvection.ipynb

Some of the discretized solutions studied may be solutions to more than one PDE, even within the span of the candidate functions used in PDE-FIND. An example is the single soliton solution to the KdV equation, which is a hyperbolic secant squared solution

$$u(x, t) = \frac{c}{2}\text{sech}^2\left(\frac{\sqrt{c}}{2}(x - ct - a)\right) \tag{10}$$

that travels with a speed c. However, the one-way wave equation $u_t + cu_x = 0$ generically admits solutions of the form $u = f(x - ct)$ where the function f is prescribed by the initial data. If the initial data was a hyperbolic secant squared, then the solution for both the KdV and one-way wave equation admit the same traveling wave solution. Application of our sparse regression framework would select the one-way wave equation since it is the more sparse of the two PDEs. Ultimately, a technique must be capable of disambiguating between these two PDEs. This can be done by providing different initial (amplitude) conditions. For the one-way wave equation, the waves would still travel with speed c whereas for the KdV the speed c would be amplitude dependent.

table S3. Summary of PDE-FIND for identifying Burgers' equation.

Correct PDE	$u_t + uu_x = u_{xx}$
Identified PDE (clean data)	$u_t + 1.001uu_x = 0.100u_{xx}$
Identified PDE (1% noise)	$u_t + 1.010uu_x = 0.103u_{xx}$

We constructed two solutions of this form having c equal to 1 or 5 on grids with 256 spatial points and 50 timesteps. In this case noise was not added to the solution so derivatives were taken via finite differences. When using a single traveling wave in PDE-FIND, we identified the advection equation with corresponding c. We also tried training a sparse predictor using both datasets by stacking them on top of one another in a linear system

$$\mathbf{U}_t = \begin{bmatrix} \mathbf{U}_t^{c=1} \\ \mathbf{U}_t^{c=5} \end{bmatrix} = \begin{bmatrix} \mathbf{\Theta}^{c=1} \\ \mathbf{\Theta}^{c=5} \end{bmatrix} \xi = \mathbf{\Theta}\xi$$

Any solution of this linear system must represent a nonlinear PDE, since an increase in amplitude results in a faster wave. When PDE-FIND is used for both traveling waves simultaneously we identify the KdV equation.

3.2 Burgers' equation

Ipython notebook: PDE-FIND/Examples/Burgers.ipynb

Burgers' equation is derived from the Navier Stokes equations for the velocity field by dropping the pressure gradient term. Despite it's relation to the much more complicated Navier Stokes equations, Burgers' equation does not exhibit turbulent behavior and may even be linearized through the Cole-Hopf transform [30, 34]. PDE-FIND was tested on a solution to Burgers' equation with a Gaussian initial condition, propagating into a traveling wave. Unlike many solutions to the inviscid Burgers' equation ($u_t + uu_x = 0$), the dissipative term u_{xx} prevents a shock from forming. This is important for our being able to identify the PDE since PDE-FIND relies on differentiable solutions. See fig. S3 for numerical solution. Results are summarized in table S3.

3.3 Quantum Harmonic Oscillator

Ipython notebook: PDE-FIND/Examples/QuantumHarmonicOscillator.ipynb

The quantum harmonic oscillator is Schrödinger's with a parabolic potential. The harmonic oscillator can be solved explicitly using Gauss-Hermite polynomials. It models the time evolution of the wave function associated with a particle in the parabolic potential, and at any point will give the distribution function of the particles location via taking the squared magnitude. It is important to note that measuring this function in practice is impossible as one can only observe a particles location and not it's distribution function. Further, even if some statistical distribution is gathered from many experiments, it will lack any information regarding the complex phase of the wave function. Therefore, this particular example is merely theoretical, though it does show the effectiveness of PDE-FIND for complex valued functions. The magnitude of the solution used is shown in fig. S4.

Candidate functions for the PDE were given as quadratic polynomials in u, $|u|$, and $q = x^2/2$ multiplying either a constant or a derivative of u up to the third derivative; note that in the quan-
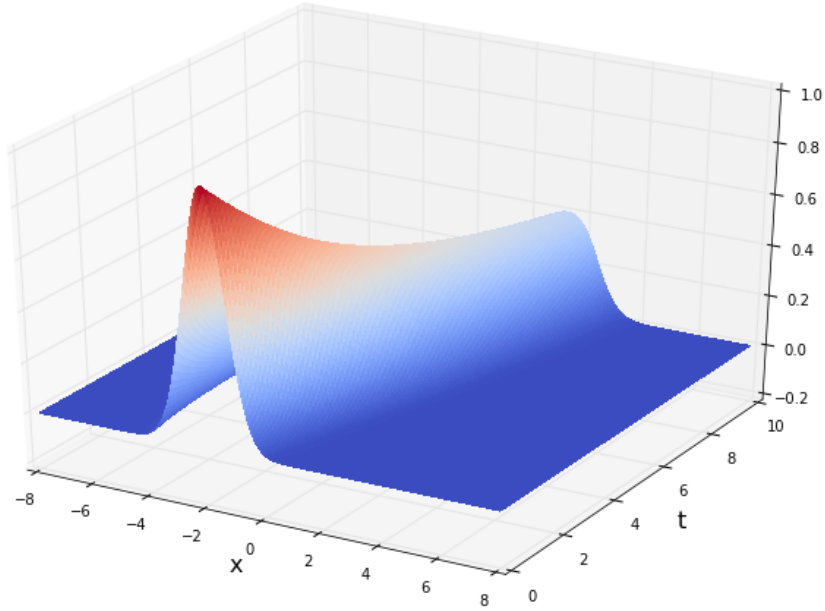
fig. S3. The numerical solution to the Burgers' equation plotted in space-time.

table S4. Summary of PDE-FIND for identifying the Schrödinger equation.

