Hyperparameter optimization in the estimation of PDE and delay-PDE models from data

IOP Publishing New J. Phys. 28 (2026) 013701 https://doi.org/10.1088/1367-2630/ae2e33

New Journal of Physics
The open access journal at the forefront of physics

Deutsche Physikalische Gesellschaft DPG
IOP Institute of Physics

Published in partnership with: Deutsche Physikalische Gesellschaft and the Institute of Physics

PAPER

Hyperparameter optimization in the estimation of PDE and delay-PDE models from data

Oliver Mai[1,*], Tim W Kroll[1], Uwe Thiele[1,2,3] and Oliver Kamps[3]

[1] Institute of Theoretical Physics, University of Münster, Wilhelm-Klemm-Str. 9, 48149 Münster, Germany
[2] Center for Data Science and Complexity (CDSC), University of Münster, Corrensstr. 2, 48149 Münster, Germany
[3] Center for Multiscale Theory and Computation (CMTC), University of Münster, Corrensstr. 40, 48149 Münster, Germany
[*] Author to whom any correspondence should be addressed.

E-mail: oliver.mai@uni-muenster.de

Keywords: data-driven modeling, dynamical systems, partial differential equations, delay equations

OPEN ACCESS

RECEIVED
22 August 2025

REVISED
28 November 2025

ACCEPTED FOR PUBLICATION
17 December 2025

PUBLISHED
13 January 2026

Abstract

We propose an improved method for estimating partial differential equations (PDEs) and delay PDEs from data, using Bayesian optimization and the Bayesian information criterion to automatically find suitable hyperparameters for the method itself and also for the equations (such as a time-delay). We show that combining time integration into an established model estimation method increases robustness and yields predictive models. Allowing hyperparameters to be optimized as part of the model estimation results in a wider modeling scope. We demonstrate the method's performance for a number of synthetic benchmark problems of different complexity, representing different classes of physical behavior. This includes the Allen–Cahn and Cahn–Hilliard models, as well as different reaction-diffusion systems without and with time-delay.

1. Introduction

The recent advancements in data collection and storage, as well as the rising availability of computational resources, have facilitated data-driven learning methods and impacted modern science in many ways. As large-scale experiments and computer simulations continue to produce complex data sets, typical methods for discovering governing physical laws and translating them into mathematical models, such as first-principle derivations, take more time and effort to complete or may not account for all the available data. Thus over recent years computer-aided discovery of dynamical equations has attracted more and more attention.

During their inception many symbolic data-driven modeling methods focused on ordinary differential equations (ODEs). As a results there are numerous such methods available, e.g. the multiple shooting method [1], sensitivity analysis [2] or generalized Gauss–Newton methods [3]. Then, [4] proposed to reframe an inverse problem in terms of a least-square problem with a suitable set of candidate model functions. This spawned various alterations and in recent developments the combination of various different methods. This includes combining de-noising and data preparation into the learning method [5], and incorporating methods that allow for domain-specific user input to cope with sparsely sampled data [6].

However, for many (especially spatially extended) systems a description using ODEs is insufficient and partial differential equations (PDEs) are needed. Although, in principle, methods originally designed for ODEs can readily be adapted for use with PDEs, often new challenges arise regarding scalability and performance. Attempts to specifically identify PDEs from data started in the late 1990s [7–10] and have seen various improvements and modifications over the years. Typically, just as for ODEs, a model structure is either proposed by prior knowledge about the system or expressed by a set of candidate functions, which are then narrowed down using, e.g. sparsity-promoting algorithms [4, 11–13], sensitivity-analysis [14, 15] and information criteria [16, 17]. The models are optimized to either minimize the residual of the PDE [4, 11, 18] or its predictive error [19, 20]. While there have been variations and

alternative methods, where the system is described via a weak-formulation [21, 22], methods which allow for additional complexity such as time-delayed variables [23, 24] remain few and narrow in scope.

A parallel development is found in physics-informed neural networks [25] and related approaches. There, knowledge about the underlying physical systems is incorporated to subsidize otherwise black-box approaches to predict system behavior or to recover specific system parameters, but not to generate a comprehensive mathematical description of the provided data.

For many of the model learning methods the sole benchmark remains the rediscovery of physical laws based on numerical simulations. The performance on these benchmarks has made significant progress for both ODEs and PDEs. While this facilitates the use of these methods for real world applications, practical application to study open problems still faces many challenges. Examples include microscopy data [26], gene expression [23] and monitoring of motors [27] or tokamak discharge [28], but poor knowledge of involved parameters and overfitting onto noise still pose significant hurdles. The model quality is often only evaluated after the learning method has concluded [17]. This results in models that are numerically unstable during time simulations or fail to replicate initial data at all, without additional clean-up or corrections using prior knowledge.

In many data-driven model estimation methods, hyperparameters that govern model complexity, regularization, data preprocessing or optimization strategies have a vital influence on the performance of the learning algorithms [29, 30]. These may gauge overfitting or set learning rates or thresholds during optimization. Often these hyperparameters are not learned from data, but instead set beforehand and either adjusted via a trial-and-error approach or via computationally expensive systematic grid-based or random searches.

In [31] the REBEL (Reconstruction Error Based Estimation of dynamical Laws) method, a novel method for identifying sparse ODE systems from data, is introduced. It combines the efficient sparse parameter estimation approach of [4] with an error estimate based on a combination of the integrated least squares error and the Wasserstein metric. Integrated within a Bayesian optimization framework, this method efficiently determines optimal hyperparameters, leading to a better data approximation with fewer parameters. Our goal is to leverage this approach for robust and efficient sparse PDE model estimation from data. The hyperparameter optimization strategy in [31] also enables us to efficiently estimate delay PDEs, thereby significantly expanding the applicability of the method.

In section 2 we develop our numerical method. Then, in section 3.1.1 we show that it yields virtually identical results for the elementary example analyzed in [11]. The same holds true in section 3.1.2, where we compare resilience to noise and low spatial sample rate. To further investigate the comparative performance, we vary the sampling frequency in section 3.1.3 and show that our approach is more robust than the one in [11] with respect to data that are under-sampled in time. Then, in section 3.2 we consider phase-field models with and without mass-conservation and show that further restrictions are not needed to achieve mass-conservation and that our implementation allows more flexibility for model ansatz functions. Finally, section 3.3 showcases how the method accommodates multiple hyperparameters, such as multiple method thresholds and time-delays, which enables us to treat models of higher complexity. Section 4 summarizes our findings and proposes further fields of study. The appendix houses more detailed information on creating a default set of ansatz functions in appendix A, a pseudocode representation of the model estimation procedure in appendix B, an in-depth numerical tutorial in appendix C and unabbreviated numerical results for all examples in appendix D.

2. Numerical methods

2.1. Optimization framework

The symbolic approach to estimating dynamic equations in the form of PDEs from data relies on narrowing down suitable functions for the right hand sides of the equations from a set of ansatz- or library-functions. In its simplest form this means taking time series data and then fitting the library-functions to the time derivative of the data. In practice, when using a least-squares fit, this often results in extensive and unwieldy expressions, where most of the functions only give small contributions. The aim is to single out only major contributions and to produce well-established and/or interpretable models.

We assume that there exists a functional expression F, which describes the true dynamics of the time evolution of a physical quantity $u(x,t) \in \mathbb{R}^n$ (where n is the number of field variables) we wish to model:

$$\frac{\partial u}{\partial t} = F(u)\,. \tag{1}$$

Here, t is the time and $x \in \mathbb{R}^d, d \in \{1,2,3\}$ is the position in d-dimensional space. $F(u)$ may include nonlinearities and spatial derivatives of u and in principle any function that contributes to the dynamics of u. We assume that each individual contribution in F (however complex it may be) can be combined linearly, so F could look as follows:

$$\frac{\partial u}{\partial t} = au + bu^2 + c\nabla u + d\sin(t) + \ldots, \tag{2}$$

where a, b, c and d are coefficients. Now, for an unknown system it may be unclear which exact functions appear on the right-hand side of equation (2), but many scientific disciplines hold broad catalogs of modeling approaches. As such we assemble a large number of ansatz- or library-functions that could capture the dynamics of an unknown system we wish to model (see appendix A for more details on library creation). We collect all of the library-functions in a vector Θ (with k functions) and all of their coefficients in a matrix $\sigma \in \mathbb{R}^{n \times k}$, so that we can rewrite equation (2) as the linear matrix multiplication

$$\frac{\partial u}{\partial t} = \sigma \cdot \Theta(u). \tag{3}$$

Of course, if the true dynamics falls outside of the space spanned by our library functions, then it can only be approximated.

The goal is now to find a sparse set of values for σ that best describes the time evolution of u. To that end, we employ a regression method. See figure 1 for an overview of the entire optimization framework. So we search for an approximate set of coefficients, that has a residual error with respect to the true system:

$$R(\sigma) = ||\partial_t u - \sigma \cdot \Theta(u)||_2. \tag{4}$$

Here,

$$||z||_2 = \left(\sum_{r=1}^{N_t} \sum_{l=1}^{N^2} |z_{rl}|^2 \right)^{\frac{1}{2}} \tag{5}$$

is the ℓ^2-norm for N_t number of samples in time and N^2 sample points in space. Now we want to find the set $\hat{\sigma}$, that minimizes R. In other words, we wish to solve the optimization problem

$$\hat{\sigma} = \arg\min_{\sigma} \; R(\sigma) = \arg\min_{\sigma} \; ||\partial_t u - \sigma \cdot \Theta(u)||_2. \tag{6}$$

Since $\sigma \cdot \Theta$ is a linear matrix multiplication, this problem can be numerically solved by any least-squares routine.

This leaves two problems. First, as previously mentioned, a least-square routine by itself typically yields non-sparse results. Second, the residual in equation (4) is formulated with respect to the time derivative of the original data and as shown in [31] the minimum of such a residual is not identical to the one using the deviation from the integrated estimate to the original time series:

$$L(\hat{\sigma}) = \left|\left| u - \int_0^{\mathrm{T}} \hat{\sigma} \cdot \Theta(u)\, \mathrm{d}t \right|\right|_2 = ||u - \hat{u}||_2. \tag{7}$$

The use of numerical time integration (from $t = 0$ to the final time $t = T$) means, that a reconstructed time series $\hat{u}$, now only implicitly depends on our coefficients $\hat{\sigma}$.

To overcome both of these problems we employ the same strategy as [31], i.e. we make use of the Bayesian information criterion (BIC), which takes the maximized likelihood $\tilde{L}$ of our estimated model (which is equivalent to minimizing equation (7)) and penalizes it with the number of model parameters:

$$\mathrm{BIC} = s\ln(N_t) - 2\ln\left(\tilde{L}\right). \tag{8}$$

Here s is the number of non-zero coefficients in $\hat{\sigma}$. When $\hat{\sigma}$ is close (or equal) to the true system parameters, $L(\hat{\sigma})$ should also be minimal, thus $L(\hat{\sigma}) = \tilde{L}$. This results in the scheme shown in figure 1:

1. Load time series data of $u(x,t)$ that shall be modeled.

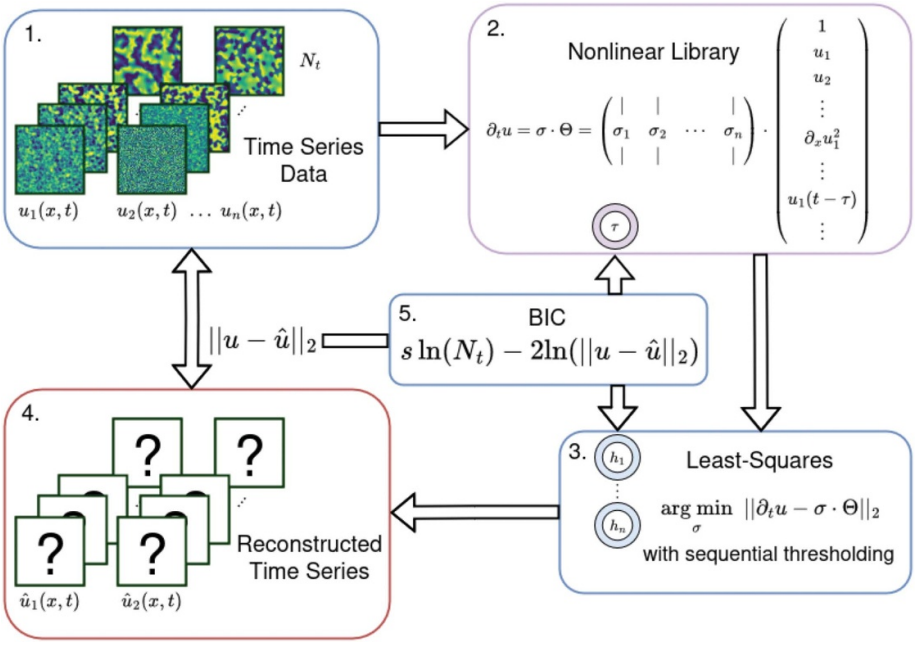

Figure 1. Scheme of the model estimation procedure. The steps are: 1. load time series data; 2. compute suitable library functions; 3. use least-squares-based sparse regression method to estimate model parameters; 4. integrate estimated model in time; 5. compute BIC and repeat steps 2 (if library terms have changed), 3 and 4.

2. Construct the library Θ of nonlinear ansatz functions which may contain hyperparameters such as a time delay τ.
3. Perform a least-squares fit to find estimated system parameters $\hat{\sigma}$ and sequential thresholding with hyperparameters $h_1, \ldots, h_n$, where contributions of $\hat{\sigma}_i$ to the evolution of variable u_i below the threshold h_i are subsequently discarded until sparsity no longer increases.
4. Perform time integration using the estimated parameters $\hat{\sigma}$ and library Θ.
5. Use the deviation from the original time series and the BIC to optimize the hyperparameters and repeat steps 2 (if any library term has changed due to a hyperparameter), 3 and 4.

In other words, we begin by using a sparsity promoting least-squares routine to find an initial $\hat{\sigma}$, which minimizes the residual (4) with respect to the time derivative, then compute L using $\hat{\sigma}$ (and any additional model parameters) by integrating our model in time and finally use another numerical optimization procedure to iterate previous steps to minimize the BIC. This yields optimized method hyperparameters, such as any thresholds h in this sparsity promoting variant of the least-squares routine, and refined model parameters. Additionally, we may add any other hyperparameter to be optimized, such as ones that change our ansatz functions (e.g. a time delay τ or a phase shift φ). For more details on the entire optimization routine see again figure 1 and section 2.2.

