Equation Discovery with Bayesian Spike-and-Slab Priors and Efficient Kernels

Da Long
The University of Utah

Wei Xing
The University of Sheffield

Aditi Krishnapriyan
University of California, Berkeley

Robert M. Kirby
The University of Utah

Shandian Zhe*
The University of Utah

Michael W. Mahoney
University of California, Berkeley
Lawrence Berkeley National Laboratory
International Computer Science Institute

Abstract

Discovering governing equations from data is important to many scientific and engineering applications. Despite promising successes, existing methods are still challenged by data sparsity as well as noise issues, both of which are ubiquitous in practice. Moreover, state-of-the-art methods lack uncertainty quantification and/or are costly in training. To overcome these limitations, we propose a novel equation discovery method based on Kernel learning and BAyesian Spike-and-Slab priors (KBASS). We use kernel regression to estimate the target function, which is flexible, expressive, and more robust to data sparsity and noises. We combine it with a Bayesian spike-and-slab prior — an ideal Bayesian sparse distribution — for effective operator selection and uncertainty quantification. We develop an expectation-propagation expectation-maximization (EP-EM) algorithm for efficient posterior inference and function estimation. To overcome the computational challenge of kernel regression, we place the function values on a mesh and induce a Kronecker product construction, and we use tensor algebra to enable efficient computation and optimization. We show the advantages of KBASS on a list of benchmark ODE and PDE discovery tasks.

Proceedings of the 27[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

1 Introduction

Many scientific and engineering disciplines use differential equations to model systems of interest. These equations not only allow accurate predictions (by numerical solvers), but they also provide interpretation of the mechanism, *i.e.*, physical laws. However, for many realistic systems, it is difficult in practice to write down a full set of partial or ordinary differential equations (PDEs/ODEs). Hence, using data-driven machine learning methods to help discover underlying physical equations is a promising approach to advance our understanding of these systems and to develop powerful, reliable predictive and analysis tools.

There have been several approaches for equation discovery, including: Sparse Identification of Nonlinear Dynamics (SINDy) (Brunton et al., 2016) and extensions (Rudy et al., 2017; Schaeffer, 2017; Zhang and Ma, 2020; Lagergren et al., 2020); physics-informed neural networks (PINNs) (Raissi et al., 2019) with L_1 regularization (Berg and Nyström, 2019; Both et al., 2021); PINN using alternating direction optimization to identify equation operators for symbolic regression (PINN-SR) (Chen et al., 2021b); and the recent work (Sun et al., 2022) that uses Bayesian spline learning (BSL) to estimate the target function and relevant vector machine (Tipping, 2001) to select the candidate operators.

Despite promising successes, the existing methods still have challenges with data sparsity and noise. In addition, for practical usage, they often lack uncertainty calibration and/or are costly in training. For example, SINDy uses numerical differentiation to evaluate the derivatives of candidate operators, and it can easily fail on sparsely sampled data. While PINN-based approaches avoid numerical differentiation by using neural networks (NNs) to approximate the target function,

they need a careful choice of the network architectures and tedious tuning of many hyperparameters, and they have by-now well-known failure modes (Krishnapriyan et al., 2021). Applying differentiation operators on NNs further complicates the loss and makes the training expensive (Krishnapriyan et al., 2021; Mojgani et al., 2022). Nonetheless, PINN-based methods typically underperform in ODE discovery tasks (Sun et al., 2022). In addition, both SINDy and PINN methods lack uncertainty quantification of the learned equations. Although the most recent BSL method comes with uncertainty estimation, it is quite costly to learn the spline coefficients, especially in higher dimensions.

To promote the performance of data-driven equation discovery and to be more amenable for practical usage, we propose KBASS, a novel method based on Kernel learning and BAyesian Spike-and-Slab priors. The major contributions of our work are as follows.