Correct PDE	$u_t = 0.5iu_{xx} - iuV$
Identified PDE (clean data)	$u_t = 0.499iu_{xx} - 0.997iuV$
Identified PDE (1% noise)	$u_t = 0.417iu_{xx} - 1.027iuV$

tum mechanics literature, the wavefunction u is often denoted ψ and the potential q is often denoted $V(x)$. Though the magnitude $|u|$ does not appear in the correct PDE, we naively assume that any complex function may follow a PDE involving it's absolute value (as in the nonlinear Schrödinger equation) and allow for our sparse regression to show that it does not. Results of the PDE-FIND algorithm are shown in table S4.

3.4 Nonlinear Schrödinger equation

Ipython notebook: PDE-FIND/Examples/NLS.ipynb

The Nonlinear Schrödinger equation is used to study nonlinear wave propagation in optical fibers and/or waveguides, Bose-Einstein condensates (BECs), and plasma waves. In optics, the nonlinear term arises from the intensity dependent index of refraction of a given material. Similarly, the nonlinear term for BECs is a result of the mean-field interactions of an interacting, N-body system. We use PDE-FIND to identify the equation from a breather solution with Gaussian initial condition. Since the function is complex valued, we consider candidate functions that have terms depending on the magnitude of the solution, $|u|$, which is of course necessary for the correct identification of the dynamics.
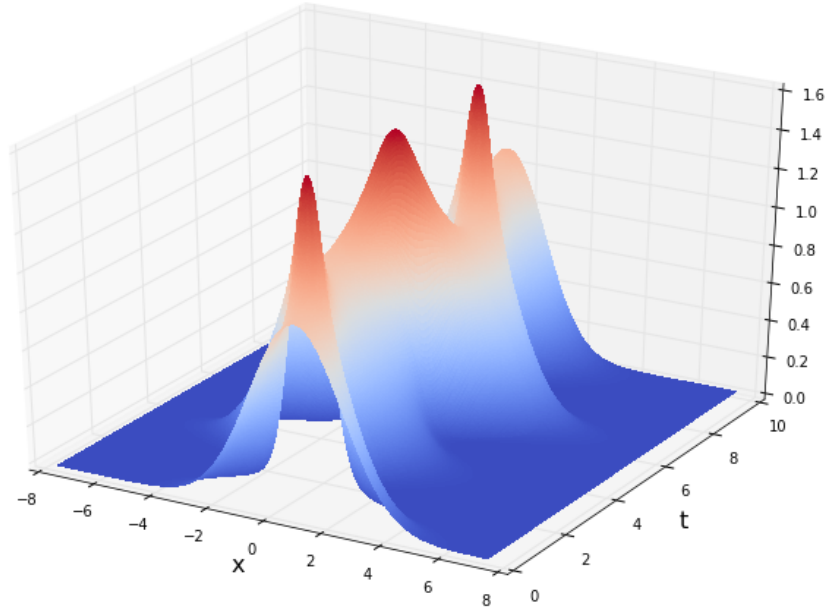
fig. S4. The magnitude of the numerical solution to the Schrödinger's equation plotted in space-time.

table S5. Summary of PDE-FIND for identifying the nonlinear Schrödinger equation.

Correct PDE	$u_t = 0.5iu_{xx} + i|u|^2u$
Identified PDE (clean data)	$u_t = 0.500iu_{xx} + 1.000i|u|^2u$
Identified PDE (1% noise)	$u_t = 0.479iu_{xx} + 0.982i|u|^2u$

3.5 Kuramoto-Sivashinsky equation

Ipython notebook: PDE-FIND/Examples/KurSiva.ipynb

The Kuramoto-Sivashinsky (KS) equation has been independently derived in the context of several extended physical systems driven far from equilibrium by intrinsic instabilities [32]. It has been posited as a model for instabilities of dissipative trapped ion modes in plasmas, laminar flame fronts, phase dynamics in reaction-diffusion systems, and fluctuations in fluid films on inclines. More broadly, it is a canonical model of a pattern forming system with spatio-temporal chaotic behavior. Like the Burgers' equation, the KS equation provides a diffusive regularization of the nonlinear wave-breaking dynamics given by $u_t + uu_x = 0$. In this case, the stabilizing regularization is accomplished by a fourth-order diffusion term since the second order diffusion corresponds to long-wavelength instabilities, i.e. it is the backwards diffusion equation, which leads to blowup. This diffusive regularization is much like the Swift-Hohenberg equation.