2.2. Estimation procedure

Even for a large library $\Theta(u)$, problem (3) remains a linear matrix equation. Thus we initially estimate parameters $\hat{\sigma}$ using the least-squares routine from the popular computing package *Numpy* [32], minimizing the difference to our time derivative:

$$\hat{\sigma} = \arg\min_{\sigma} \; ||\partial_t u - \sigma \cdot \Theta(u)||_2.$$

The time derivative of the original time series data $\partial_t u$ is computed using finite differences. Spatial derivatives during library evaluation can be computed via FFT or finite differences (supplied by the package *findiff* [33]). Our routine uses the latter per default. The accuracy of these derivatives can greatly impact the estimation performance, albeit with diminishing returns: Falling below a certain accuracy will result in a worse estimation performance depending on the problem at hand. Higher accuracy, however, will not necessarily increase the estimation performance. The accuracy of the finite differences method can be freely adjusted. The default is second order accuracy.

Sparsity of $\hat{\sigma}$ is then promoted using sequential thresholding least-squares (STLS, [4]), i.e. parameters with a magnitude below a threshold value h are set to zero and excluded, then the least-square fit is

repeated until either sparsity no longer increases or a maximum number of iterations is reached. The threshold h constitutes a hyperparameter that greatly influences the resulting model. In [11] the STLS algorithm is used with one identical threshold value for all parameters. In contrast, here, we allow for a threshold h_i per system variable u_i. Additionally, one may want to group thresholds by the influence of certain groups of library terms. For example when adding time-delayed library terms, we add a threshold h_τ, for those terms.

To find these threshold hyperparameters we use the optimization package *Hyperopt* [34] and its tree-structured Parzen estimator (TPE) [35], which is commonly used for hyperparameter tuning in machine learning. The TPE is a Bayesian optimization method that models a given objective function probabilistically. It compares the ratio of two density estimators to focus on promising regions of the search space. We set the objective function of the TPE to be the BIC (8). This requires the computation of equation (7) and therefore $\hat{u}$, which we obtain by integrating the model in time. This time integration makes use of the initial value problem solver of *SciPy* [36] and its explicit Runge–Kutta method of order 8 [37], although any method from SciPy may be selected (e.g. an implicit method). Such time integration can be a difficult task, however, since in the process of finding a feasible PDE description, in some cases, candidate PDEs do not guarantee numerical stability. Hence we utilize a trapezoidal integration scheme (also the one implemented in SciPy [36]) as a fallback method when integration fails, or to reduce computation time. While more stable or higher-order integrators may enable successful estimation where simpler methods fail, they introduce increased computation time and may not necessarily improve estimation performance otherwise.

Additionally we may feed any other hyperparameter that defines our ansatz library into the TPE. One example is the later described time-delay τ for certain terms. In these cases values of the functions in Θ change, so they are recomputed during the optimization if needed. Again, for an overview of the estimation routine see figure 1 or the pseudo-code representation in algorithm 1 in appendix B.

3. Results

All results in this section (if not otherwise stated) have been achieved using the Python package *TSME* ('time series model estimation', software publication pending, preliminary release in [38]). All relevant data and Python code for the examples given here can be found in the GitHub repository in [39].

This section investigates several synthetic inverse problems to illustrate the performance of the developed method. First, we consider a symmetric two-field reaction-diffusion model equivalent to the complex Ginzburg–Landau (cGL) equation, which was used as a benchmark in [11]. In particular, the goodness of fit is investigated when varying the sample frequency in space and time or with added noise. As the software package *PySINDy* [40] that originally implemented the method proposed in [4] was later extended for use with PDEs via the method introduced in [11], that implementation will be used as a reference. The second case is the Cahn–Hilliard (CH) equation, a model with mass conservation for which corresponding state-of-the-art software packages are to our knowledge not yet applicable, due to their restrictive ansatz library. The final three examples focus on the application of multiple hyperparameters. The FitzHugh–Nagumo (FHN) reaction-diffusion model and the chaotic regime of the cGL equation are employed to illustrate how incorporating semantic information into the choice of multiple thresholds may lead to improved results. Finally we show for the Fischer–Kolmogorov–Petrovsky–Piskunov (Fisher-KPP) equation with added time delay how the method can incorporate additional hyperparameters, aside from thresholds. While all examples are based on synthetic data, they serve as proof-of-concept and show that for cases where underlying equations are unknown the inclusion of additional hyperparameters strongly improves model estimation abilities.

3.1. Dependence of performance on data quality

3.1.1. Baseline performance

As in [11] we generate synthetic data for a reaction-diffusion model with third-order nonlinearities corresponding to a cGL equation, namely:

$$\begin{aligned} \partial_t u &= D_u \nabla^2 u + (1-A)\, u + A v \\ \partial_t v &= D_v \nabla^2 v + (1-A)\, v - A u \\ A &= u^2 + v^2. \end{aligned} \tag{9}$$

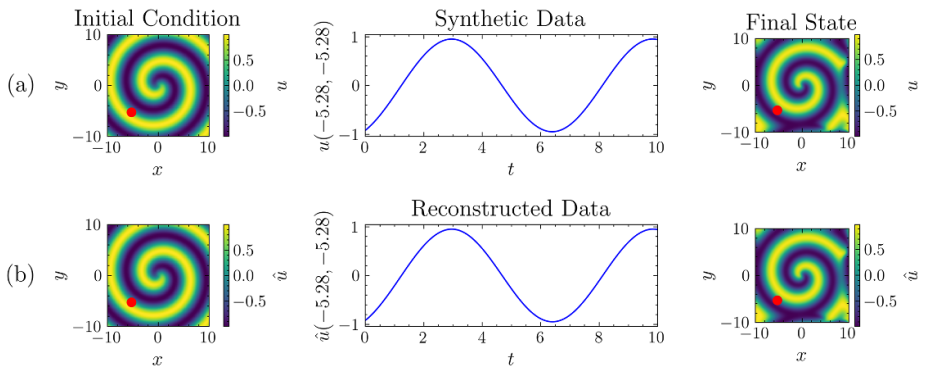

Figure 2. Time simulation of equations (9) with $D_u = D_v = 0.1$ (first row) and of the data-based estimated equations (12) (second row). In both cases the spatial grid is 128×128, the domain size is $L_x = L_y = 20$ and the time step is 0.05. Images on the left show the identical initial conditions, images on the right show the final snapshot. The graphs in the center give the time trace at the location marked in red within the snapshots. As the spiral rotates the traces oscillate. Very good agreement between the original synthetic data and their reconstruction with the estimated model is found.

We use a two-dimensional spatial domain with coordinates x and y each ranging from -10 to 10 (thus with domain size $L_x \times L_y = 20 \times 20$) and periodic boundary conditions. The initial condition corresponds to a spiral centered at the origin given by

$$\begin{aligned} u_0(x,y) &= \tanh(r)\cos(\theta - r), \\ v_0(x,y) &= \tanh(r)\sin(\theta - r), \\ \text{where} \quad r &= \sqrt{x^2+y^2}, \quad \theta = \arg(x+iy). \end{aligned} \tag{10}$$

In terms of a linear combination of polynomial functions in the fields and their derivatives the system reads

$$\begin{aligned} \partial_t u &= D_u \partial_x^2 u + D_u \partial_y^2 u + u - u^3 - uv^2 + u^2 v + v^3 \\ \partial_t v &= D_v \partial_x^2 v + D_v \partial_y^2 v + v - v^3 - u^2 v - uv^2 - u^3. \end{aligned} \tag{11}$$

Synthetic data are generated using a time simulation of equations (9) with initial condition (10). The u-component is shown in the first row of figure 2. Here and in the following, results shown for PySINDy have been produced using the corresponding Python package [40]. For the sake of comparability, here, we use our method with only one threshold h and with the identical library as employed in [40], as described in appendix A.2.

With a maximum order $m = 3$ of power products and a maximum order $p = 2$ for spatial derivatives both, our method implemented in TSME using the BIC (equation (8)) as well as the one implemented in [40] (PySINDy) identify the system as:

$$\begin{aligned} \partial_t \hat{u} &= 0.10\partial_x^2 \hat{u} + 0.10\partial_y^2 \hat{u} + 0.96\hat{u} - 0.96\hat{u}^3 - 0.96\hat{u}\hat{v}^2 + 1.00\hat{u}^2\hat{v} + 1.00\hat{v}^3 \\ \partial_t \hat{v} &= 0.10\partial_x^2 \hat{v} + 0.10\partial_y^2 \hat{v} + 0.96\hat{v} - 0.96\hat{v}^3 - 0.96\hat{u}^2\hat{v} - 1.00\hat{u}\hat{v}^2 - 1.00\hat{u}^3 \end{aligned} \tag{12}$$

(here and in the following results are rounded to the second decimal place, see appendix D table 1 for complete results). The time simulation for this reconstructed model is given in the second row of figure 2. Both methods correctly identify the underlying equations out of 110 candidate functions with a single-digit error percentage ($\approx 4.09\,\%$) for the prefactors. While for PySINDy the threshold from the original reporting in [11] is used ($h = 0.08$), our optimization procedure found a threshold $h \approx 0.0812$. Note, that [11] seems to have determined h heuristically, while the here developed optimization method automatically chooses h to minimize the BIC (8).

3.1.2. Influence of spatial discretization and noise

Here, we first briefly compare the performance of our method implemented in TSME and and the method of [11] implemented in PySINDy [40] with respect to the spatial sampling rate N for the $N \times N$ spatial grid of data points. We again perform time simulations for equations (9) with identical model parameters, time interval and boundary conditions, but for varying N and with random initial conditions, where for each variable all N^2 values are sampled from a uniform distribution over the interval

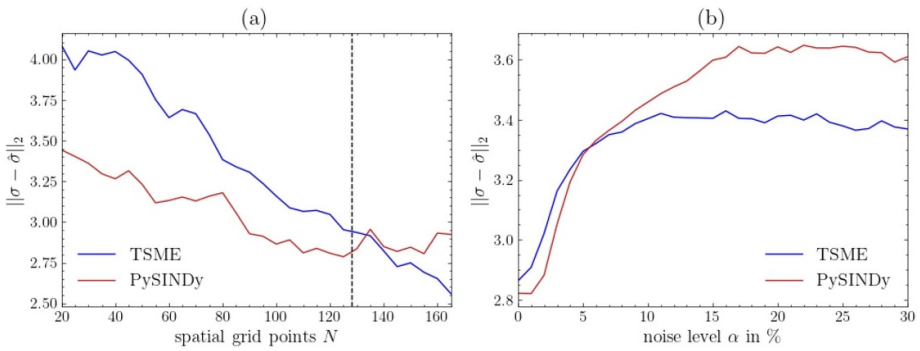

Figure 3. Average ℓ^2-error of the model parameters $||\sigma - \hat{\sigma}||_2$ over 10 data series for (a) varying spatial sampling rate N and without additive noise and (b) for additive noise of various strength α at fixed $N = 128$. In panel (a) N increases from 20 to 165 in increments of 5 and in panel (b) α increases from 0 % to 30 % in increments of 1 %. The vertical dashed line in panel (a) indicates the origin of panel (b) at $N = 128$ and $\alpha = 0$.

$[-0.25, 0.25)$. Thereby, PySINDy optimizes only with respect to the residual $R = ||\partial_t u - \hat{\sigma} \cdot \theta||_2$ $(h = 0.08)$ and TSME instead incorporates the BIC (in both cases we use the library from section 3.1.1 with $m = 3, p = 2$) averaged over 10 model estimations for each N. Figure 3(a) shows the error of the estimated model parameters. We see that both methods are similarly susceptible to a low spatial sampling rate and have a performance that improves with increasing N. While TSME is not as good as PySINDy for small N it shows a stronger improvement with increasing N and is the better method at larger N.

Second, we compare the performance of TSME and PySINDy with respect to their resilience to noisy data, in particular, we fix $N = 128$ and add noise to all input data using

$$\tilde{u} = u(1 + \alpha\eta), \tag{13}$$

where η is a random real number sampled from a normal distribution with mean value 0 and standard deviation 1 and α is a scaling factor that gauges noise strength. Otherwise, our methodology remains unchanged, we take the same model parameters, time interval, boundary and initial conditions as before and perform 10 model estimations for each of various α. The results are shown in figure 3(b). We see that the performance of both methods suffers after only a few percent of noise added. While the overall error of our method is lower for stronger noise, both TSME and PySINDy fail to reliably recover the correct model after approximately 5 %. This implies that a de-noising of input data is advisable when dealing with noisy data. Alternatives are briefly discussed in the conclusion.

3.1.3. Varying sample frequency in time

Here, we investigate the performance of TSME and PySINDy for different time step sizes of the data. Time simulations are performed for equations (9) with the previous model parameters and random initial conditions without additive noise at fixed $N = 128$. The resulting figure 4 shows (a) the average deviation of the estimated model parameters from their true value, (b) the average number of incorrectly identified model parameters, and (c) the average deviation of a reconstructed time series using the estimated model from the original time series.

As we can see in figure 4(a) for sufficiently small time steps both methods yield parameters very close to their true values. Thereby, PySINDy shows a smaller deviation from the parameter values for smaller time steps, but also less sparsity across all time step sizes (i.e. more incorrectly identified model parameters in figure 4(b)) and thus a greater error in the reconstructed time series (figure 4(c)). So while the proposed optimization method deviates more from true model parameter values for smaller time step sizes, it identifies correct model terms more reliably and thus yields better results for time integration. It should be noted, that PySINDy contains various extensions to improve the performance when sub-sampling data or to accommodate poorly sampled data, which are not being used here. By simply incorporating time integration into these methods, one should be able to increase the robustness further.

Between the two methods the average number of incorrectly identified terms varies greatly, but seems to be independent of the time step size between samples. While the variance could be reduced by increasing the number of time series per sample, it remains unclear as to why sparsity does not diminish for increasing time step size as one would expect. Similarly, the errors between the estimated time series and the original one do not mirror the trend seen in the errors of the model parameters.

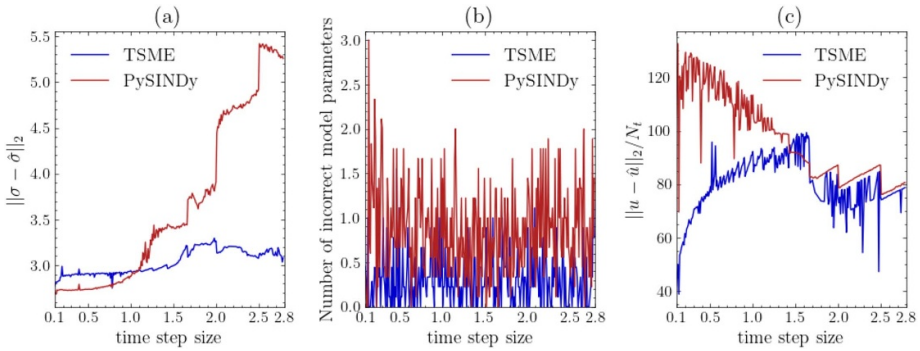

Figure 4. Average estimation error over 10 data series with random initial conditions for varying time step size, ranging from 0.1 to 2.8 in increments of 0.01. Panel (a) shows the average ℓ^2-error of the model parameters $||\sigma - \hat{\sigma}||_2$, panel (b) gives the average number of falsely identified model parameters, which may be a positive real number due to averaging, panel (c) shows the average root mean squared error (RMSE) of the reconstructed time series using the estimated model $||u - \hat{u}||_2/N_t$. The proposed BIC error shows a slightly increased deviation in its model parameters for lower time step sizes, but greatly reduced error for larger time step sizes. Sparsity of the BIC error consistently outperforms the error w.r.t. only the time derivative.