- **Model**. We use kernel regression to estimate the target function and its derivatives, an approach that is flexible, expressive, and more robust to data sparsity and noises; and we combine it with a Bayesian spike-and-slab prior to select equation operators and to estimat operator weights. As a gold standard in Bayesian sparse learning, the spike-and-slab prior not only enjoys nice theoretical properties, such as selective shrinkage (Ishwaran and Rao, 2005), but it also enables posterior inference of selection indicators, with which we can decide the selection results and avoid hard thresholding over the weight values. Tuning the weight threshold (as in existing methods) can be troublesome and inconvenient in practice.
- **Algorithm**. To overcome the computational challenge of kernel learning, we place the function values on a mesh and induce a Kronecker product in the kernel matrices. We use Kronecker product properties and tensor algebra methods to enable highly efficient computation. We then develop an expectation-propagation expectation-maximization (EP-EM) algorithm for efficient alternating posterior inference and function estimation. In the E step, we perform EP to infer quickly the posterior of the selection indicators and operator weights; while in the M step, we maximize the expected model likelihood to solve the current equation (identified by EP) and update the function estimate and kernel parameters.
- **Results**. We examine KBASS in discovering a list of benchmark equations, including three ODE systems, a nonlinear diffusion-reaction equation (diffusion rate 10^{-4}), Burgers' equations with small viscosity (0.1, 0.01, 0.005), and a Kuramoto-Sivashinsky equation. We compared KBASS with state-of-the-art data-driven equation discovery methods. We observed that KBASS can use a much smaller number of data points in the presence of much more noise and still recover the equations, while the competing methods failed in most cases; and that KBASS obtained more accurate weight estimation and provided reasonable uncertainty calibration. Meanwhile, KBASS exhibits significant advantages in computational efficiency.

2 Preliminaries

Kernel Regression. Denote by $\mathcal{H}_\kappa$ the Reproducing Kernel Hilbert Space (RKHS) induced by a Mercer kernel function $\kappa(\cdot,\cdot)$, and by $\|\cdot\|_{\mathcal{H}_\kappa}$ its norm. Given a training dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, we use the regularized regression framework to estimate the target function, $f^*(\mathbf{x}) = \min_{f\in\mathcal{H}_\kappa} \sum_{n=1}^N L(y_n, f(\mathbf{x}_n)) + \sigma^2\|f\|_{\mathcal{H}_\kappa}$, where the solution can be shown to take the form $f(\mathbf{x}) = \sum_{n=1}^N \alpha_n \kappa(\mathbf{x}, \mathbf{x}_n)$ and each $\alpha_n \in \mathbb{R}$. The learning is therefore equivalent to solving an optimization problem for the coefficients $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_N]^\top$, and $\|f\|_{\mathcal{H}_\kappa} = \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha}$, where $\mathbf{K}$ is an $N \times N$ kernel matrix, and each $[\mathbf{K}]_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. Denote by $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)]^\top$ the function values at the training inputs. Since $\mathbf{f} = \mathbf{K}\boldsymbol{\alpha}$, we can instead model the target function as

$$f(\mathbf{x}) = \kappa(\mathbf{x}, \mathbf{X})\mathbf{K}^{-1}\mathbf{f}, \tag{1}$$

where $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$ and $\kappa(\mathbf{x}, \mathbf{X}) = [\kappa(\mathbf{x}, \mathbf{x}_1), \ldots, \kappa(\mathbf{x}, \mathbf{x}_N)]$. The learning amounts to minimizing the following regularized loss, $\operatorname{argmin}_{\mathbf{f}} \sum_{n=1}^N L(y_n, f(\mathbf{x}_n)) + \sigma^2 \mathbf{f}^\top \mathbf{K}^{-1}\mathbf{f}$. When we use the squared loss, the optimum is $\mathbf{f}^* = \mathbf{K}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$, and the function estimate is $f^*(\mathbf{x}) = \kappa(\mathbf{x}, \mathbf{X})(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$. This can also be explained in a Gaussian process (GP) regression framework, where $f^*(\mathbf{x})$ is the posterior mean of GP prediction (Williams and Rasmussen, 2006).