We simulated the Kuramoto Sivashinsky equation using a spectral method [48] with 1024 spatial gridpoints for 251 timesteps. While a courser solution sufficed for identifying the dynamics with clean data, the fine grid was needed to accurately identify the dynamics with noise. Results of the PDE-FIND algorithm are shown in Table 6. Note that though the correct terms were
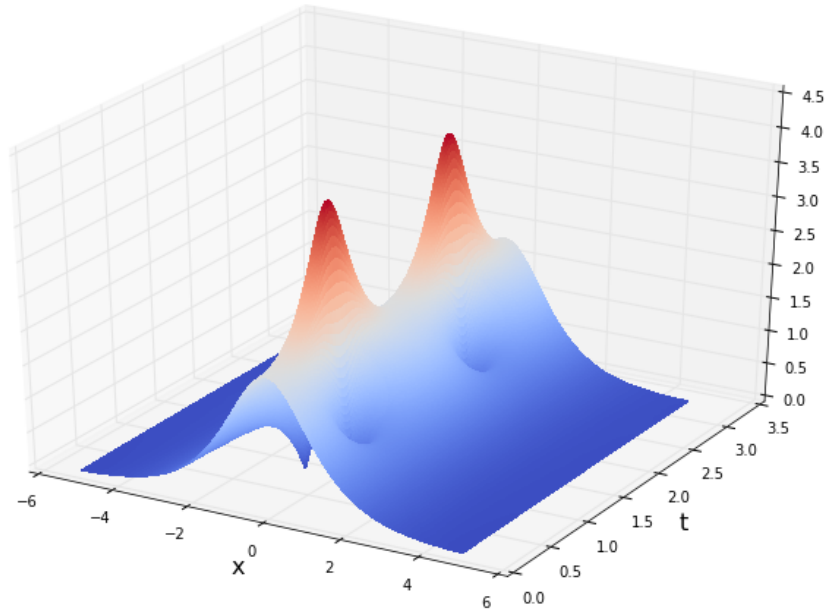
fig. S5. The magnitude of the numerical solution to the nonlinear Schrödinger's equation plotted in space-time.

table S6. Summary of PDE-FIND for identifying the Kuramoto-Sivashinksy equation.

Correct PDE	$u_t + uu_x + u_{xx} + u_{xxxx} = 0$
Identified PDE (clean data)	$u_t + 0.984uu_x + 0.994u_{xx} + 0.999u_{xxxx} = 0$
Identified PDE (1% noise)	$u_t + 0.459uu_x + 0.481u_{xx} + 0.492u_{xxxx} = 0$

identified for the PDE, parameter error for the noisy data was very high. All coefficients were underestimated by roughly 50%. For lesser amounts of noise a similar undershooting of coefficients was observed.

3.6   Reaction diffusion equation

Ipython notebook: PDE-FIND/Examples/ReactionDiffusion.ipynb

Reaction diffusion systems have been widely used in mathematical physics to study pattern forming systems [32]. The dynamics produced by reaction diffusion systems can encompass most patterns observed in nature including target patterns, spiral waves, rolls, zig-zags, etc. As such, they have been the subject of intense research over many decades. Interestingly, most reaction diffusion systems are qualitatively derived and lack connection to first principles modeling. One
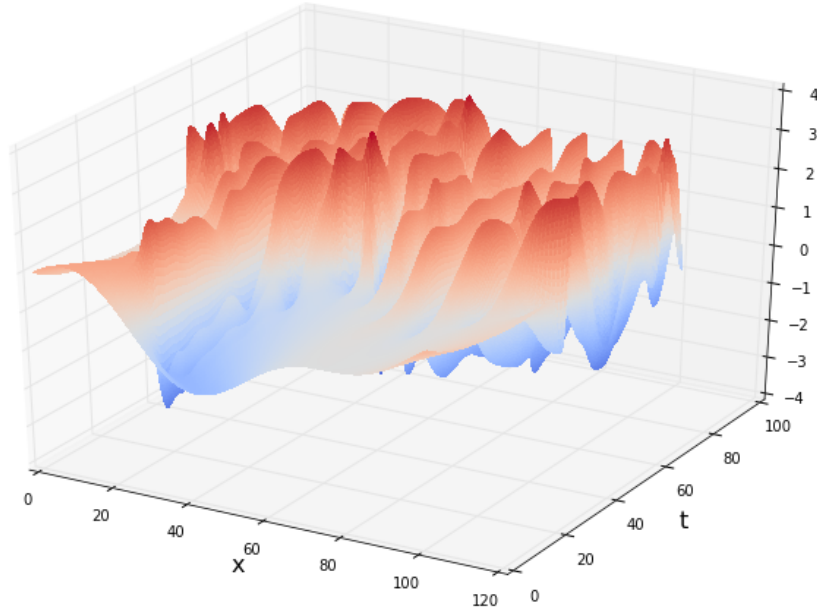
fig. S6. The numerical solution to the Kuramoto-Sivashinsky equation plotted in space-time.

class of reaction diffusion system commonly considered is the $\lambda - \omega$ system of the form

$$u_t = 0.1\nabla^2 u + \lambda(A)u - \omega(A)v \tag{11a}$$

$$v_t = 0.1\nabla^2 v + \omega(A)u + \lambda(A)v \tag{11b}$$

$$A = u^2 + v^2, \omega = -\beta A^2, \lambda = 1 - A^2 \tag{11c}$$

This particular reaction diffusion equation exhibits spiral waves on a 2D domain with periodic boundaries. To denoise the solution, we used the proper orthogonal decomposition and truncated to a lower dimensional representation based on the apparent Pareto front in the singular values. Both u and v were projected onto the 15 dimensional subspace defined by the vectors corresponding to there largest singular values.

Denoising via the SVD tends to work best when the solution is inherently low dimensional in a stationary frame, which is not the case for a traveling wave. Denoising was therefore less effective here than for a solution with more stationary features. The reaction diffusion equation was the only one of our examples for which we were unable to accurately identify the model with 1% noise and instead used 0.5%. Raising the noise to 1%, we were almost able to identify the correct PDE but the equation for U_t had an added linear dependance on v and v_t on u. In our identification of the reaction diffusion system, we subsampled 5000 spatial points and used 30 timepoints at each, resulting in 150,000 points or ~1.14% of the dataset.