3.2. Phase field models for pattern formation

Phase field models, such as the Allen–Cahn (AC) equation and the Cahn-Hilliard (CH) equation, provide a description of interface evolution and phase separation in various physical systems. The key distinction between these two models is their 'mass-conservation character': The AC equation describes the dynamics of a non-conserved order parameter, where the local field can change dynamically without restriction, e.g. in reaction to external fields. In contrast, the CH equation locally enforces mass conservation as it has the form of a continuity equation. This ensures that phase separation does not alter the total quantity of each material. The non-conserved case lends itself readily to regression methods for model estimation, whereas the conserved case often requires constrained optimization approaches. In this section we show that the here proposed optimization method can directly be applied in both cases.

3.2.1. Allen–Cahn equation

The AC equation has been widely used to model phase transitions across different scientific fields. In material science, it describes the phase separation in binary alloys and order-disorder transitions [41]. As such it has become a basic model to study crystal growth and interfacial dynamics [42, 43]. In biology it forms a basis for models simulating cancer cell migration, providing insights into tumor development and metastasis [44].

We consider the following AC equation, that represents a gradient dynamics on a double-well potential:

$$\partial_t u = \nabla^2 u + u - u^3. \tag{14}$$

Here, $u(x,t)$ is a non-conserved order parameter, typically representing a phase field of a bistable system. We rewrite it in terms of our default library as

$$\partial_t u = \partial_x^2 u + \partial_y^2 u + u - u^3. \tag{15}$$

A time simulation of equation (14) for periodic boundary conditions and uniform random initial conditions (over the interval $[-0.25, 0.25)$) can be found in the first row of figure 5. Now we apply the BIC optimization method with the default library (see appendix A.1) with $m = 3$ and $p = 4$ (i.e. we allow for spatial derivatives up to 4th order and up to cubic nonlinearities encompassing in total 46 terms). We obtain

$$\partial_t \hat{u} = 0.99\partial_x^2 \hat{u} + 0.99\partial_y^2 \hat{u} + 0.99\hat{u} - 0.99\hat{u}^3 \tag{16}$$

(for complete results see appendix D table 2). A time simulation of the estimated model is found in the second row of figure 5.

Consistent with section 3.1.1 we find very good agreement between original and estimated model with errors below 2%.

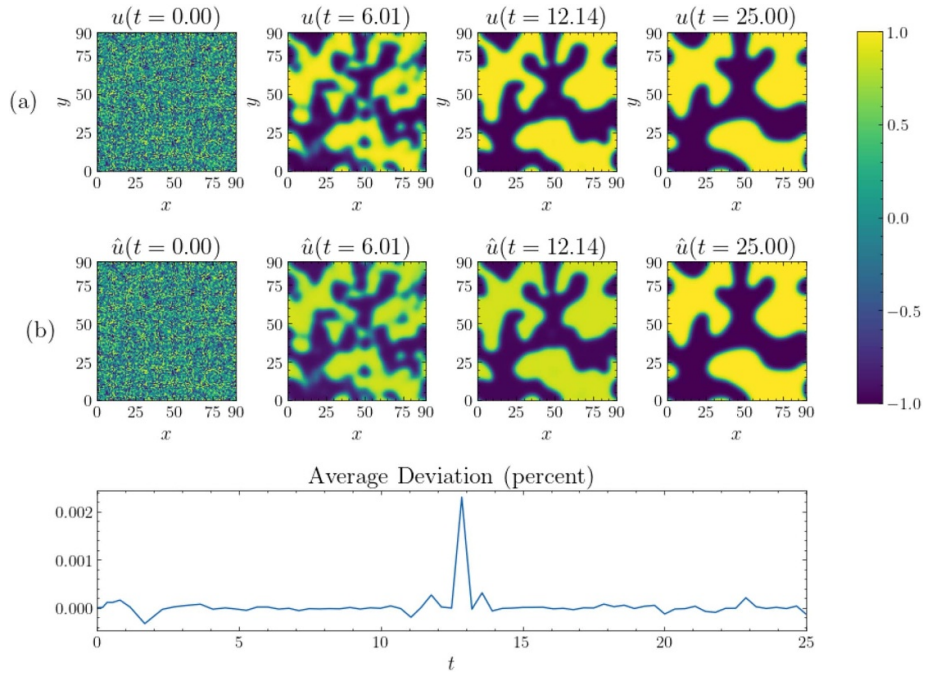

Figure 5. Time simulation of the original Allen–Cahn equation (14) (top row) and the estimated equation (16) (middle row) for a time step of 0.125 (spatial domain 128×128 and $L_x = L_y = 90$). The bottom row show the average deviation of the reconstructed time series in percent. The estimated model shows very good agreement.

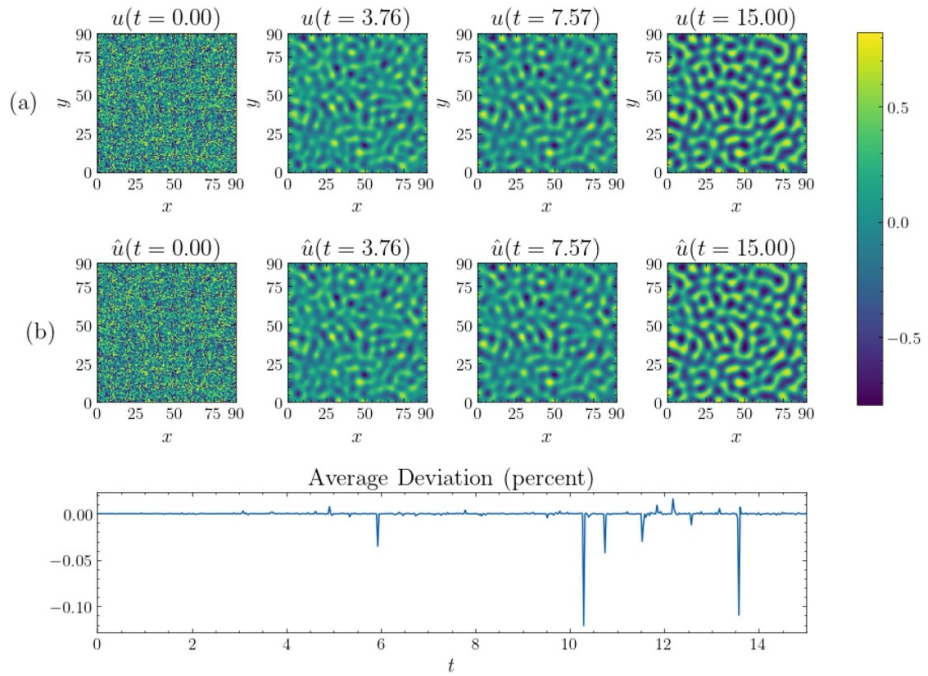

Figure 6. Time simulation of the original Cahn–Hilliard equation (17) (top row) and the estimated equation (19) (middle row) for a time step of 0.15 (spatial domain 128×128 and $L_x = L_y = 90$). The bottom row show the average deviation of the reconstructed time series in percent.

3.2.2. Cahn-Hilliard equation

Next, we consider spinodal decomposition in a binary mixture as described by the CH equation. Generally, the CH equation is used to study structure formation in phase-separating alloys or other multi-component systems, often in chemical or biological contexts [45]. Therefore, one may expect to find the CH equation or related models when investigating inverse problems involving demixing behavior.

Here, we use the original equation

$$\partial_t u = \nabla^2 \left(u^3 - u - \nabla^2 u\right), \tag{17}$$

where u denotes a difference in concentration between two species. Here as before we consider periodic boundary conditions and uniform random initial conditions (over the interval $[-0.25, 0.25)$).

As noted in [46], several points related to ill-posedness deserve attention when estimating PDE models of CH-type, i.e. gradient-dynamics systems characterized by an energy functional and a mobility function. Any method that identifies the governing PDE effectively recovers the chemical potential, which can, in principle, be integrated to obtain the corresponding energy functional. For such a mass-conserving dynamics the corresponding integration constant is arbitrary and of no importance. Likewise, a constant external chemical potential (equivalent to a linear term in the energy) merely acts as a Lagrange multiplier enforcing the mean value of the field, which, however, is fixed by the initial condition and can be read directly from the data. Furthermore, there is an ambiguity between energy functional and a mobility function that can individually only be estimated up to a relative factor. However, also this is only of limited physical consequence, as it corresponds to a choice of time and energy scales. If necessary, this can be resolved by analyzing the final steady state to extract the energy directly. Note, however, that the candidate library of terms must be expanded if the mobility is expected to depend on the field as, e.g. for thin-film equations [47].

Restated in terms of our default library again with $m = 3$ and $p = 4$ equation (17) reads:

$$\partial_t u = -\partial_x^2 u - \partial_y^2 u + \partial_x^2 u^3 + \partial_y^2 u^3 - \partial_x^4 u - 2\partial_x^2 \partial_y^2 u - \partial_y^4 u. \tag{18}$$

Note that nonlinearities with second-order spatial derivatives cannot be included using the implementation provided by PySINDy in [40] as it lacks the ability to input user–defined library terms and terms like $\partial_x^2 u^3$ are not of the form used in the example of section 3.1.1, i.e. they cannot be rewritten in terms that only include spatial derivatives of linear order of u. The implementation presented here allows the user to manipulate the default library. Furthermore, the default library already includes the nonlinear terms with second-order spatial derivatives. Utilizing this implementation we obtain as result

$$\partial_t \hat{u} = -0.99\,\partial_x^2 \hat{u} - 0.99\,\partial_y^2 \hat{u} + 0.98\,\partial_x^2 \hat{u}^3 + 0.99\,\partial_y^2 \hat{u}^3 - 1.02\,\partial_x^4 \hat{u} - 1.92\,\partial_x^2 \partial_y^2 \hat{u} - 1.02\,\partial_y^4 \hat{u}, \tag{19}$$

(for a complete list see appendix D table 3). Again the method correctly identifies the original equation correctly from 45 candidate terms within single digit ($\lesssim 4\,\%$) error percentage, despite the presence of a conservation law and without further constraints on the library. For a Python tutorial on how to obtain these results please see appendix C.

3.3. Excitable, oscillatory, and wave propagation systems

Up to this point we only utilized a single hyperparameter, i.e. one threshold for all variables in the STLS method. To illustrate the flexibility of our optimization approach we next study three systems, which inherently benefit from including more than one hyperparameter in the optimization procedure.

3.3.1. FitzHugh-Nagumo model

The FHN model [48, 49] is a two–species reaction-diffusion model central in investigations of the patterns of neuronal firing, synchronization of neural networks and the emergence of complex behavior in excitable media [50, 51]. It is derived as a simplification of a model for the transmission of electrical impulses along a nerve fiber, as introduced by Hodgkin and Huxley [52]. Due to its rich bifurcation behavior, the accurate estimation of the model parameters is vital to understanding how the states and patterns evolve in time. As such the FHN model is frequently considered in the context of parameter identification [53, 54]. Here we study the FHN model in the form

$$\begin{aligned} \partial_t u &= D_u \nabla^2 u + \lambda u - u^3 - \omega v \\ \tau \partial_t v &= D_v \nabla^2 v + u - v, \end{aligned} \tag{20}$$

where u describes a membrane potential, v a recovery variable, λ a threshold for excitability, ω gauges inhibition strength, while D_u and D_v are respective diffusion constants. Again we use periodic boundary conditions and uniform random initial conditions (over the interval $[-0.25, 0.25)$). The top row of figure 7 shows snapshots from a sample time series.

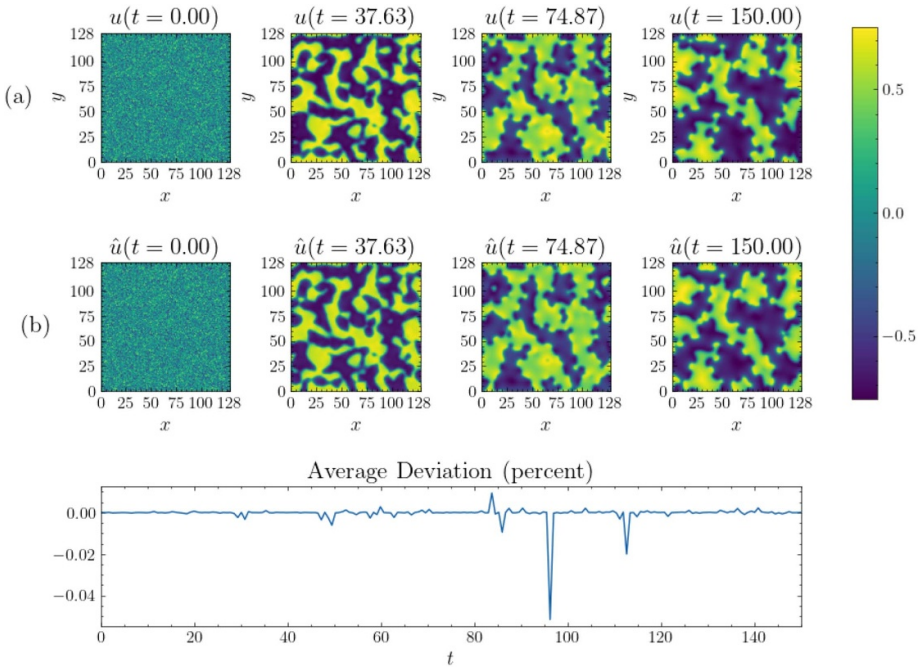

Figure 7. Snapshots from a time simulation of the original FHN equations (21) with $D_u = 0.5$, $D_v = 0.0\overline{6}$ (top row) and the estimated equations (22) (middle row) at a time step of 0.75 (spatial domain 256×256 and $L_x = L_y = 128$). The bottom row shows the average deviation of the reconstructed time series in percent.