Bayesian spike-and-slab prior is a powerful Bayesian sparse distribution to identify dominant patterns or signals from data (Mitchell and Beauchamp, 1988). Take basis selection as an example for illustration. Suppose we have M basis functions, $\{\phi_j(\mathbf{x})\}_{j=1}^M$, and the target function is $f(\mathbf{x}) = \sum_{j=1}^M w_j \phi_j(\mathbf{x})$. To select dominant or relevant bases, the spike-and-slab prior first samples a binary selection indicator $s_j \in \{0, 1\}$ for each basis j, with which to sample the basis weight w_j,

$$\begin{aligned} p(s_j) &= \text{Bern}(s_j|\rho_0) = \rho_0^{s_j}(1-\rho_0)^{1-s_j}, \\ p(w_j|s_j) &= s_j\mathcal{N}(w_j|0, \sigma_0^2) + (1-s_j)\delta(w_j), \end{aligned} \tag{2}$$

where $\delta(\cdot)$ is a Dirac-delta function. The selection indicator s_j decides the prior over w_j. If s_j is 1, meaning

that basis j is selected, the weight w_j is sampled from a flat Gaussian prior with variance σ_0^2, *i.e.*, the slab component. If s_j is 0, the basis j is pruned, and the weight w_j is sampled from the spike prior that concentrates on zero, *i.e.*, the spike component. Via the selection indicators $\{s_j\}$, the spike-and-slab prior fulfills a selective shrinkage. For the unselected bases, the weights are directly shrunk to zero by the Dirac-delta prior $\delta(\cdot)$. For the selected ones, the weights are sampled from a flat Gaussian, which performs a mild regularization and allows the weights to be adequately estimated from data. Not only is the selective shrinkage observed empirically to be better than the uniform shrinkage in L_1 regularization (Mohamed et al., 2012; Fang et al., 2020), but it is critical for effective selection in terms of risk misclassification (Ishwaran and Rao, 2005).

3 Model

We now present KBASS, our Bayesian model for equation discovery. Without loss of generality, we consider a 2D dynamic system to illustrate the idea. It is straightforward to extend the idea to higher-dimensional problems or reduce the idea to ODE systems. We denote the target (or solution) function by $u(t, x_1, x_2)$. Given a collection of measurement data, $\mathcal{D} = \{(\mathbf{z}_1, y_1), \ldots, (\mathbf{z}_N, y_N)\}$ where each $\mathbf{z}_n = (t_n, x_{n1}, x_{n2})$, we intend to estimate u and its PDE representation. To this end, we introduce a dictionary of candidate operators $\mathcal{O} = \{P_1, \ldots, P_A\}$, which includes basic differentiation operators, such as $\partial_{x_1} u$, $\partial_{x_2} u$, $\partial_{x_1 x_1} u$ and $\partial_{x_1 x_2} u$, their linear/nonlinear combinations, and composition with other functions: $u\partial_{x_1} u$, u, u^2, $\cos(u)$, $\sin(u)$, *etc.* We assume the dictionary is large enough to cover all the possible operators in the ground-truth PDE. We model the PDE with the following form, $u_t - \sum_{j=1}^{K} w_j P_j[u] = 0$, where $\{P_j\}_{j=1}^{K} \subseteq \mathcal{O}$, and each $w_j \in \mathbb{R}$ is the weight of the operator P_j.

To efficiently estimate u, we construct a mesh $\mathcal{M}$ to cover the domain and estimate the function values at the mesh. Note that the mesh is *not* necessary to be evenly spaced. We can randomly sample or design the locations at each input dimension, and then construct the mesh via a Cartesian product. In doing so, we will not only be able to compute efficiently (see Section 4 for details), but also we will be able to flexibly handle different geometries in the domain by varying the dense regions; see Appendix Fig. 8 for an illustration. Denote by $\boldsymbol{\gamma}_j$ the locations at each input dimension j, and by d_j the size of $\boldsymbol{\gamma}_j$. The mesh points are $\mathcal{M} = \boldsymbol{\gamma}_1 \times \boldsymbol{\gamma}_2 \times \boldsymbol{\gamma}_3 = \{(t', x_1', x_2') | t' \in \boldsymbol{\gamma}_1, x_1' \in \boldsymbol{\gamma}_2, x_2' \in \boldsymbol{\gamma}_3\}$.