3.7 Navier Stokes

Ipython notebook: PDE-FIND/Examples/Navier-Stokes with (clean data/noise).ipynb

The Navier-Stokes equations describing the two-dimensional fluid flow past a circular cylinder at Reynolds number 100 are simulated using the Immersed Boundary Projection Method

table S7. Summary of PDE-FIND for identifying reaction-diffusion equation.

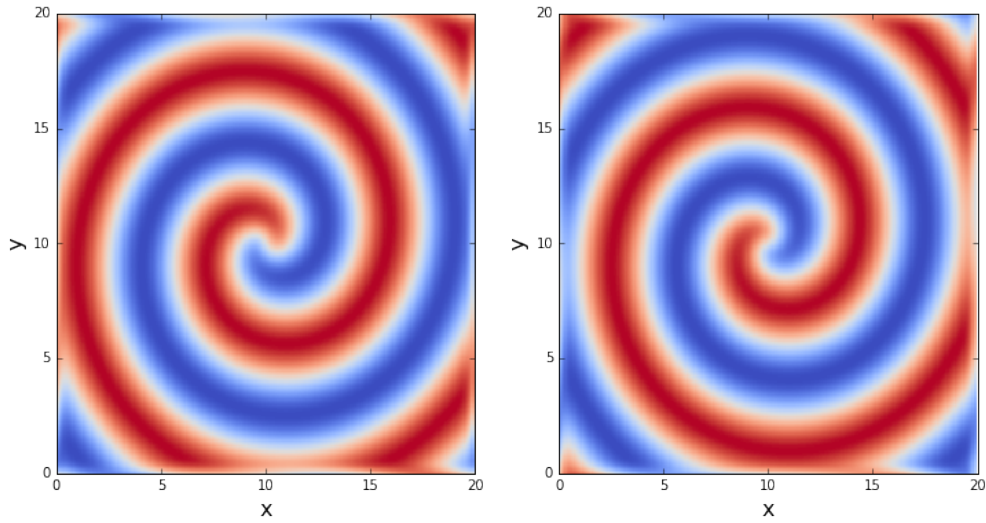| Correct PDE	$u_t = 0.1u_{xx} + 0.1u_{yy} - uv^2 - u^3 + v^3 + u^2v + u$ $v_t = 0.1v_{xx} + 0.1v_{yy} + v - uv^2 - u^3 - v^3 - u^2v$
Identified PDE (clean data)	$u_t = 0.100u_{xx} + 0.100u_{yy} - 1.000uv^2 - 1.000u^3 + 1.000v^3 + 1.000u^2v + 1.000u$ $v_t = 0.100v_{xx} + 0.100v_{yy} + 1.000v - 1.000uv^2 - 1.000u^3 - 1.000v^3 - 1.000u^2v$
Identified PDE (0.5% noise)	$u_t = 0.095u_{xx} + 0.095u_{yy} - 0.945uv^2 - 0.945u^3 + 1.000v^3 + 1.000u^2v + 0.945u$ $v_t = 0.095v_{xx} + 0.095v_{yy} + 0.946v - 1.000uv^2 - 1.000u^3 - 0.946v^3 - 0.946u^2v$

fig. S7. The numerical solution to the reaction-diffusion equation plotted in space-time.

(IBPM) [31, 46][1] . This approach utilizes a multi-domain scheme with four nested domains, each successive grid being twice as large as the previous. Length and time are nondimensionalized so that the cylinder has unit diameter and the flow has unit velocity. Data is collected on the finest domain with dimensions 9×4 at a grid resolution of 449×199. The flow solver uses a 3rd-order Runge Kutta integration scheme with a time step of $\Delta t = 0.02$, which has been verified to yield well-resolved and converged flow fields; flow snapshots are saved every $10\Delta t = 0.2$. At this Reynolds number, the flow past a cylinder is characterized by periodic laminar vortex shedding, providing a rich, yet simple prototypical dynamical system to explore high-dimensional fluid data [41].

We identified the time dependence for the vorticity of the flow field as a function of vorticity and velocity. Candidate functions were taken to be polynomial terms of the vorticity ω and the x and y coordinates of the velocity field up to second degree, multiplied by derivatives of the vorticity up to second order.

Since the data collected for Navier Stokes exhibited coherent modes which dominated the

table S8. Summary of PDE-FIND for identifying the Navier-Stokes equation.