We rewrite the model in terms of the adequate candidate functions that are a subset of our library and insert the specific parameters used in figure 7:

$$\begin{aligned} \partial_t u &= 0.5u - 0.5v - 1.0u^3 + 0.5\partial_x^2 u + 0.5\partial_y^2 u \\ \partial_t v &= 0.0\overline{6}u - 0.0\overline{6}v + 0.0\overline{3}\partial_x^2 v + 0.0\overline{3}\partial_x^2 v, \end{aligned} \tag{21}$$

Note that the order of magnitude of the parameters differs between the two equations. To account for this we enable individual thresholds h_u and h_v for each equation in the sequential thresholding procedure and optimize for both. With our default library as described in appendix A.1 with parameters $m = 3$ and $p = 2$, we identify the equations:

$$\begin{aligned} \partial_t \hat{u} &= 0.49\hat{u} - 0.49\hat{v} - 0.97\hat{u}^3 + 0.24\partial_x^2 \hat{u} + 0.24\partial_y^2 \hat{u} \\ \partial_t \hat{v} &= 0.07\hat{u} - 0.07\hat{v} + 0.02\partial_x^2 \hat{v} + 0.02\partial_y^2 \hat{v}, \end{aligned} \tag{22}$$

out of a library of 54 candidate terms for each equation (for full results see appendix D table 4). We observe that all terms have been correctly identified, albeit with a diffusion coefficient for $\hat{u}$ that seems to be diminished by half. The obtained thresholds are $h_u = 0.07$ and $h_v = 0.01$. When using only one threshold, we find two cases. Either, the threshold is small enough, such that the second equation contains non-zero entries. However, in turn this yields non-zero contributions for virtually all available library functions in the first equation. Or, the threshold is large enough to lead to a sparse representation of the first equation, but in turn eliminates all library functions from the second equation. These results indicate that one needs several thresholds, when one expects groups of model parameters to differ in their orders of magnitude.

3.3.2. Exploring chaotic regimes

Next, we revisit the cGL equation (9) from section 3.1.1, this time for a parameter regime where the system exhibits chaotic behavior like also considered in [55]. First, we show that our method maintains its performance even in a chaotic regime.

Second, we highlight the benefit of incorporating additional hyperparameters, namely thresholds for sub-groups of library terms as described in section 2.2. Here we choose a separate threshold for all the diffusion terms.

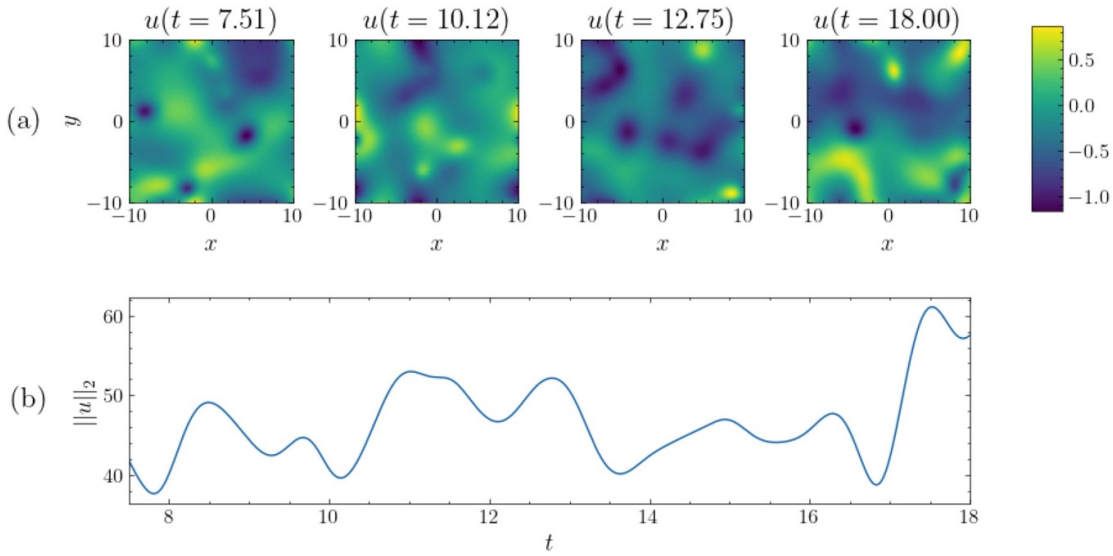

Figure 8. Time simulation in the defect turbulence regime of equations (23) at $D_1 = 1.0$, $D_2 = 2.0$ and $\beta = -4$ with adaptive time step size between 0.004 326 and 0.017 094 (spatial domain 128×128 and $L_x = L_y = 20$). The first row shows exemplary snapshots of u and the second row shows the ℓ^2-norm of u over time. The initial condition for both u and v is white noise between values -0.5 and 0.5.

To allow for different chaotic regimes we slightly modify the reaction–diffusion system, by additionally coupling the fields by cross diffusion terms, i.e.

$$\begin{aligned} \partial_t u &= D_1 \nabla^2 u - D_2 \nabla^2 v + (1 - A)\, u + \beta A v \\ \partial_t v &= D_1 \nabla^2 v + D_2 \nabla^2 u + (1 - A)\, v - \beta A u \\ A &= u^2 + v^2. \end{aligned} \tag{23}$$

Boundary conditions are periodic and initial conditions are uniform random (over the interval $[-0.25, 0.25]$). With the particular choice of diffusion constants, the system is equivalent to the cGL equation [55] that features different types of chaos. The added parameter β serves as a way to control the coupling strength, as well as the phase rotation of a complex amplitude $w = u + iv$.

Figure 8 shows a time simulation of equations (23) for $D_1 = 1, D_2 = 2, \beta = -4$, which corresponds to a chaotic defect turbulence regime. Eliminating the cross coupling terms at otherwise identical parameters places the system in an intermittency regime with more coherent structures, as illustrated in figure 9.

Applying the estimation method with a single thresholding hyperparameter in the defect turbulence regime very closely recovers the system parameters:

$$\begin{aligned} \partial_t u &= 1.00\partial_x^2 u + 1.00\partial_y^2 u - 2.00\partial_x^2 v - 2.00\partial_y^2 v + 1.00u - 1.00u^3 - 1.00uv^2 - 4.00u^2 v - 4.00v^3 \\ \partial_t v &= 1.00\partial_x^2 v + 1.00\partial_y^2 v + 2.00\partial_x^2 u + 2.00\partial_y^2 u + 1.00v - 1.00v^3 - 1.00u^2 v + 4.00uv^2 + 4.00u^3 \end{aligned}$$

(for full results see appendix D table 5). In contrast, when applying the method in this way in the intermittency regime, it either fails to eliminate higher-order diffusion terms (see terms 44, 47, 50 and 53 in table 6 of appendix D) or incorrectly eliminates all diffusion terms. The problem is resolved when making use of the methods ability to include additional hyperparameters. We incorporate two additional thresholds, one for each of the two pairs of second order diffusion terms (corresponding to $\nabla^2 u$ and $\nabla^2 v$). This dramatically improves results in the intermittency regime (see appendix D table 7 for full results):

$$\begin{aligned} \partial_t u &= 0.97\partial_x^2 u + 0.90\partial_y^2 u + 0.96u + 0.96u^3 - 0.96uv^2 + 3.99u^2 v - 3.99v^3 \\ \partial_t v &= 0.97\partial_x^2 v + 0.90\partial_y^2 + 0.96v - 0.96v^3 - 0.96u^2 v + 3.99uv^2 + 3.99u^3. \end{aligned} \tag{24}$$

This behavior may be due to rapid changes of large coherent structures in the input time series, akin to the time series rapidly oscillating. This, in turn, effectively decreases spatial resolution of diffusive boundaries and produces dynamics that are less dependent on diffusion, leading to difficulties when estimating diffusion coefficients.

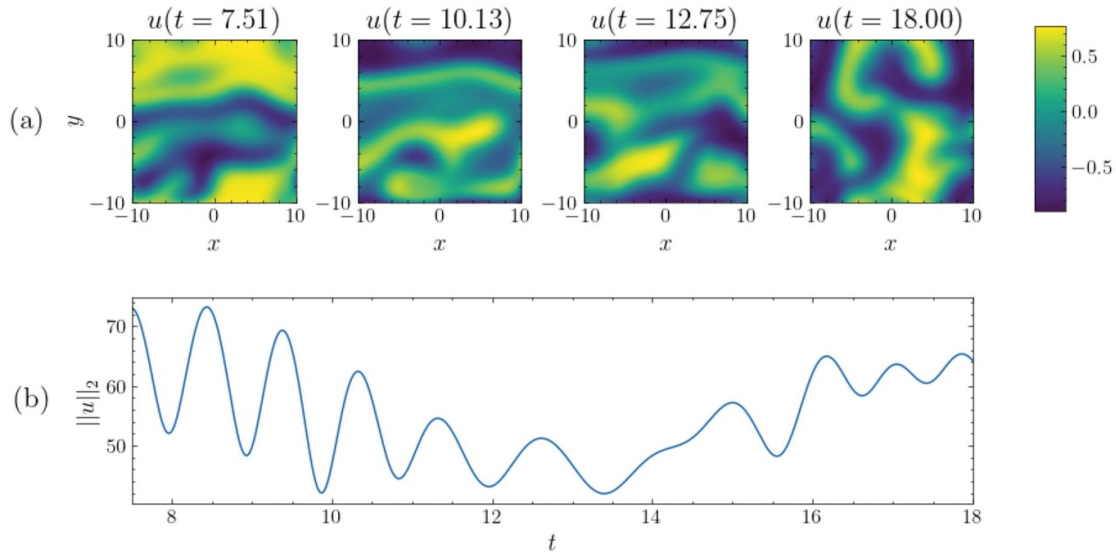

Figure 9. Time simulation in the intermittency regime of equations (23) at $D_1 = 1.0$, $D_2 = 0.0$ and $\beta = -4$ with adaptive time steps between 0.007 326 5 and 0.020 553 (spatial domain 128×128 and $L_x = L_y = 20$). The first row shows exemplary snapshots of u and the second row shows the ℓ^2-norm of u over time. The initial condition for both u and v is white noise between values -0.5 and 0.5.

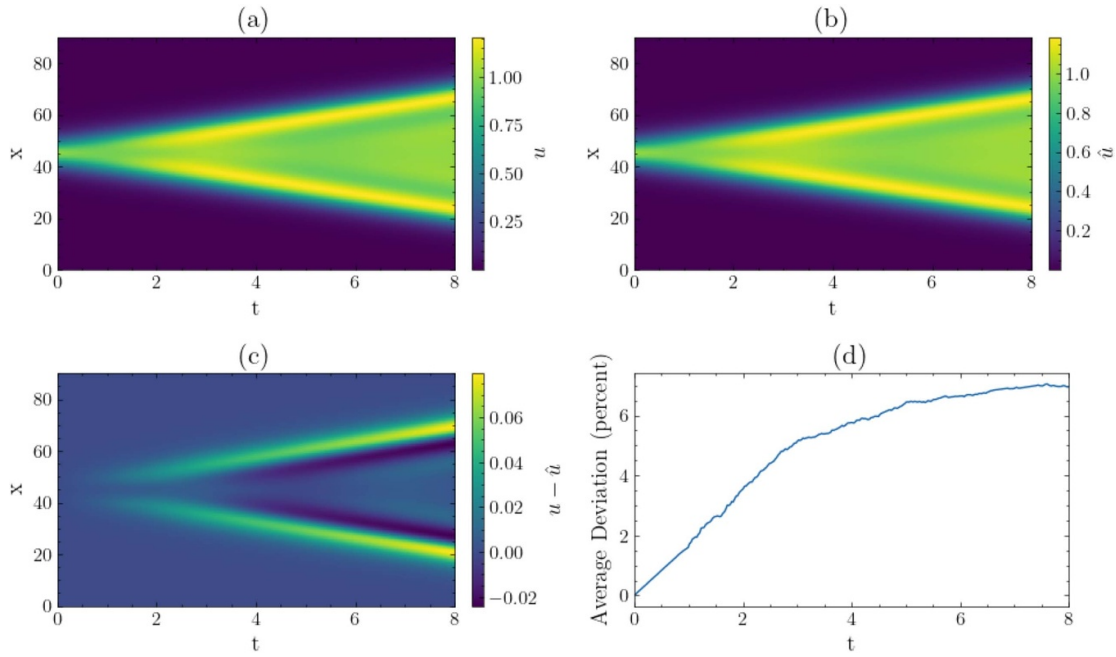

Figure 10. Time simulations of the Fisher-KPP equation (25), with all parameters set to 1.0, time step size 0.0266 (spatial discretization 128, $L_x = 90$) (top left) and estimated model (26) (top right), as well as the absolute difference between the two (bottom left) and the average deviation of the estimate relative to the original time series for every time step (bottom right).

3.3.3. Fisher-KPP equation with time delay

The efficient handling of additional hyperparameters opens a straight forward way to handle delay equations. To our knowledge this has not yet been addressed in the framework of sparse system identification. As example we consider the Fisher-KPP equation [56, 57]. It is a one-species reaction-diffusion system used, e.g. to study propagation fronts of biological populations. Here, we include a time delay τ in the reaction term as inspired by [58, 59]. The equation then reads

$$\partial_t u(t) = D\partial_x^2 u(t) + ru(t)\left(1 - \frac{u(t-\tau)}{K}\right), \tag{25}$$

where D is a diffusion coefficient, r an intrinsic growth rate and K a carrying capacity. The time delay τ represents a lag between the current population growth and past population pressure. This accounts for example for the time it takes for members of a given population to reach reproductive maturity or other delayed feedback introduced by external time-delayed resources.

For the sake of simplicity all parameters (including the time delay τ) are set to 1.0, boundary conditions are periodic and the initial condition $u_0(x) = \mathrm{sech}^2\left(0.2\left(x - \frac{L_x}{2}\right)\right)$ is set for all $t \leqslant 0$. (the domain size is $L_x = 90$.) A space-time plot of the simulation is shown figure 10 (a).

To account for the time-delayed variable, we add 7 terms in $u(t-\tau)$ to our default library with $m = 3$ and $p = 4$ and add τ to the hyperparameters for optimization, such that some library terms now change during optimization. This results in the identified model:

$$\partial_t \hat{u}(t) = 1.02\partial_x^2 \hat{u}(t) + 0.98\hat{u}(t) - 0.97u(t)\,\hat{u}(t-1.02) \tag{26}$$

(see appendix D table 8 for complete results). Again we find that the correct model is identified out of 23 candidate terms for the right-hand side. Figure 10 (b) shows a space-time plot of the estimated model. While we see that the error of the estimation is greatest at the propagation fronts (see figure 10 (c)) and increases over time (see figure 10 (d)), the average deviation of the reconstructed time series seems to approach a constant value (see again figure 10 (d)).

4. Conclusion and outlook

In this work, we have shown how Bayesian optimization can improve data-driven model estimation methods, by automatically finding hyperparameters of the numerical method and/or of the aimed-at model. This has allowed us to extend the thresholding algorithm used in [4] with additional thresholds, which affect individual parts of the estimated model structure. This has enabled model recovery from synthetic data with parameters of different orders of magnitude including models that show relaxational, time-periodic and chaotic behaviors. We have extended the method introduced in [31] for ODE models toward PDE and delay-PDE models.