Denote the function values at $\mathcal{M}$ by $\mathcal{U} = \{u(t', x_1', x_2') | (t', x_1', x_2') \in \mathcal{M}\}$. Hence, $\mathcal{U}$ is a $d_1 \times d_2 \times d_3$ tensor. We use kernel regression to estimate $u(\cdot)$, and, according to (1), the target function can be modeled as a kernel interpolation from $\mathcal{U}$,

$$u(\mathbf{z}) = \kappa(\mathbf{z}, \mathcal{M}) \mathbf{K}_{\mathcal{M},\mathcal{M}}^{-1} \mathrm{vec}(\mathcal{U}), \tag{3}$$

where $\mathbf{z} = (t, x_1, x_2)$ is an arbitrary point in the input domain, $\mathrm{vec}(\cdot)$ is vectorization, $\kappa(\cdot, \cdot)$ is a kernel function, $\mathbf{K}_{\mathcal{M},\mathcal{M}}$ is the kernel matrix on the mesh points $\mathcal{M}$, and $\kappa(\mathbf{z}, \mathcal{M}) = [\kappa(\mathbf{z}, \boldsymbol{\rho}_1), \ldots, \kappa(\mathbf{z}, \boldsymbol{\rho}_d)]$, where $\boldsymbol{\rho}_j$ are all the points in $\mathcal{M}$, $1 \le j \le d$ and $d = d_1 d_2 d_3$. According to (3), we can compute any derivative of u. Since $\mathcal{M}$ and $\mathbf{K}_{\mathcal{M},\mathcal{M}}^{-1} \mathrm{vec}(\mathcal{U})$ are both constant to the input $\mathbf{z}$, we only need to compute the derivatives of the kernel w.r.t the variables in $\mathbf{z}$, *e.g.*,

$$\partial_{x_1 x_2} u(\mathbf{z}) = \partial_{x_1 x_2} \kappa(\mathbf{z}, \mathcal{M}) \cdot \mathbf{K}_{\mathcal{M},\mathcal{M}}^{-1} \mathrm{vec}(\mathcal{U}). \tag{4}$$

Here $\partial_{x_1 x_2} \kappa(\mathbf{z}, \mathcal{M}) \triangleq [\partial_{x_1 x_2} \kappa(\mathbf{z}, \boldsymbol{\rho}_1), \ldots, \partial_{x_1 x_2} \kappa(\mathbf{z}, \boldsymbol{\rho}_d)]$, and all $\{\boldsymbol{\rho}_j\}_{j=1}^{d}$ constitute $\mathcal{M}$. In this way, given the current function estimate (3), we can evaluate each operator in $\mathcal{O}$ on mesh $\mathcal{M}$. We denote them by $\widehat{\mathcal{P}} = \{\mathcal{P}_1, \ldots, \mathcal{P}_A\}$ where each is a $d_1 \times d_2 \times d_3$ tensor.

Based on $\widehat{\mathcal{P}}$, we propose a Bayesian sparse model to identify the operators in the PDE and to estimate the operator weights. Specifically, for each operator j, we use a spike-and-slab prior to sample a selection indicator s_j and operator weight w_j as in (2).

Then, conditioned on $\mathbf{w} = [w_1, \ldots, w_A]^\top$ and $\widehat{\mathcal{P}}$, we sample a virtual dataset $\widehat{\mathbf{y}} = \mathbf{0}$, as

$$p(\widehat{\mathbf{y}} | \boldsymbol{\Phi}, \mathbf{w}) = \mathcal{N}(\widehat{\mathbf{y}} | \mathbf{h} - \boldsymbol{\Phi}\mathbf{w}, \tau \mathbf{I}) = \mathcal{N}(\mathbf{h} | \boldsymbol{\Phi}\mathbf{w}, \tau \mathbf{I}), \tag{5}$$

where $\boldsymbol{\Phi} = [\mathrm{vec}(\mathcal{P}_1), \ldots, \mathrm{vec}(\mathcal{P}_A)]$, $\mathbf{h}$ is $\frac{\partial u}{\partial t}$ evaluated at $\mathcal{M}$ and has been flattened into a vector. The equation likelihood (5) measures how consistent the selected equation (by the spike-and-slab prior) is with the estimated target function at the mesh $\mathcal{M}$. Finally, we use a Gaussian noise model to fit the measurement data,