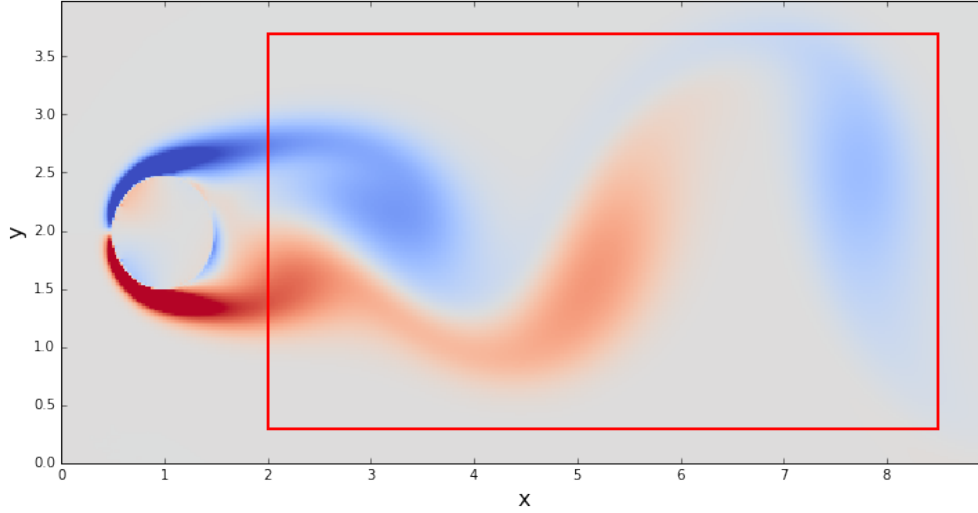| Correct PDE	$\omega_t = 0.01\omega_{xx} + 0.01\omega_{yy} - u\omega_x - v\omega_y$
Identified PDE (clean data)	$\omega_t = 0.00988\omega_{xx} + 0.00990\omega_{yy} - 0.990u\omega_x - 0.987v\omega_y$
Identified PDE (1% noise)	$\omega_t = 0.0107\omega_{xx} + 0.0083\omega_{yy} - 0.988u\omega_x - 0.983v\omega_y$

fig. S8. A single snapshot of the vorticity field is illustrated for the fluid flow past a cylinder. The sampling region is outlined in red.

long term behavior of the solution, the singular values of the dataset fell off rapidly, indicating that the majority of the behavior could be characterized with only a few dominant modes. This was especially useful in the case of noisy data since we were able to use the SVD to denoise the data more effectively than for the reaction diffusion system.

Like for the reaction diffusion problem, we also subsampled for Navier Stokes. The dataset we used for the Navier Stokes equations consisted of roughly 13.5 million datapoints. Constructing a data matrix for this many points across an overcomplete library of candidate functions could quickly become intractable, both for the time required to take derivatives as well as the space needed to store and work with that large of a matrix. We sampled 5000 spatial locations within a region of the domain downstream from the cylinder (see fig. S8) and collected the data and derivatives at these points for 60 different times, resulting in a matrix with 2.22% of the rows that would be present in the full dataset.

3.8 Diffusion from a random walk

Ipython notebook: PDE-FIND/Examples/DiffusionFromRandomWalk.ipynb

PDE-FIND was used to identify the long celebrated relation between Brownian motion and the diffusion equation. The Fokker-Planck equation associated with a particle's position, for Brownian motion where $x(t + dt) \sim \mathcal{N}(x(t), dt)$ is $u_t = 0.5u_{xx}$. Using a single trajectory of Brownian motion we are able to consistently derive the diffusion equation from a small number of observations.
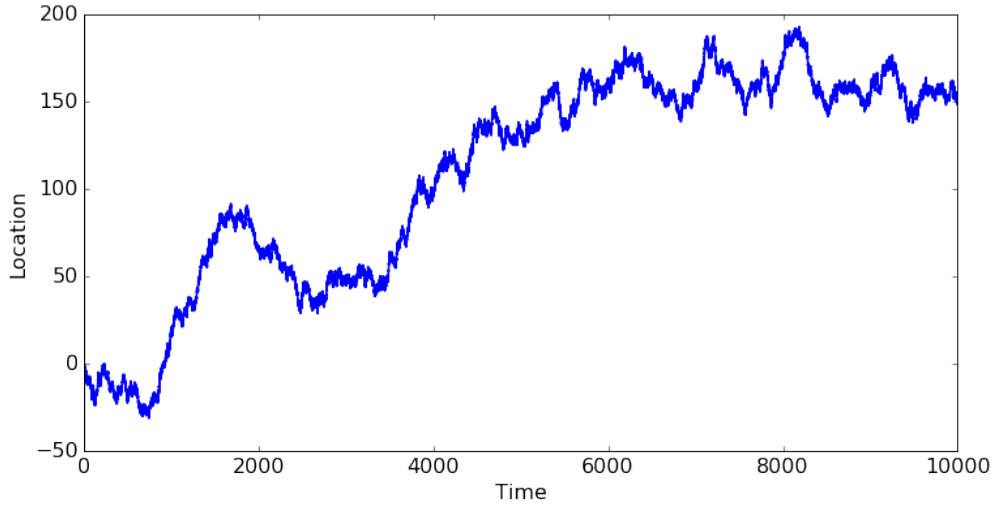
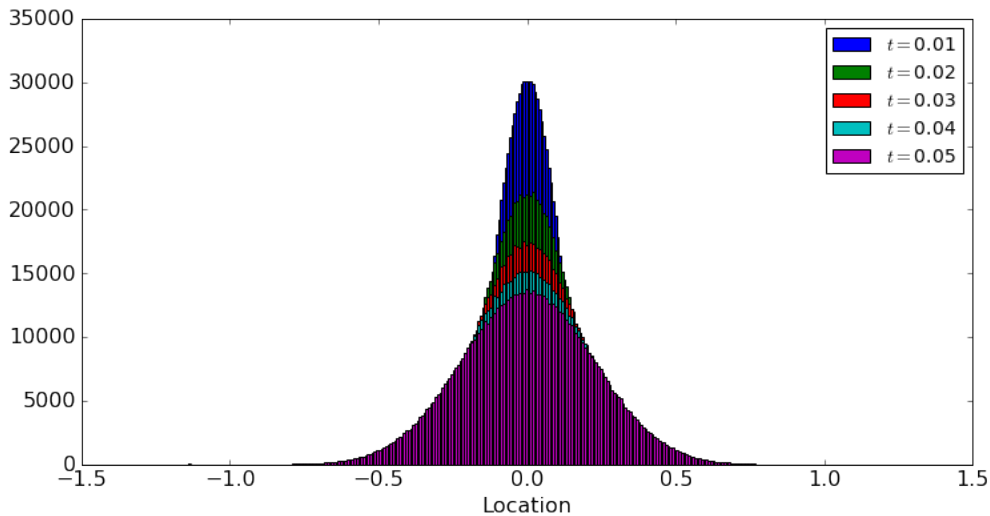fig. S9. A single stochastic realization of Brownian motion.