In sections 3.1.1 and 3.1.2 we have shown that, for an elementary case, the overall performance and robustness against under-sampling in space and against the occurrence of additive noise are similar for our method implemented in TSME and the established method implemented in PySINDy. For both methods, further considerations are required when dealing with any significant amount of noise. In our case, this implies that the supplied data need to be de-noised before estimation. In the future, one may adapt more robust variations of the SINDy algorithm (such as, the one presented in [26, 60]) for use with hyperparameter optimization and the BIC.

While TSME results in higher computational cost, we have shown that it outperforms PySINDy in cases of data that are more sparsely sampled in time (section 3.1.3). TSME allows for a wider range of permissible model structures and provides an implementation that allows for user-defined library terms. Furthermore, section 3.2.1 has shown that our method performs well in scenarios where conservation laws typically require additional constraints. In particular, in section 3.2.2 we have recovered a Cahn-Hilliard model where PySINDy's functional library is ill-suited. Notably, as long as the employed data represent a dynamics that involves a range of field values that is not too small the Cahn-Hilliard and Allen–Cahn systems can still be identified even without reaching extremal values or an equilibrium, because the library functions naturally extrapolate toward the extremal values. Then, once a model has been estimated it may be simulated in time to any extent and thus can reach values not present in the initial data. Several points related to ill-posedness [46] have been discussed in section 3.2.2.

Additionally, via the usage of additional hyperparameters our method can accommodate on the one hand differences in the order of magnitude of parameters either between equations for the different field variables or between groups of terms within an equation (section 3.3.1) and on the other hand added system parameters such as a time delay (section 3.3.3). This would typically require either a grid-based search through the hyperparameter space or an altogether altered implementation compared to the one presented in [40]. We have shown that our method is very flexible and can accommodate any number of hyperparameters. This can be used to account for semantic connections between ansatz functions by making them share an additional hyperparameter. This yields more precise results as shown for some cases in section 3.3.2.

Since model estimation of higher quality takes precedent over computation time, our results imply that the use of time integration should in the future be incorporated into other existing methods. Our method can be further improved by including sub-sampling or gradient-based optimization. Its library can easily accommodate more complex ansatz functions, which rely on additional hyperparameters, such

as external driving forces or temperature dependent functions. This greatly increases the number of permissible modeling approaches for unknown systems. Another topic of interest are models with discontinuous solutions, e.g. a thin-film equation with a wetting energy that does not diverge for zero film height. While the present method could apply to such problems in theory, they should be treated in a dedicated manner, either by adapting the target function of the optimization or the optimization scheme itself.

Importantly, after the synthetic data used here, in the future, the method needs to be applied to data obtained in discrete stochastic model approaches and experiments, where the continuum description of the dynamics is not explicitly known (but ideally a rich theoretical background exists that allows one to tune the term library and hyperparameter strategy).

Data availability statement

The data that support the findings of this study are openly available at the following URL: https://github.com/CDSC-CoSyML/tsme_examples [39].

Appendix A. Generating the library of functions

In this section we detail different strategies to generate the set of library functions used by the method developed in section 2. They are based on the physical quantity $u(x,t) \in \mathbb{R}^n$ we ultimately wish to describe, with t being the time and $x \in \mathbb{R}^d, d \in \{1,2,3\}$ the position in d-dimensional space. Each library function has a coefficient associated with it, that will be optimized to best describe data of u by the method of section 2.

A.1. Default library

We construct a set of monomials (or power products) in the n field variables u_i from zeroth order up to order m. Then, for each non-trivial monomial we compute all (partial) spatial derivatives up to order p. More specifically let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ be a multi-index such that

$$\boldsymbol{\alpha} \in \mathbb{N}_0^n, |\boldsymbol{\alpha}| = \sum_{i=1}^{n} \alpha_i \leqslant m, \tag{27}$$

then, each monomial is of the form

$$U_{\boldsymbol{\alpha}} = \prod_{i=1}^{n} u_\mathrm{i}^{\alpha} = u_1^{\alpha_1} u_2^{\alpha_2} \cdots u_n^{\alpha_n}. \tag{28}$$

For example, in the case $n=2$, $m=2$ and $p=0$ the set of monomials is

$$\left\{1, u_1, u_2, u_1^2, u_1 u_2, u_2^2\right\}.$$

Similarly for the spatial derivatives, let $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_d)$, where d is the spatial dimension, be a multi-index such that

$$\boldsymbol{\beta} \in \mathbb{N}_0^d, |\boldsymbol{\beta}| = \sum_{j=1}^{d} \beta_j \leqslant p, \tag{29}$$

where the corresponding differential operator is

$$\partial_x^{\boldsymbol{\beta}} = \partial_{x_1}^{\beta_1} \partial_{x_2}^{\beta_2} \cdots \partial_{x_d}^{\beta_d}. \tag{30}$$

For example, with $d=2$ and $p=2$ the set of differential operators is

$$\left\{\partial_{x_1}^0 \partial_{x_2}^0, \partial_{x_1}^1 \partial_{x_2}^0, \partial_{x_1}^0 \partial_{x_2}^1, \partial_{x_1}^2 \partial_{x_2}^0, \partial_{x_1}^1 \partial_{x_2}^1, \partial_{x_1}^0 \partial_{x_2}^2\right\}.$$

Now we apply each differential operator to each monomial except the one for $|\alpha| = 0$. For the sake of sequencing, let us group all terms by order $|\boldsymbol{\beta}|$. This gives the vector of ansatz functions, by order q of their respective differential operator:

$$\Theta^{(q)} = \left[\partial_x^{\boldsymbol{\beta}} U_{\boldsymbol{\alpha}} \;\middle|\; 1 \leqslant |\boldsymbol{\alpha}| \leqslant m, |\boldsymbol{\beta}| = q\right]. \tag{31}$$

So for any maximal order p, we take all ansatz functions with $0 \leqslant q \leqslant p$. Combining the earlier examples ($d=2, n=2, m=2$, and $p=2$, i.e. $q=0,1,2$):

$$\Theta^{(0)} = \left[u_1, u_2, u_1^2, u_1u_2, u_2^2\right]$$
$$\Theta^{(1)} = \left[\partial_{x_1}u_1, \partial_{x_1}u_2, \partial_{x_1}u_1^2, \partial_{x_1}\left(u_1u_2\right), \partial_{x_1}u_2^2, \partial_{x_2}u_1, \partial_{x_2}u_2, \partial_{x_2}u_1^2, \partial_{x_2}\left(u_1u_2\right), \partial_{x_2}u_2^2\right]$$
$$\Theta^{(2)} = \left[\partial_{x_1}^2u_1, \partial_{x_1}^2u_2, \partial_{x_1}^2u_1^2, \ldots, \partial_{x_1}\partial_{x_2}u_1, \partial_{x_1}\partial_{x_2}u_2, \ldots, \partial_{x_2}^2u_2^2\right].$$

We note here, that terms such as $\partial_x(u_1u_2)$ in $\Theta^{(1)}$ could be expressed as two terms ($u_1\partial_x u_2$ and $u_2\partial_x u_1$) and therefore could have different coefficients within the estimation procedure, e.g. for the case of non-linear diffusion. An entirely inclusive case could be based on the generalized product rule. So finally our entire library consists of the concatenation of all the elements of these vectors:

$$\Theta = \left[1\,\Theta^{(0)}\,\Theta^{(1)}\ldots\Theta^{(p)}\right]^{\mathrm{T}}. \tag{32}$$

The total number k of all these elements may become fairly large due to combinatorics:

$$k = 1 + (\text{Number of monomials with } |\boldsymbol{\alpha}| \geqslant 1) \times (\text{Number of derivatives}) \tag{32}$$
$$= 1 + \left(\sum_{r=1}^{m}\binom{n+r-1}{r}\right)\cdot\left(\sum_{q=0}^{p}\binom{d+q-1}{q}\right). \tag{33}$$

Following through with our example for $d=2, n=2, m=2, p=2$ we find $k=31$. This default library may be modified by removing certain terms or adding terms, such as functions with explicit time dependence or $u(x, t-\tau)$, where τ is a set time delay. Also non-analytic functions may be included.

A.2. Alternative library: PySINDy

For the sake of comparison with [40] we recount the library used in PySINDy. It consists of power products as in our case and terms linear in spatial derivatives of u. This leads to the following entries in the library:

$$\Theta^{(q)} = \left[U_{\boldsymbol{\alpha}}\partial_x^{\boldsymbol{\beta}}u_{\mathrm{i}} \;\middle|\; 0 \leqslant |\boldsymbol{\alpha}| \leqslant m, |\boldsymbol{\beta}| = q, 0 \leqslant i \leqslant n\right] \tag{34}$$

and just as before (only now with the trivial element included in $\Theta^{(0)}$):

$$\Theta = \left[\Theta^{(0)}\,\Theta^{(1)}\ldots\Theta^{(p)}\right]^{\mathrm{T}}.$$

For example with $d=2, n=2, m=2, p=2$:

$$\begin{aligned}\Theta^{(0)} &= \left[1, u_1, u_2, u_1^2, u_1u_2, u_2^2\right]\\ \Theta^{(1)} &= [\partial_{x_1}u_1, \partial_{x_1}u_2, u_1\partial_{x_1}u_1, u_2\partial_{x_1}u_1, u_1\partial_{x_1}u_2, u_2\partial_{x_1}u_2,\\ &\qquad u_1^2\partial_{x_1}u_1, u_1u_2\partial_{x_1}u_1, u_1u_2\partial_{x_1}u_2, u_2^2\partial_{x_1}u_1, u_2^2\partial_{x_1}u_2,\\ &\qquad \partial_{x_2}u_1, \partial_{x_2}u_2, \ldots, u_2^2\partial_{x_2}u_2]\\ \Theta^{(2)} &= \left[\partial_{x_1}^2u_1, \partial_{x_1}^2u_2, u_1\partial_{x_1}^2u_1, \ldots, u_2^2\partial_{x_1}u_2,\right.\\ &\qquad \partial_{x_1}\partial_{x_2}u_1, \partial_{x_1}\partial_{x_2}u_2, u_1\partial_{x_1}\partial_{x_2}u_1, \ldots, u_2^2\partial_{x_1}\partial_{x_2}u_2,\\ &\qquad \left.\partial_{x_2}^2u_1, \partial_{x_2}^2u_2, u_1\partial_{x_2}^2u_1, \ldots, u_2^2\partial_{x_2}^2u_2\right]\end{aligned}$$

(Please note that here the ordering differs [40].) The number k of library elements in PySINDy is:

$$k = (\text{Number of monomials}) \times (1 + \text{Number of derivatives})$$
$$= \binom{n+m}{m}\left(1 + n\sum_{q=1}^{p}\binom{d+q-1}{q}\right), \tag{35}$$

which for this example gives $k=66$. Please note that in this case m strictly only gives the highest order of monomial, not the highest order of the entire library (which is $m+1$ as each monomial is being multiplied by a derivative of a linear term in u_{i}). This approach gives more individual terms for the same values of parameters d, n, m and p than our approach, but not an identical order or coverage of the function space.

Appendix B. Pseudo-code for estimation procedure

Algorithm 1. Structure of the estimation method.

Load time series data u, time stamps t and spacial dimensions $L_1, \ldots, L_d$
if delay τ is to be used **then**
  Ensure u has values for $-\tau \leqslant t \leqslant 0$ or set those values to $u(t=0)$
end if
Compute $\partial_t u$ via finite differences and time stamps t
Create a semantic representation of (default) library terms Θ
(Optional) Modify library by removing or adding terms (such as $u(t-\tau)$)
Evaluate all library terms Θ at all time steps using FFT (or finite difference) for spacial derivatives, with applicable spatial domain sizes $L_1, \ldots, L_d$
Define search space of the TPE for all hyperparameters: thresholds h_i and τ if applicable
for $l=0$ to $l=l_{\max}$ **do**
  Recompute any library terms that may have changed (e.g. time delayed terms)
  Perform sequential thresholding least-squares with h_i to find values for $\hat{\sigma}$ minimizing $||\partial_t u - \hat{\sigma}\cdot\Theta||_2$
  Perform time integration using $\hat{\sigma}$ (and τ) to obtain $\hat{u}$
  Evaluate BIC $= s\ln(N_t) - 2\ln(||u-\hat{u}||_2)$
  Use TPE w.r.t. BIC to find new values for h_i (and τ)
end for

Appendix C. Tutorial : Cahn–Hilliard (CH) equation

This section is a modified example taken from the documentation of the Python package TSME [38] found in [61]. and will walk through the Python code needed to reproduce the results shown here (all Python code can be found in [39]) .

We consider the CH equation in 2D ($d=2$, $n=1$) discussed in section 3.2.2:

$$\partial_t u = \nabla^2\left(u^3 - u - \nabla^2 u\right).$$

We begin by performing a time simulation. As we are dealing with a spatially extended 2D system we first set our spatial discretization `N` and domain sizes `Lx` and `Ly` in x and y direction. Then, we define a time interval `time_interval`, as well as the time stamps `t_eval` where we want to sample the trajectory. The initial condition `u0` is set to uniform random values over the interval $[-0.25, 0.25)$.

```
In [1]:
import numpy as np
np.random.seed(12389)
N = 128
Lx = 90
Ly = 90
domain = ((0, Lx), (0, Ly))
time_interval = [0, 15]
t_eval = np.linspace(time_interval[0], time_interval[-1], 100)
u0 = (np.random.random((N, N)) - 0.5) * 0.5
```

Next, we create the CH problem using TSME as an implementation of the supplied `AbstractTimeSimulator` class:

```
In [2]:
from tsme.time_simulation import AbstractTimeSimulator
class CHESim(AbstractTimeSimulator):
def __init__(self, ic=None, dom=domain, params=None,
bc="periodic", diff="finite_difference"):
super().__init__(ic, domain=dom, bc=bc, diff=diff)
if params is None:
params = [1.0]
```

```
self.a = params[0]
def rhs(self, t, u_in):
u = u_in[0]
f = u ** 3 - u
- self.a * (self.diff.d_dx(u, 2) + self.diff.d_dy(u, 2))
u_next = self.diff.d_dx(f, 2) + self.diff.d_dy(f, 2)
return np.array([u_next])
```

This class allows for a number of different expressions in the right-hand side function `rhs`. Here we use the spatial derivatives in both spatial directions to construct our Laplace operator.

Now all our preparations are done, and we can perform the actual time simulation:

```
In [3]:
sim = CHESim(ic=np.array([u0]), dom=domain, params=[1.0])
sol = sim.simulate(time_interval, method="DOP853", t_eval=t_eval)

IVP: 100%|##########| 15.0/15.0 [06:19< 00:00, 25.33s/ut]
```

The output shown here is a progress bar (that would be updated in real time), which shows in percent how far in time the solver has progressed, how much time has elapsed since the computation began and how long the rest of the computation is estimated to take. Since the computation here is already finished, we see that it took about six and a half minutes to complete and that there is no computation time remaining.