$$p(\mathbf{y} | \mathcal{U}) = \mathcal{N}(\mathbf{y}_{\mathrm{tr}} | \mathbf{u}_{\mathrm{tr}}, v\mathbf{I}) \tag{6}$$

where $\mathbf{u}_{\mathrm{tr}}$ is obtained via the kernel interpolation (3) at the training inputs and $\mathbf{y}_{\mathrm{tr}} = [y_1, \ldots, y_N]^\top$ are the training outputs.

Note that a particular advantage of using the spike-and-slab prior (2) is that we can estimate the posterior of each selection indicator $p(s_j | \mathcal{D}, \widehat{\mathbf{y}})$, and use it to decide if an operator should be selected, *e.g.*, checking if $p(s_j = 1 | \mathcal{D}, \widehat{\mathbf{y}}) > 0.5$. We never need to set a threshold over the weight values, *e.g.*, $|w_j| > 10^{-3}$, as popular methods (like SINDy and PINN-SR) do. Tuning a weight threshold can be troublesome and inconvenient in practice. A bigger threshold can miss important operators with small weights, *e.g.*, in Burger's equation with a small viscosity; a smaller one, however, can easily select false operators. Moreover, the threshold is often not general. When switching to a different problem, one has to carefully tune it from scratch.

4 Algorithm

Given the measurement data $\mathcal{D}$ and the dictionary $\mathcal{O}$, the learning of our model amounts to estimating $\mathcal{U}$, the kernel parameters, and the posterior distribution of $\mathbf{s}$ and $\mathbf{w}$. This is challenging. First, when the number of mesh points is large, the kernel interpolation that requires the inverse of $\mathbf{K}_{\mathcal{M},\mathcal{M}}$ can be extremely costly or even infeasible; see (3) and (4). Second, because the spike-and-slab prior (2) mixes binary and continuous random variables, the posterior distribution is analytically intractable. To address these challenges, we induce a Kronecker product in kernel computation, and use tensor algebra to avoid operating on full matrices. We then develop an expectation-propagation expectation-maximization (EP-EM) algorithm. In each iteration, our algorithm performs two steps. In the E step, we fix $\mathcal{U}$ and the kernel parameters, and use EP (Minka, 2001a) to estimate the posterior of $\mathbf{s}$ and $\mathbf{w}$. In the M step, we maximize the expected model likelihood to update $\mathcal{U}$ and the kernel parameters. The alternating of the E and M steps keeps mutually improving equation discovery and function estimation until convergence. See Algorithm 1 for a summary.

Specifically, thanks to the usage of a mesh for placing the function values to be estimated, $\mathcal{U}$ is a $d_1 \times d_2 \times d_3$ tensor, and if we use a product-kernel, $\kappa(\mathbf{z}, \mathbf{z}') = \kappa(t, t')\kappa(x_1, x_1)\kappa(x_2, x_2)$, where $\mathbf{z} = (t, x_1, x_2)$ and $\mathbf{z}' = (t', x_1', x_2')$, we can immediately induce a Kronekcer product structure in the kernel matrix,

$$\mathbf{K}_{\mathcal{M},\mathcal{M}} = \mathbf{K}_1 \otimes \mathbf{K}_2 \otimes \mathbf{K}_3, \tag{7}$$

where $\mathbf{K}_1 = \kappa(\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_1)$, $\mathbf{K}_2 = \kappa(\boldsymbol{\gamma}_2, \boldsymbol{\gamma}_2)$, and $\mathbf{K}_3 = \kappa(\boldsymbol{\gamma}_3, \boldsymbol{\gamma}_3)$ are the kernel matrices of the inputs at each dimension of $\mathcal{M}$. Note that the popular square exponential (SE) kernel is a product kernel. One can also construct a product kernel from any other kernels.