fig. S10. Five empirical distributions , illustrating the statistical spread of a particle's expected location over time, are presented. The data used to construct these histograms is collected from a single Brownian motion trajectory, see fig. S9.

We simulated Brownian motion at evenly space time points by adding a normally distributed random variable with variance dt to the time series. An example is shown in fig. S9. Histograms of the particles displacement were taken using a number of bins that balanced resolution without overfitting the amount of data. Small changes in the number of bins, especially for longer time series, did not have a large effect on the result of PDE-FIND. Figure S10 shows an example of the histograms used to approximate the distribution function for a time series of length 10^6.

Since the approximate distribution function created via histograms was inherently noisy, we used polynomial interpolation to differentiate in the spatial dimension. However, since there were very few timesteps, we were unable to differentiate accurately with respect to time using

polynomial interpolation and instead used finite differences for the time derivatives.

Figure 2(c) in the main text shows the average ℓ^1 parameter error ($\|\xi - \xi^*\|_1$) in identifying the diffusion equation across 10 trials for various lengths of time series. For very short time series PDE-FIND had a tendency to incorrectly identify the sparsity pattern of the PDE. In many of these cases, the coefficients in the right hand side were very large. For example, in one misidentification using a time series of length 681, the identified PDE was

$$u_t = -2776u^2 + 753361u^2u_x + 44531u^2u_{xx}$$

For shorter time series in which the sparsity was correctly identified, the error was often from PDE-FIND underestimating the value of the coefficient for diffusion. This results in an $\mathcal{O}(1)$ error. As a result the parameter error averaged over many trials was either very large (when PDE was misidentified) or no larger than $\mathcal{O}(1)$. This is illustrated in Fig. 2(c) where we see unpredictable error to a point before the PDE is consistently identified, then decreasing error as PDE-FIND is better able to identify the diffusion coefficient. For longer time series, error was closer to $\mathcal{O}(10^{-3})$.

PDE-FIND was also used to identify the Fokker-Planck equation for Brownian motion with a bias. That is $x(t+dt) \sim \mathcal{N}(x(t)+cdt, dt)$ A time series was generated by adding Gaussian random variables at each timestep along with a drift term cdt with $c = 2$. This is the same as taking

$$x_{n+1} \sim \mathcal{N}(x_n + cdt, dt) \qquad c = 2$$

Histograms used for approximating the distribution function are shown in fig. S11. The Fokker-Planck equation for the distribution function is

$$u_t + cu_x = 0.5u_{xx}$$

Normally, the best results in the PDE-FIND algorithm were obtained using sequential threshold ridge regression with columns of Θ normalized to have unit variance, however, that approach was the worst in the case of identifying the advection diffusion equation from a biased random walk, failing to identify the advection term. Sequential threshold ridge without normalization, LASSO, and the forward-backward greedy algorithm all correctly identified the PDE.

$u_t = 0.500872u_{xx}$ STRidge with normalization

$u_t = -2.000641u_x + 0.503582u_{xx}$ STRidge without normalization, LASSO, and Greedy

4 Limitations

4.1 PDE-FIND with an incomplete library

In applying the PDE-FIND algorithm we assume that the column space of Θ is sufficiently rich to have a sparse representation of the time dynamics of the dataset. However, when applying the algorithm to a data set where the dynamics are in fact unknown it is not unlikely that the column space of Θ is insufficient. We cannot provide a comprehensive analysis of what the PDE-FIND algorithm will converge to but will provide two examples of where this is the case.

4.1.1 Kuramoto-Sivashinsky equation with incomplete library

The Kuramoto-Sivashinsky equation involves a fourth spatial derivative, which one may not guess to allow when looking for models to fit the observed dynamics. We tried identifying an
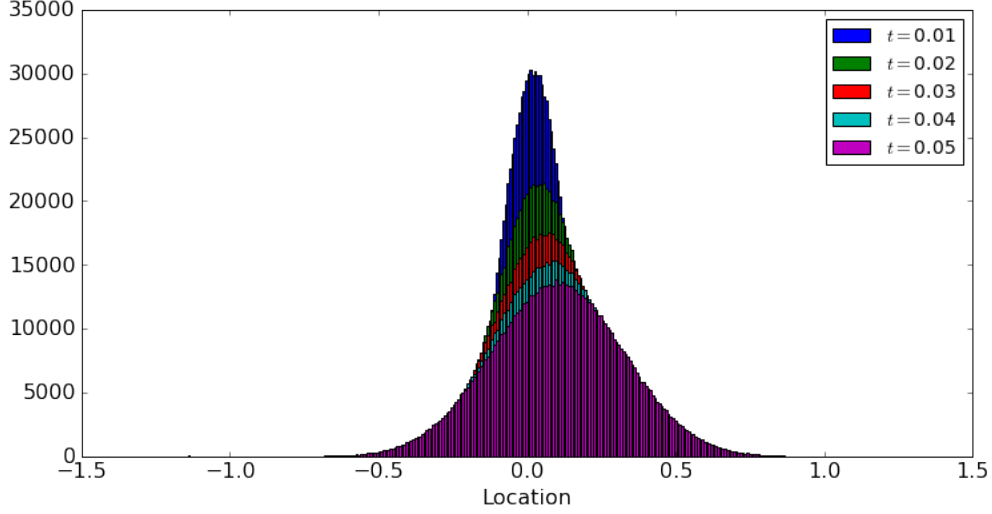
fig. S11. Five empirical distributions , illustrating the statistical spread of a particle's expected location over time, are presented. The data used to construct these histograms is collected from a single biased Brownian motion trajectory.