Now we import the model that will estimate the differential equations.

```
In [4]: from tsme.model_estimation.model import Model
```

All we need to do now is pass along the trajectory, domain size and the time stamps to our model. We can then automatically generate a library of possible test functions for our right-hand side. Here we take the powers of up to third order ($m = 3$) and then, all spatial derivatives up to fourth order ($p = 4$) of said powers (these include both spatial directions and their mixed derivatives). In total these are 46 terms.

```
In [5]:
estimated_model = Model(sol, sim.time, phys_domain=domain)
estimated_model.init_library(3, 4)
estimated_model.print_library()

| Index | Term |                                Value 0   |
|---------|-------------------------------|-----------|
| 0 | 1.0                                 |       0   |
| 1 | u[0]                                |       0   |
| 2 | u[0] * u[0]                         |       0   |
| 3 | u[0] * u[0] * u[0]                  |       0   |
| 4 | d_dx(u[0],1)                        |       0   |
| 5 | d_dy(u[0],1)                        |       0   |
| 6 | d_dx(u[0] * u[0],1)                 |       0   |
| 7 | d_dy(u[0] * u[0],1)                 |       0   |
| 8 | d_dx(u[0] * u[0] * u[0],1)          |       0   |
| 9 | d_dy(u[0] * u[0] * u[0],1)          |       0   |
| 10 | d_dx(u[0],2)                       |       0   |
| 11 | d_dy(u[0],2)                       |       0   |
| 12 | dd_dxdy(u[0],(1,1))                |       0   |
| 13 | d_dx(u[0] * u[0],2)                |       0   |
| 14 | d_dy(u[0] * u[0],2)                |       0   |
| 15 | dd_dxdy(u[0] * u[0],(1,1))         |       0   |
[...]
```

Of course just after initialization the coefficients of all these library terms are yet to be determined and are hence set to zero. The library supports some basic manipulations at this point, for more corresponding details see online tutorials. Now we can call the optimization routine to find the best sparse combination of the library functions.

```
In [6]: estimated_model.optimize_sigma(max_evals=10)

Generating library functions (this may take some time)...
100%|##########| 10/10 [08:13< 00:00, 49.39s/trial,
best loss: 3438.0777397909883]
Optimal threshold(s) found: {'h0': 0.8189787983433431}
New Sigma set to:
| Index | Term                        | Value  0  |
|---------|-------------------------|-----------|
| 10 | d_dx(u[0],2)                 | -0.991169 |
| 11 | d_dy(u[0],2)                 | -0.993026 |
| 16 | d_dx(u[0] * u[0] * u[0],2)   | 0.983057  |
| 17 | d_dy(u[0] * u[0] * u[0],2)   | 0.982041  |
| 31 | d_dx(u[0],4)                 | -1.01657  |
| 32 | d_dy(u[0],4)                 | -1.01647  |
| 35 | dd_dxdy(u[0],(2,2))          | -1.91144  |
```

We find the results also reported in section 3.2.2, which agree very well with the original model. Note that the optimization routine itself has hyperparameters (such as the maximum number of evaluations) that will be passed to the tree-structured Parzen estimator of Hyperopt.

Appendix D. Detailed results for coefficients

In this section we give unabbreviated numerical results for all computations performed throughout section 3, this includes all terms in our ansatz library, their estimated coefficients and the true model coefficients. All library terms that have zero coefficients are listed as 'discarded terms', while terms with any non-zero coefficient for a field variable are listed in a table similar to the numerical output from the tutorial in appendix C, and compared to a table with the true model coefficients.

Table 1. Full results for estimating the reaction diffusion equation (9) from data shown in figure 2 from section 3.1.1. Identical results for both our method (found threshold: $h = 0.081\,199\,193\,5628$) and *PySINDy* (threshold: $h = 0.08$).

Estimated model				True model			
Index	Term	$\partial_t u_1$	$\partial_t u_2$	Index	Term	$\partial_t u_1$	$\partial_t u_2$
1	u_1^1	0.983 454	0	1	u_1^1	1.0	0
2	u_2^1	0	0.983 574	2	u_2^1	0	1.0
6	u_1^3	−0.983 372	−0.999 644	6	u_1^3	−1.0	−1.0
7	$u_1^2 u_2^1$	0.999 632	−0.98 347	7	$u_1^2 u_2^1$	1.0	−1.0
8	$u_1^1 u_2^2$	−0.983 362	−0.99 963	8	$u_1^1 u_2^2$	−1.0	−1.0
9	u_2^3	0.999 646	−0.98 349	9	u_2^3	1.0	−1.0
12	$\partial_y^2(u_1^1)$	0.0982 818	0	12	$\partial_y^2(u_1^1)$	0.1	0
13	$\partial_y^2(u_2^1)$	0	0.0986 993	13	$\partial_y^2(u_2^1)$	0	0.1
18	$\partial_x^2(u_1^1)$	0.0986 863	0	18	$\partial_x^2(u_1^1)$	0.1	0
19	$\partial_x^2(u_2^1)$	0	0.0982 933	19	$\partial_x^2(u_2^1)$	0	0.1

100 Discarded Terms:

1.0, u_1^2, $u_1^1 u_2^1$, u_2^2, $\partial_y^1(u_1^1)$, $\partial_y^1(u_2^1)$, $\partial_x^1(u_1^1)$, $\partial_x^1(u_2^1)$, $\partial_x^1\partial_y^1(u_1^1)$, $\partial_x^1\partial_y^1(u_2^1)$, $u_1^1\partial_y^1(u_1^1)$, $u_2^1\partial_y^1(u_1^1)$, $u_1^2\partial_y^1(u_1^1)$, $u_1^1 u_2^1\partial_y^1(u_1^1)$, $u_2^2\partial_y^1(u_1^1)$, $u_1^3\partial_y^1(u_1^1)$, $u_1^2 u_2^1\partial_y^1(u_1^1)$, $u_1^1 u_2^2\partial_y^1(u_1^1)$, $u_2^3\partial_y^1(u_1^1)$, $u_1^1\partial_y^1(u_2^1)$, $u_2^1\partial_y^1(u_2^1)$, $u_1^2\partial_y^1(u_2^1)$, $u_1^1 u_2^1\partial_y^1(u_2^1)$, $u_2^2\partial_y^1(u_2^1)$, $u_1^3\partial_y^1(u_2^1)$, $u_1^2 u_2^1\partial_y^1(u_2^1)$, $u_1^1 u_2^2\partial_y^1(u_2^1)$, $u_2^3\partial_y^1(u_2^1)$, $u_1^1\partial_y^2(u_1^1)$, $u_2^1\partial_y^2(u_1^1)$, $u_1^2\partial_y^2(u_1^1)$, $u_1^1 u_2^1\partial_y^2(u_1^1)$, $u_2^2\partial_y^2(u_1^1)$, $u_1^3\partial_y^2(u_1^1)$, $u_1^2 u_2^1\partial_y^2(u_1^1)$, $u_1^1 u_2^2\partial_y^2(u_1^1)$, $u_2^3\partial_y^2(u_1^1)$, $u_1^1\partial_y^2(u_2^1)$, $u_2^1\partial_y^2(u_2^1)$, $u_1^2\partial_y^2(u_2^1)$, $u_1^1 u_2^1\partial_y^2(u_2^1)$, $u_2^2\partial_y^2(u_2^1)$, $u_1^3\partial_y^2(u_2^1)$, $u_1^2 u_2^1\partial_y^2(u_2^1)$, $u_1^1 u_2^2\partial_y^2(u_2^1)$, $u_2^3\partial_y^2(u_2^1)$, $u_1^1\partial_x^1(u_1^1)$, $u_2^1\partial_x^1(u_1^1)$, $u_1^2\partial_x^1(u_1^1)$, $u_1^1 u_2^1\partial_x^1(u_1^1)$, $u_2^2\partial_x^1(u_1^1)$, $u_1^3\partial_x^1(u_1^1)$, $u_1^2 u_2^1\partial_x^1(u_1^1)$, $u_1^1 u_2^2\partial_x^1(u_1^1)$, $u_2^3\partial_x^1(u_1^1)$, $u_1^1\partial_x^1(u_2^1)$, $u_2^1\partial_x^1(u_2^1)$, $u_1^2\partial_x^1(u_2^1)$, $u_1^1 u_2^1\partial_x^1(u_2^1)$, $u_2^2\partial_x^1(u_2^1)$, $u_1^3\partial_x^1(u_2^1)$, $u_1^2 u_2^1\partial_x^1(u_2^1)$, $u_1^1 u_2^2\partial_x^1(u_2^1)$, $u_2^3\partial_x^1(u_2^1)$, $u_1^1\partial_x^1\partial_y^1(u_1^1)$, $u_2^1\partial_x^1\partial_y^1(u_1^1)$, $u_1^2\partial_x^1\partial_y^1(u_1^1)$, $u_1^1 u_2^1\partial_x^1\partial_y^1(u_1^1)$, $u_2^2\partial_x^1\partial_y^1(u_1^1)$, $u_1^3\partial_x^1\partial_y^1(u_1^1)$, $u_1^2 u_2^1\partial_x^1\partial_y^1(u_1^1)$, $u_1^1 u_2^2\partial_x^1\partial_y^1(u_1^1)$, $u_2^3\partial_x^1\partial_y^1(u_1^1)$, $u_1^1\partial_x^1\partial_y^1(u_2^1)$, $u_2^1\partial_x^1\partial_y^1(u_2^1)$, $u_1^2\partial_x^1\partial_y^1(u_2^1)$, $u_1^1 u_2^1\partial_x^1\partial_y^1(u_2^1)$, $u_2^2\partial_x^1\partial_y^1(u_2^1)$, $u_1^3\partial_x^1\partial_y^1(u_2^1)$, $u_1^2 u_2^1\partial_x^1\partial_y^1(u_2^1)$, $u_1^1 u_2^2\partial_x^1\partial_y^1(u_2^1)$, $u_2^3\partial_x^1\partial_y^1(u_2^1)$, $u_1^2\partial_x^2(u_1^1)$, $u_2^1\partial_x^2(u_1^1)$, $u_1^3\partial_x^2(u_1^1)$, $u_1^2 u_2^1\partial_x^2(u_1^1)$, $u_1^1 u_2^2\partial_x^2(u_1^1)$, $u_2^3\partial_x^2(u_1^1)$, $u_1^1\partial_x^2(u_2^1)$, $u_2^1\partial_x^2(u_2^1)$, $u_1^2\partial_x^2(u_2^1)$, $u_1^1 u_2^1\partial_x^2(u_2^1)$, $u_2^2\partial_x^2(u_2^1)$, $u_1^3\partial_x^2(u_2^1)$, $u_1^2 u_2^1\partial_x^2(u_2^1)$, $u_1^1 u_2^2\partial_x^2(u_2^1)$, $u_2^3\partial_x^2(u_2^1)$

Table 2. Full results for estimating the Allen–Cahn equation (14) from data shown in figure 5 from section 3.2.1. Found hyperparameter (threshold): $h_1 = 0.330\,099\,846\,257\,6791$.

Estimated model			True model		
Index	Term	$\partial_t u_1$	Index	Term	$\partial_t u_1$
1	u_0^1	0.98 642	1	u_0^1	1
3	u_0^3	−0.986 013	3	u_0^3	−1
10	$\partial_x^2(u_0^1)$	0.987 685	10	$\partial_x^2(u_0^1)$	1
11	$\partial_y^2(u_0^1)$	0.987 623	11	$\partial_y^2(u_0^1)$	1

42 Discarded Terms:

1.0, u_0^1, $\partial_x^1(u_0^1)$, $\partial_y^1(u_0^1)$, $\partial_x^1(u_0^2)$, $\partial_y^1(u_0^2)$, $\partial_x^1(u_0^3)$, $\partial_y^1(u_0^3)$, $\partial_x^1\partial_y^1(u_0^1)$, $\partial_x^2(u_0^2)$, $\partial_y^2(u_0^2)$, $\partial_x^1\partial_y^1(u_0^2)$, $\partial_x^2(u_0^3)$, $\partial_y^2(u_0^3)$, $\partial_x^1\partial_y^1(u_0^3)$, $\partial_x^3(u_0^1)$, $\partial_y^3(u_0^1)$, $\partial_x^1\partial_y^2(u_0^1)$, $\partial_x^2\partial_y^1(u_0^1)$, $\partial_x^3(u_0^2)$, $\partial_y^3(u_0^2)$, $\partial_x^1\partial_y^2(u_0^2)$, $\partial_x^2\partial_y^1(u_0^2)$, $\partial_x^3(u_0^3)$, $\partial_y^3(u_0^3)$, $\partial_x^1\partial_y^2(u_0^3)$, $\partial_x^2\partial_y^1(u_0^3)$, $\partial_x^4(u_0^1)$, $\partial_y^4(u_0^1)$, $\partial_x^1\partial_y^3(u_0^1)$, $\partial_x^3\partial_y^1(u_0^1)$, $\partial_x^2\partial_y^2(u_0^1)$, $\partial_x^4(u_0^2)$, $\partial_y^4(u_0^2)$, $\partial_x^1\partial_y^3(u_0^2)$, $\partial_x^3\partial_y^1(u_0^2)$, $\partial_x^2\partial_y^2(u_0^2)$, $\partial_x^4(u_0^3)$, $\partial_y^4(u_0^3)$, $\partial_x^1\partial_y^3(u_0^3)$, $\partial_x^3\partial_y^1(u_0^3)$, $\partial_x^2\partial_y^2(u_0^3)$

Table 3. Full results for estimating the Cahn–Hilliard equation (17) from data shown in figure 6 from section 3.2.2. Found hyperparameter (threshold): $h_1 = 0.330\,035\,457\,717\,988\,15$.