equation using the same numerical solution to the Kuramoto-Sivashinsky equation but not allowing for fourth order derivatives. The result was that the u_{xxxx} was replaced with a uu_{xxx} term. Each of the other two terms was correctly identified but with large coefficient error.

$$u_t = -0.189uu_x + 0.108u_{xx} - 0.076uu_{xxx}$$

While uu_{xxx} was positively correlated with u_{xxxx} on the training data the dynamics are radically different. A numerical solution to the misidentified PDE is shown in fig. S12.

4.1.2 Nonlinear Schrödinger equation with incomplete library

The Nonlinear Schrödinger equation relates the temporal derivative of u not only to u and its spatial derivatives but also to $|u|$. We tried fitting the solution to the NLS equation to a library of candidate functions that does not contain any power of $|u|$. Unlike the Kuramoto-Sivashinsky example, the result did not appear related to the true equation. The resulting PDE is

$$
\begin{aligned}
u_t =\,&(0.368545 - 0.386346i) + (-0.000381 + 0.100149i)u_{xx} + (-0.233212 - 0.242627i)u^2 \\
&+ (0.066812 - 0.000737i)u^3 + (-0.001855 + 3.796550i)u + (0.024974 - 0.000055i)u^2 u_{xx} \\
&+ (0.001612 - 0.001612i)u^3 u_{xx} + (0.059966 + 0.059281i)uu_{xx}
\end{aligned}
$$

A numerical solution obtained via spectral differentiation and ODE-45 to the misidentified PDE for the NLS data is shown in fig. S13.

4.2 Higher levels of noise

Identifying governing equations with noisy datasets is made difficult due to the challenge in numerically differentiating noisy data. We have provided examples of this for each of the canonical
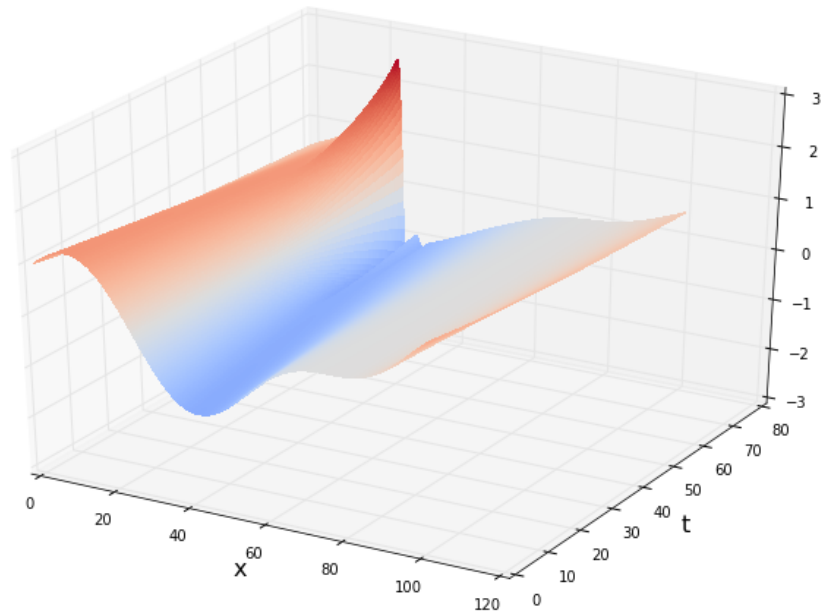
fig. S12. The numerical solution to the misidentified Kuramoto-Sivashinsky equation.
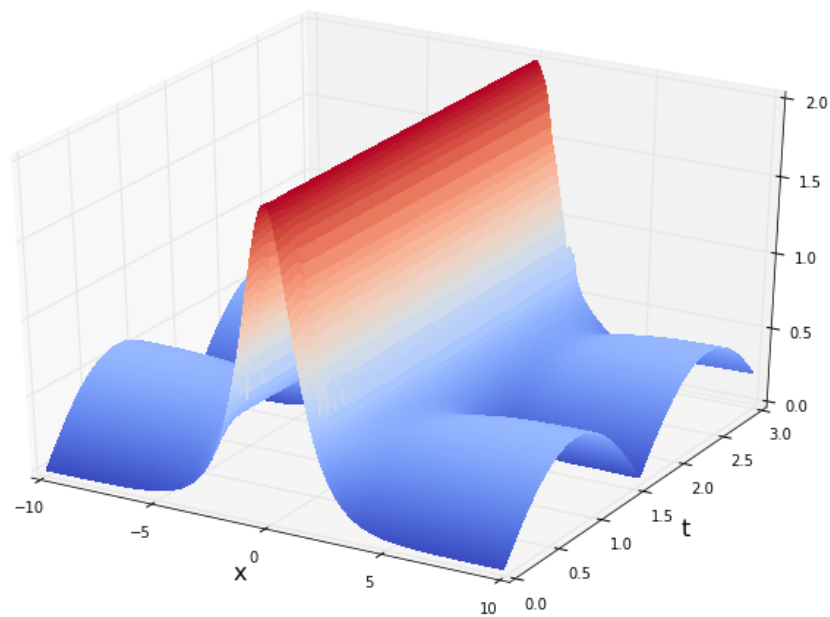

fig. S13. The numerical solution to the misidentified nonlinear Schrödinger equation.