Estimated model		
Index	Term	$\partial_t u_1$
10	$\partial_x^2\left(u_1^1\right)$	−0.986 104
11	$\partial_y^2\left(u_1^1\right)$	−0.989 164
16	$\partial_x^2\left(u_1^3\right)$	0.977 613
17	$\partial_y^2\left(u_1^3\right)$	0.985 195
31	$\partial_x^4\left(u_1^1\right)$	−1.02 871
32	$\partial_y^4\left(u_1^1\right)$	−1.0264
35	$\partial_x^2\partial_y^2\left(u_1^1\right)$	-1.91 485

True model		
Index	Term	$\partial_t u_1$
10	$\partial_x^2\left(u_1^1\right)$	−1.0
11	$\partial_y^2\left(u_1^1\right)$	−1.0
16	$\partial_x^2\left(u_1^3\right)$	1.0
17	$\partial_y^2\left(u_1^3\right)$	1.0
31	$\partial_x^4\left(u_1^1\right)$	−1.0
32	$\partial_y^4\left(u_1^1\right)$	−1.0
35	$\partial_x^2\partial_y^2\left(u_1^1\right)$	-2.0

39 Discarded Terms:

$1.0, u_1^1, u_2^1, u_1^2, u_1^1u_2^1, u_2^2, u_1^3, u_1^1u_2^2, u_1^2u_2^1, u_2^3, \partial_x^1\left(u_2^1\right), \partial_y^1\left(u_2^1\right), \partial_x^1\left(u_1^2\right), \partial_y^1\left(u_1^2\right), \partial_x^1\left(u_2^2\right), \partial_y^1\left(u_2^2\right), \partial_x^1\left(u_1^3\right), \partial_y^1\left(u_1^3\right), \partial_x^1\left(u_1^1u_2^2\right), \partial_y^1\left(u_1^1u_2^2\right), \partial_x^1\left(u_1^2u_2^1\right), \partial_y^1\left(u_1^2u_2^1\right), \partial_x^1\left(u_2^3\right), \partial_y^1\left(u_2^3\right), \partial_x^2\left(u_1^1\right), \partial_y^2\left(u_1^1\right), \partial_x^1\partial_y^1\left(u_1^1\right), \partial_x^1\partial_y^1\left(u_2^1\right), \partial_x^2\left(u_1^2\right), \partial_x^1\partial_y^1\left(u_1^2\right), \partial_x^2\left(u_1^1u_2^1\right), \partial_y^2\left(u_1^1u_2^1\right), \partial_x^1\partial_y^1\left(u_1^1u_2^1\right), \partial_x^2\left(u_2^2\right), \partial_y^2\left(u_2^2\right), \partial_x^1\partial_y^1\left(u_2^2\right), \partial_x^2\left(u_1^3\right), \partial_y^2\left(u_1^3\right), \partial_x^1\partial_y^1\left(u_1^3\right)$

Table 4. Full results for estimating the FitzHugh–Nagumo equation (21) from data shown in figure 7 from section 3.3.1. Found hyperparameter (thresholds): $h_1 = 0.066\,428\,028\,399\,326\,97$ for all terms for $\partial_t u_1$ and $h_2 = 0.013\,702\,806\,950\,437\,454$ for all terms for $\partial_t u_2$.

Estimated model			
Index	Term	$\partial_t u_1$	$\partial_t u_2$
1	u_1^1	0.485 341	0.066 4892
2	u_2^1	−0.493 396	−0.066 4834
6	u_1^3	−0.965 856	0
28	$\partial_x^2\left(u_1^1\right)$	0.242 025	0
29	$\partial_y^2\left(u_1^1\right)$	0.242 146	0
31	$\partial_x^2\left(u_2^1\right)$	0	0.016 7489
32	$\partial_y^2\left(u_2^1\right)$	0	0.016 7483

True model			
Index	Term	$\partial_t u_1$	$\partial_t u_2$
1	u_1^1	0.5	0.066
2	u_2^1	−0.5	−0.066
6	u_1^3	−1.0	0
28	$\partial_x^2\left(u_1^1\right)$	0.5	0
29	$\partial_y^2\left(u_1^1\right)$	0.5	0
31	$\partial_x^2\left(u_2^1\right)$	0	0.0166
32	$\partial_y^2\left(u_2^1\right)$	0	0.0166

48 Discarded Terms:

$1.0, u_1^2, u_1^1u_2^1, u_2^2, u_1^1u_2^2, u_1^2u_2^1, u_2^3, \partial_x^1\left(u_1^1\right), \partial_y^1\left(u_1^1\right), \partial_x^1\left(u_2^1\right), \partial_y^1\left(u_2^1\right), \partial_x^1\left(u_1^2\right), \partial_y^1\left(u_1^2\right), \partial_x^1\left(u_1^1u_2^1\right), \partial_y^1\left(u_1^1u_2^1\right), \partial_x^1\left(u_2^2\right), \partial_y^1\left(u_2^2\right), \partial_x^1\left(u_1^3\right), \partial_y^1\left(u_1^3\right), \partial_x^1\left(u_1^1u_2^2\right), \partial_y^1\left(u_1^1u_2^2\right), \partial_x^1\left(u_1^2u_2^1\right), \partial_y^1\left(u_1^2u_2^1\right), \partial_x^1\left(u_2^3\right), \partial_y^1\left(u_2^3\right), \partial_x^1\partial_y^1\left(u_1^1\right), \partial_x^1\partial_y^1\left(u_2^1\right), \partial_x^2\left(u_1^2\right), \partial_y^2\left(u_1^2\right), \partial_x^1\partial_y^1\left(u_1^2\right), \partial_x^2\left(u_1^1u_2^1\right), \partial_y^2\left(u_1^1u_2^1\right), \partial_x^1\partial_y^1\left(u_1^1u_2^1\right), \partial_x^2\left(u_2^2\right), \partial_y^2\left(u_2^2\right), \partial_x^1\partial_y^1\left(u_2^2\right), \partial_x^2\left(u_1^3\right), \partial_y^2\left(u_1^3\right), \partial_x^1\partial_y^1\left(u_1^3\right), \partial_x^2\left(u_1^1u_2^2\right), \partial_y^2\left(u_1^1u_2^2\right), \partial_x^1\partial_y^1\left(u_1^1u_2^2\right), \partial_x^2\left(u_1^2u_2^1\right), \partial_y^2\left(u_1^2u_2^1\right), \partial_x^1\partial_y^1\left(u_1^2u_2^1\right), \partial_x^2\left(u_2^3\right), \partial_y^2\left(u_2^3\right), \partial_x^1\partial_y^1\left(u_2^3\right)$

Table 5. Full results for estimating the reaction diffusion equations (23) from data shown in figure 8 from section 3.3.2. Found hyperparameters (thresholds) are: $h_1 = 0.306\,675\,543\,669\,0512$ for all terms for $\partial_t u_1$ and $h_2 = 0.177\,583\,539\,319\,337\,46$ for all terms for $\partial_t u_2$.

Estimated model			
Index	Term	$\partial_t u_1$	$\partial_t u_2$
1	u_1^1	0.999 621	0
2	u_2^1	0	0.999 654
6	u_1^3	−0.999 402	3.999 25
7	$u_1^2 u_1^1$	−3.999 19	−0.999 489
8	$u_1^1 u_1^2$	−0.999 413	3.999 25
9	u_2^3	−3.999 18	−0.999 47
28	$\partial_x^2 (u_1^1)$	0.977 855	1.999 41
29	$\partial_y^2 (u_1^1)$	0.983 671	1.9994
31	$\partial_x^2 (u_2^1)$	−1.999 34	0.974 819
32	$\partial_y^2 (u_2^1)$	−1.999 35	0.982 875

True model			
Index	Term	$\partial_t u_1$	$\partial_t u_2$
1	u_1^1	1.0	0
2	u_2^1	0	1.0
6	u_1^3	−1.0	4.0
7	$u_1^2 u_1^1$	−4.0	−1.0
8	$u_1^1 u_1^2$	−1.0	4.0
9	u_2^3	−4.0	−1.0
28	$\partial_x^2 (u_1^1)$	1.0	2.0
29	$\partial_y^2 (u_1^1)$	1.0	2.0
31	$\partial_x^2 (u_2^1)$	−2.0	1.0
32	$\partial_y^2 (u_2^1)$	−2.0	1.0

45 Discarded Terms:

$1.0, u_1^2, u_1^1 u_2^1, u_2^2, \partial_x^1 (u_1^1), \partial_y^1 (u_1^1), \partial_x^1 (u_2^1), \partial_y^1 (u_2^1), \partial_x^1 (u_1^2), \partial_y^1 (u_1^2), \partial_x^1 (u_1^1 u_2^1), \partial_y^1 (u_1^1 u_2^1), \partial_x^1 (u_2^2), \partial_y^1 (u_2^2), \partial_x^1 (u_1^3), \partial_y^1 (u_1^3), \partial_x^1 (u_1^1 u_2^2), \partial_y^1 (u_1^1 u_2^2), \partial_x^1 (u_1^2 u_2^1), \partial_y^1 (u_1^2 u_2^1), \partial_x^1 (u_2^3), \partial_y^1 (u_2^3), \partial_x^1 \partial_y^1 (u_1^1), \partial_x^1 \partial_y^1 (u_1^2), \partial_x^2 (u_1^2), \partial_y^2 (u_1^2), \partial_x^1 \partial_y^1 (u_1^2), \partial_x^2 (u_1^1 u_2^1), \partial_y^2 (u_1^1 u_2^1), \partial_x^1 \partial_y^1 (u_1^1 u_2^1), \partial_x^2 (u_2^2), \partial_y^2 (u_2^2), \partial_x^1 \partial_y^1 (u_2^2), \partial_x^2 (u_1^3), \partial_y^2 (u_1^3), \partial_x^1 \partial_y^1 (u_1^3), \partial_x^2 (u_1^1 u_2^2), \partial_y^2 (u_1^1 u_2^2), \partial_x^1 \partial_y^1 (u_1^1 u_2^2), \partial_x^2 (u_1^2 u_2^1), \partial_y^2 (u_1^2 u_2^1), \partial_x^1 \partial_y^1 (u_1^2 u_2^1), \partial_x^2 (u_2^3), \partial_y^2 (u_2^3), \partial_x^1 \partial_y^1 (u_2^3)$

Table 6. Full results for estimating reaction diffusion equations (23) from data shown in figure 9 from section 3.3.2 without additional thresholds. Found hyperparameters (thresholds): $h_1 = 0.306\,675\,543\,669\,0512$ for all terms for $\partial_t u_1$ and $h_2 = 0.177\,583\,539\,319\,337\,46$ for all terms for $\partial_t u_2$.

Estimated model			
Index	Term	$\partial_t u_1$	$\partial_t u_2$
1	u_1^1	0.977 683	0
2	u_2^1	0	0.979 197
6	u_1^3	−0.998 367	3.996 46
7	$u_2^1 u_0^2$	−3.996 25	−0.980 391
8	$u_2^2 u_0^1$	−0.979 247	3.998 25
9	u_2^3	−3.996 15	−1.000 44
28	$\partial_x^2 (u_1^1)$	0.978 702	0
29	$\partial_y^2 (u_1^1)$	0.961 488	0
31	$\partial_x^2 (u_2^1)$	0	0.979 449
32	$\partial_y^2 (u_2^1)$	0	0.966 95
44	$\partial_y^2 (u_1^3)$	−0.146 276	0
47	$\partial_y^2 (u_2^1 u_0^2)$	0	−0.122 473
50	$\partial_y^2 (u_2^2 u_0^1)$	−0.119 995	0
53	$\partial_y^2 (u_2^3)$	0	−0.154 368

True model			
Index	Term	$\partial_t u_1$	$\partial_t u_2$
1	u_1^1	1.0	0
2	u_2^1	0	1.0
6	u_1^3	−1.0	4.0
7	$u_1^2 u_1^1$	−4.0	−1.0
8	$u_1^1 u_1^2$	−1.0	4.0
9	u_2^3	-4.0	-1.0
28	$\partial_x^2 (u_1^1)$	1.0	0
29	$\partial_y^2 (u_1^1)$	1.0	0
31	$\partial_x^2 (u_2^1)$	0	1.0
32	$\partial_y^2 (u_2^1)$	0	1.0

41 Discarded Terms:

$1.0, u_1^2, u_1^1 u_2^1, u_2^2, \partial_x^1 (u_1^1), \partial_y^1 (u_1^1), \partial_x^1 (u_2^1), \partial_y^1 (u_2^1), \partial_x^1 (u_1^2), \partial_y^1 (u_1^2), \partial_x^1 (u_1^1 u_2^1), \partial_y^1 (u_1^1 u_2^1), \partial_x^1 (u_2^2), \partial_y^1 (u_2^2), \partial_x^1 (u_1^3), \partial_y^1 (u_1^3), \partial_x^1 (u_1^1 u_2^2), \partial_y^1 (u_1^1 u_2^2), \partial_x^1 (u_1^2 u_2^1), \partial_y^1 (u_1^2 u_2^1), \partial_x^1 (u_2^3), \partial_y^1 (u_2^3), \partial_x^1 \partial_y^1 (u_1^1), \partial_x^1 \partial_y^1 (u_1^2), \partial_x^2 (u_1^2), \partial_y^2 (u_1^2), \partial_x^1 \partial_y^1 (u_1^2), \partial_x^2 (u_1^1 u_2^1), \partial_y^2 (u_1^1 u_2^1), \partial_x^1 \partial_y^1 (u_1^1 u_2^1), \partial_x^2 (u_2^2), \partial_y^2 (u_2^2), \partial_x^1 \partial_y^1 (u_2^2), \partial_x^2 (u_1^3), \partial_x^1 \partial_y^1 (u_1^3), \partial_x^2 (u_1^1 u_2^2), \partial_x^1 \partial_y^1 (u_1^1 u_2^2), \partial_x^2 (u_1^2 u_2^1), \partial_x^1 \partial_y^1 (u_1^2 u_2^1), \partial_x^2 (u_2^3), \partial_x^1 \partial_y^1 (u_2^3)$

Table 7. Full results for estimating reaction diffusion equations (23) from data shown in figure 9 from section 3.3.2 with additional thresholds for the terms $\partial_x^2 u_1, \partial_y^2 u_0$ and $\partial_x^2 u_2, \partial_y^2 u_1$. Found hyperparameters (thresholds): $h_1 = 0.3066755436690512$ for all terms for $\partial_t u_1$ except indices 28 and 29, $h_2 = 0.177\,583\,539\,319,337\,46$ for all terms for $\partial_t u_2$ except indices 31 and 32, and $h_{\text{sub}1} = 0.005\,074\,209\,512\,766\,352$ for terms with indices 28 and 29 for $\partial_t u_1$, $h_{\text{sub}2} = 0.000\,669\,976\,353\,564\,3887$ for terms with indices 31 and 32 for $\partial_t u_2$.

Estimated model				True model			
Index	Term	$\partial_t u_1$	$\partial_t u_2$	Index	Term	$\partial_t u_1$	$\partial_t u_2$
1	u_1^1	0.956 482	0	1	u_1^1	1.0	0
2	u_2^1	0	0.957 13	2	u_2^1	0	1.0
6	u_1^3	−0.959 624	3.992 63	6	u_1^3	−1.0	4.0
7	$u_1^2 u_1^1$	−3.991 89	−0.960 558	7	$u_1^2 u_1^1$	−4.0	−1.0
8	$u_1^1 u_1^2$	−0.959 277	3.992 66	8	$u_1^1 u_1^2$	−1.0	4.0
9	u_2^3	−3.9922	−0.959 894	9	u_2^3	−4.0	−1.0
28	$\partial_x^2 (u_1^1)$	0.974 305	0	28	$\partial_x^2 (u_1^1)$	1.0	0
29	$\partial_y^2 (u_1^1)$	0.896 47	0	29	$\partial_y^2 (u_1^1)$	1.0	0
31	$\partial_x^2 (u_2^1)$	0	0.974 735	31	$\partial_x^2 (u_2^1)$	0	1.0
32	$\partial_y^2 (u_2^1)$	0	0.898 804	32	$\partial_y^2 (u_2^1)$	0	1.0

45 Discarded Terms:

$1.0, u_1^2, u_1^1 u_2^1, u_2^2, \partial_x^1 (u_1^1), \partial_y^1 (u_1^1), \partial_x^1 (u_2^1), \partial_y^1 (u_2^1), \partial_x^1 (u_1^2), \partial_y^1 (u_1^2), \partial_x^1 (u_1^1 u_2^1), \partial_y^1 (u_1^1 u_2^1), \partial_x^1 (u_2^2), \partial_y^1 (u_2^2), \partial_x^1 (u_1^3), \partial_y^1 (u_1^3), \partial_x^1 (u_1^1 u_2^2), \partial_y^1 (u_1^1 u_2^2), \partial_x^1 (u_1^2 u_2^1), \partial_y^1 (u_1^2 u_2^1), \partial_x^1 (u_2^3), \partial_y^1 (u_2^3), \partial_x^1 \partial_y^1 (u_1^1), \partial_x^1 \partial_y^1 (u_2^1), \partial_x^2 (u_1^2), \partial_y^2 (u_1^2), \partial_x^1 \partial_y^1 (u_1^2), \partial_x^2 (u_1^1 u_2^1), \partial_y^2 (u_1^1 u_2^1), \partial_x^1 \partial_y^1 (u_1^1 u_2^1), \partial_x^2 (u_2^2), \partial_y^2 (u_2^2), \partial_x^1 \partial_y^1 (u_2^2), \partial_x^2 (u_1^3), \partial_y^2 (u_1^3), \partial_x^1 \partial_y^1 (u_1^3), \partial_x^2 (u_1^1 u_2^2), \partial_y^2 (u_1^1 u_2^2), \partial_x^1 \partial_y^1 (u_1^1 u_2^2), \partial_x^2 (u_1^2 u_2^1), \partial_y^2 (u_1^2 u_2^1), \partial_x^1 \partial_y^1 (u_1^2 u_2^1), \partial_x^2 (u_2^3), \partial_y^2 (u_2^3), \partial_x^1 \partial_y^1 (u_2^3)$

Table 8. Full results for estimating the time-delayed Fisher-KPP equation (25) from data shown in figure 10 from section 3.3.3. Found hyperparameter (threshold and time-delay): $h_1 = 0.623\,871\,336\,231\,7317$ and $\tau = 1.017\,246\,252\,677\,2437$.

Estimated model			True model ($\tau = 1.0$)		
Index	Term	$\partial_t u_1$	Index	Term	$\partial_t u_1$
1	u_1^1	0.976 494	1	u_1^1	1.0
7	$\partial_x^2 (u_1^1)$	1.018 92	7	$\partial_x^2 (u_1^1)$	1.0
17	$u_1^1 u_{\tau 1}^1$	−0.979 02	17	$u_1^1 u_{\tau 1}^1$	−1.0

20 Discarded Terms:

$1.0, u_1^2, u_1^3, \partial_x^1 (u_1^1), \partial_x^1 (u_1^2), \partial_x^1 (u_1^3), \partial_x^2 (u_1^2), \partial_x^2 (u_1^3), \partial_x^3 (u_1^1), \partial_x^3 (u_1^2), \partial_x^3 (u_1^3), \partial_x^4 (u_1^1), \partial_x^4 (u_1^2), \partial_x^4 (u_1^3), u_{\tau 1}^1, u_1^2 u_{\tau 1}^1, \partial_x^1 (u_{\tau 1}^1), \partial_x^2 (u_{\tau 1}^1), \partial_x^3 (u_{\tau 1}^1), \partial_x^4 (u_{\tau 1}^1)$

ORCID iDs

Oliver Mai 0009-0007-5864-7674
Uwe Thiele 0000-0001-7989-9271
Oliver Kamps 0000-0003-0986-0878

References

[1] Bock H G and Plitt K J 1984 A multiple shooting algorithm for direct solution of optimal control problems *IFAC Proc. Volumes* vol 17 pp 1603–8
[2] Guay M and Mclean D D 1995 Optimization and sensitivity analysis for multiresponse parameter estimation in systems of ordinary differential equations *Comput. Chem. Eng.* **19** 1271–85
[3] Houska B, Logist F, Diehl M and Impe J V 2012 A tutorial on numerical methods for state and parameter estimation in nonlinear dynamic systems *Identification for Automotive Systems* ed D Alberer, H Hjalmarsson and L Re (*Lecture Notes in Control and Information Sciences* vol 418) (Springer) pp 67–88
[4] Brunton S L, Proctor J L and Kutz J N 2016 Discovering governing equations from data by sparse identification of nonlinear dynamical systems *Proc. Natl Acad. Sci.* **113** 3932–7
[5] Egan K, Li W and Carvalho R 2024 Automatically discovering ordinary differential equations from data with sparse regression *Commun. Phys.* **7** 2
[6] Omejc N, Gec B, Brence J, Todorovski L and Džeroski S 2024 Probabilistic grammars for modeling dynamical systems from coarse, noisy and partial data *Mach. Learn.* **113** 7689–721
[7] Bär M, Hegger R and Kantz H 1999 Fitting partial differential equations to space-time dynamics *Phys. Rev. E* **59** 337–42

[8] Ljung L 1998 *System Identification: Theory for the User Prentice Hall Information and System Sciences Series* 2nd edn (Prentice Hall)
[9] Voss H U, Kolodner P, Abel M and Kurths J 1999 Amplitude equations from spatiotemporal binary-fluid convection data *Phys. Rev. Lett.* **83** 3422–5
[10] Voss H, Bünner M and Abel M 1998 Identification of continuous, spatiotemporal systems *Phys. Rev. E* **57** 2820–3
[11] Rudy S H, Brunton S L, Proctor J L and Kutz J N 2017 Data-driven discovery of partial differential equations *Sci. Adv.* **3** 614
[12] Voss H U, Timmer J and Kurths J 2004 Nonlinear dynamical system identification from uncertain and indirect measurements *Int. J. Bifur. Chaos* **14** 1905–33
[13] Wang Z, Huan X and Garikipati K 2019 Variational system identification of the partial differential equations governing the physics of pattern-formation: inference under varying fidelity and noise *Comput. Methods Appl. Mech. Eng.* **356** 44–74
[14] Zhao H, Storey B D, Braatz R D and Bazant M Z 2020 Learning the physics of pattern formation from images *Phys. Rev. Lett.* **124** 060201
[15] Naozuka G T, Rocha H L, Silva R S and Almeida R C 2022 SINDy-SA framework: enhancing nonlinear system identification with sensitivity analysis *Nonlinear Dyn.* **110** 2589–609
[16] Mangan N M, Kutz J N, Brunton S L and Proctor J L 2017 Model selection for dynamical systems via sparse regression and information criteria *Proc. R. Soc. A* **473** 20170009
[17] Dong X, Bai Y, Lu Y and Fan M 2022 An improved sparse identification of nonlinear dynamics with akaike information criterion and group sparsity *Nonlinear Dyn.* **111** 1485–510
[18] Rudy S, Alla A, Brunton S L and Kutz J N 2019 Data-driven identification of parametric partial differential equations *SIAM J. Appl. Dyn. Syst.* **18** 643–60
[19] Long Z, Lu Y and Dong B 2019 PDE-Net 2.0: learning PDEs from data with a numeric-symbolic hybrid deep network *J. Comput. Phys.* **399** 108925
[20] Fullana J, Rossi M and Zaleski S 1997 Parameter identification in noisy extended systems: a hydrodynamic case *Physica D* **103** 564–75
[21] Messenger D A and Bortz D M 2021 Weak SINDy: Galerkin-based data-driven model selection *Multiscale Model. Simul.* **19** 1474–97
[22] Tang M, Liao W, Kuske R and Kang S H 2023 Weakident: weak formulation for identifying differential equation using narrow-fit and trimming *J. Comput. Phys.* **483** 112069
[23] Sandoz A, Ducret V, Gottwald G A, Vilmart G and Perron K 2023 SINDy for delay-differential equations: application to model bacterial zinc response *Proc. R. Soc. A* **479** 56
[24] Stephany R 2024 DDE-Find: Learning delay differential equations from noisy, limited data (arXiv:2405.02661)
[25] Zubov K *et al* 2021 NeuralPDE: automating physics-informed neural networks (PINNs) with error approximations (arXiv:2107.09443)
[26] Maddu S, Cheeseman B L, Sbalzarini I F and Müller C L 2022 Stability selection enables robust learning of differential equations from limited noisy data *Proc. R. Soc. A* **478** 0916
[27] Koch J 2021 Data-driven modeling of nonlinear traveling waves *Chaos* **31** 255
[28] Wan C, Yu Z, Wang F, Liu X and Li J 2021 Experiment data-driven modeling of tokamak discharge in EAST *Nucl. Fusion* **61** 066015
[29] Zhao J, Wang W and Sheng C 2018 Parameter estimation and optimization *Data-Driven Prediction for Industrial Processes and Their Applications* (*Information Fusion and Data Science* vol 1) (Springer) pp 269–350
[30] Machlanski D, Samothrakis S and Clarke P 2023 The challenges of hyperparameter tuning for accurate causal effect estimation (arXiv:2303.01412)
[31] Kroll T W and Kamps O 2025 Sparse identification of evolution equations via Bayesian model selection ((arXiv:2501.01476)
[32] Harris C R *et al* 2020 Array programming with NumPy *Nature* **585** 357–62
[33] Baer M 2018 findiff software package (available at: https://github.com/maroba/findiff)
[34] Bergstra J, Komer B, Eliasmith C, Yamins D and Cox D D 2015 Hyperopt: a Python library for model selection and hyperparameter optimization *Comput. Sci. Discov.* **8** 014008
[35] Bergstra J, Bardenet R, Bengio Y and Kégl B 2011 Algorithms for hyper-parameter optimization *Advances in Neural Information Processing Systems* vol 24, ed J Shawe-Taylor, R Zemel, P Bartlett, F Pereira and K Q Weinberger (Curran Associates, Inc)
[36] Virtanen P *et al* 2020 SciPy 1.0: fundamental algorithms for scientific computing in Python *Nat. Methods* **17** 261–72
[37] Hairer E, Norsett S P and Wanner G 1993 *Solving Ordinary Differential Equations I (Springer Series in Computational Mathematics)* 2nd edn (Springer) (https://doi.org/10.1007/978-3-540-78862-1)
[38] Mai O 2024 TSME PyPi repository (available at: https://pypi.org/project/tsme/)
[39] Mai O 2025 TSME examples GitHub repository (available at: https://github.com/CDSC-CoSyML/tsme_examples)
[40] De Silva B, Champion K, Quade M, Loiseau J, Kutz J and Brunton S 2020 PySINDy: a Python package for the sparse identification of nonlinear dynamical systems from data *J. Open Source Softw.* **5** 2104
[41] Cahn J W and Novick-Cohen A 1994 Evolution equations for phase separation and ordering in binary alloys *J. Stat. Phys.* **76** 877–909
[42] Hussain S, Shah A, Ayub S and Ullah A 2019 An approximate analytical solution of the Allen-Cahn equation using homotopy perturbation method and homotopy analysis method *Heliyon* **5** e03060
[43] Stegemerten F, Gurevich S V and Thiele U 2020 Bifurcations of front motion in passive and active Allen–Cahn-type equations *Chaos* **30** 3271
[44] Bandeira J G P, Buske D, De Quadros R S and Kurz G B 2024 Allen-Cahn equation for modeling temporal evolution of non-conserved field variables in cancer cell migration *Ciênc. Nat.* **46** e87268
[45] Kim J, Lee S, Choi Y, Lee S and Jeong D 2016 Basic principles and practical applications of the Cahn–Hilliard equation *Math. Problems Eng.* **2016** 1–11
[46] Brunk A, Egger H and Habrich O 2023 On uniqueness and stable estimation of multiple parameters in the Cahn–Hilliard equation *Inverse Problems* **39** 065002
[47] Craster R V and Matar O K 2009 Dynamics and stability of thin liquid films *Rev. Mod. Phys.* **81** 1131–98
[48] FitzHugh R 1961 Impulses and physiological states in theoretical models of nerve membrane *Biophys. J.* **1** 445–66
[49] Nagumo J, Arimoto S and Yoshizawa S 1962 An active pulse transmission line simulating nerve axon *Proc. IRE* **50** 2061–70
[50] Zheng Q and Shen J 2015 Pattern formation in the FitzHugh–Nagumo model *Comput. Math. Appl.* **70** 1082–97
[51] Cebrián-Lacasa D, Parra-Rivas P, Ruiz-Reynés D and Gelens L 2024 Six decades of the FitzHugh–Nagumo model: a guide through its spatio-temporal dynamics and influence across disciplines *Phys. Rep.* **1096** 1–39

[52] Hodgkin A L and Huxley A F 1952 A quantitative description of membrane current and its application to conduction and excitation in nerve *J. Physiol.* **117** 500–44
[53] Dong X and Wang C 2015 Identification of the FitzHugh–Nagumo model dynamics via deterministic learning *Int. J. Bifurc. Chaos* **25** 1550159
[54] Ahmed S E, San O and Lakshmivarahan S 2022 Forward sensitivity analysis of the FitzHugh–Nagumo system: parameter estimation *Advances in Nonlinear Dynamics NODYCON Conf.Proc. Series* vol 3 ed W Lacarbonara, B Balachandran, M J Leamy, J Ma, J A T Machado and G Stepan (Springer) pp 93–103
[55] Shraiman B I, Pumir A, Van Saarloos W, Hohenberg P C, Chaté H and Holen M 1992 Spatiotemporal chaos in the one-dimensional complex Ginzburg-Landau equation *Physica D* **57** 241–8
[56] Fisher R A 1937 The wave of advance of advantageous genes *Ann. Eugen.* **7** 355–69
[57] Kolmogorov A N, Petrovskii I G and Piskunov N S 1937 A study of the diffusion equation with increase in the amount of substance and its application to a biological problem *Bull. Moscow Univ. Math. Mech* **1** 1–26
[58] Ducrot A and Nadin G 2014 Asymptotic behaviour of travelling waves for the delayed Fisher–KPP equation *J. Differ. Equ.* **256** 3115–40
[59] Zhang J, Hu H and Huang C 2025 Wave fronts for a class of delayed Fisher–KPP equations *Appl. Math. Lett.* **163** 109406
[60] Fasel U, Kutz J N, Brunton B W and Brunton S L 2022 Ensemble-SINDy: robust sparse model discovery in the low-data, high-noise limit, with active learning and control *Proc. R. Soc. A* **478** 904
[61] Mai O 2022 TSME documentation: Cahn-Hilliard equation (available at: https://nonlinear-physics.zivgitlabpages.uni-muenster.de/ag-kamps/tsme/source/notebooks/cahn_hilliard_equation.html)