PDEs discussed in the paper but did not exceed 1% noise. Figure S14 shows the error in identifying Burgers' equation with noise levels of up to 12%. In this case, PDE-FIND is able to correctly identify the correct terms in the PDE with modestly increasing error until the noise level reaches

	Candidate Functions															
	1	u	u^2	u^3	u_x	uu_x	u^2u_x	u^3u_x	u_{xx}	uu_{xx}	u^2u_{xx}	u^3u_{xx}	u_{xxx}	uu_{xxx}	u^2u_{xxx}	u^3u_{xxx}
Clean						-0.986			0.119							
1%						-0.986			0.119							
2%						-0.986			0.118							
3%						-0.986			0.118							
4%						-0.985			0.118							
5%						-0.984			0.117							
6%						-0.983			0.117							
7%						-0.982			0.116							
8%						-0.980			0.115							
9%						-0.978			0.114							
10%				0.168		-1.221			0.076	0.237					-0.090	
11%						-1.127			0.086	0.115					-0.073	
12%	0.001	0.054	-0.239	0.319	-0.064	-0.950	0.190	-0.449	0.035	0.591	-1.179	0.942	-0.006	0.081	-0.300	0.178

(Row label, vertical: Noise Level)

fig. S14. Results of PDE-FIND applied to Burgers' equation for varying levels of noise.

12%, at which point we see extra terms added to the identified model. At 12%, every candidate function is included in the identified dynamics.

The reaction diffusion equation was much less robust to noise. Even at 1%, PDE-FIND misidentifies the terms in the PDE. The true model, as well as identified models with clean data, 0.5% noise, and 1% noise are given below. Note that coefficients for clean data are inexact but have been rounded to three decimal points.

$$\left.\begin{aligned} u_t &= 0.1u_{xx} + 0.1u_{yy} - uv^2 - u^3 + v^3 + u^2v + u \\ v_t &= 0.1v_{xx} + 0.1v_{yy} - uv^2 - u^3 - v^3 - u^2v + v \end{aligned}\right\} \text{True model}$$

$$\left.\begin{aligned} u_t &= 0.100u_{xx} + 0.100u_{yy} - 1.000uv^2 - 1.000u^3 + 1.000v^3 + 1.000u^2v + 1.000u \\ v_t &= 0.100v_{xx} + 0.100v_{yy} - 1.000uv^2 - 1.000u^3 - 1.000v^3 - 1.000u^2v + 1.000v \end{aligned}\right\} \text{Clean Data}$$

$$\left.\begin{aligned} u_t &= 0.095u_{xx} + 0.095u_{yy} - 0.945uv^2 - 0.945u^3 + 1.000v^3 + 1.000u^2v + 0.945u \\ v_t &= 0.095v_{xx} + 0.095v_{yy} - 1.000uv^2 - 1.000u^3 - 0.946v^3 - 0.946u^2v + 0.946v \end{aligned}\right\} \text{0.5% Noise}$$

$$\left.\begin{aligned} u_t &= 0.077u_{xx} + 0.077u_{yy} - 0.731uv^2 - 0.732u^3 + 0.810v^3 + 0.810u^2v + 0.776u + 0.169v \\ v_t &= 0.081v_{xx} + 0.079v_{yy} - 0.832uv^2 - 0.832u^3 - 0.772v^3 - 0.772u^2v + 0.815v - 0.149u \end{aligned}\right\} \text{1% Noise}$$

4.3 Limited Data

In each of the examples provided, we have far more data points than library functions. The importance of using a large number of points lies more in the numerical evaluation of derivatives than in supplying sufficient data for the regression. In table 9 we test the PDE-FIND method on a number of discretizations of a solution to Burgers' equation. An initial solution was computed on a fine grid and data points from the fine solution were taken on courser grids to evaluate the performance of the method with courser sampling. As expected, we notice a decline in accuracy with courser grids and eventually the inability to accurately identify the true sparsity pattern of the coefficient vector. We should note though that this is not due to a lack of points in the regres-

	Temporal Points: m				
	256	128	64	32	16
512	0.113	0.153	0.896	2.446	
256	0.777	0.509	0.238		
128	3.417	3.140		1.161	
64	13.72				
32					

(Spatial Points: n)

table S9. Accuracy of PDE-FIND on Burger's equation with various grid sizes. Red table entries denote a misidentification of the sparisty pattern either due to the inclusion of extra terms or missing one of the two terms in Burgers' equation. Blue entries show average parameter error as percent of true value. Measurements on all grids were taken from numerical solution on fine grid to ensure error in the method is intrinsic to PDE-FIND and not the numerical solution of Burgers' equation.

sion. Using the original grid (1024 x 512) to evaluate derivatives we were able accurately identify the PDE using just 10 points with 0.023% error in the coefficients.