We can then leverage the nice properties of Kronecker product (Minka, 2000) and tensor algebra (Kolda, 2006) to avoid computing the full kernel matrix and tremendously improve the efficiency. First, to compute (3), we can derive that $u(\mathbf{z}) = \kappa(t, \boldsymbol{\gamma}_1) \otimes \kappa(x_1, \boldsymbol{\gamma}_2) \otimes \kappa(x_2, \boldsymbol{\gamma}_3) (\mathbf{K}_1 \otimes \mathbf{K}_2 \otimes \mathbf{K}_3)^{-1} \operatorname{vec}(\mathcal{U}) = \left(\kappa(t, \boldsymbol{\gamma}_1)\mathbf{K}_1^{-1} \otimes \kappa(x_1, \boldsymbol{\gamma}_2)\mathbf{K}_2^{-1} \otimes \kappa(x_2, \boldsymbol{\gamma}_3)\mathbf{K}_3^{-1}\right) \operatorname{vec}(\mathcal{U}) = \mathcal{U} \times_1 \kappa(t, \boldsymbol{\gamma}_1)\mathbf{K}_1^{-1} \times_2 \kappa(x_1, \boldsymbol{\gamma}_2)\mathbf{K}_2^{-1} \times_3 \kappa(x_2, \boldsymbol{\gamma}_3)\mathbf{K}_3^{-1}$, where $\times_k$ is the tensor-matrix product at mode k (Kolda, 2006). Hence, we only need to compute the inverse of each $\mathbf{K}_j$ $(1 \le j \le 3)$, which takes time complexity $O(d_1^3 + d_2^3 + d_3^3)$, and the tensor-matrix product takes $O(d)$ time complexity $(d = d_1 d_2 d_3)$ — linear in the size of $\mathcal{U}$. Hence, it is much more efficient than the naive computation — $O(d^3)$ time complexity.

Next, to compute a derivative of u at $\mathcal{M}$, we find that, because *the product kernel is decomposed over individual input variables*, the corresponding kernel differentiation still maintains the product structure. For example, to obtain $\partial_{x_1 x_2} u$, we have the corresponding kernel derivative (see (4)) as $\partial_{x_1 x_2}\kappa(\mathbf{z}, \boldsymbol{\rho}_j) = \partial_{x_1 x_2}\left[\kappa(t, \rho_{j1})\kappa(x_1, \rho_{j2})\kappa(x_2, \rho_{j3})\right] = \kappa(t, \rho_{j1}) \cdot \partial_{x_1}\kappa(x_1, \rho_{j2}) \cdot \partial_{x_2}\kappa(x_2, \rho_{j3})$. Accordingly, to compute $\partial_{x_1 x_2}\mathcal{U} \triangleq \{\partial_{x_1 x_2} u(\mathbf{z}) | \mathbf{z} \in \mathcal{M}\}$, we have $\operatorname{vec}(\partial_{x_1 x_2}\mathcal{U}) = (\mathbf{K}_1 \otimes D_1[\mathbf{K}_2] \otimes D_1[\mathbf{K}_3]) (\mathbf{K}_1 \otimes \mathbf{K}_2 \otimes \mathbf{K}_3)^{-1} \operatorname{vec}(\mathcal{U}) = \left(\mathbf{I} \otimes D_1[\mathbf{K}_2]\mathbf{K}_2^{-1} \otimes D_1[\mathbf{K}_3]\mathbf{K}_3^{-1}\right) \operatorname{vec}(\mathcal{U})$, and hence $\partial_{x_1 x_2}\mathcal{U} = \mathcal{U} \times_2 D_1[\mathbf{K}_2]\mathbf{K}_2^{-1} \times_3 D_1[\mathbf{K}_3]\mathbf{K}_3^{-1}$, where $D_1[\cdot]$ means taking the partial derivative w.r.t the first input variable for each kernel function inside, $D_1[\mathbf{K}_2] = [\partial_\gamma \kappa(\gamma, \gamma')]_{\gamma,\gamma' \in \boldsymbol{\gamma}_2}$, and $D_1[\mathbf{K}_3] = [\partial_\gamma \kappa(\gamma, \gamma')]_{\gamma,\gamma' \in \boldsymbol{\gamma}_3}$. The multilinear operation has the time complexity $O((d_2 + d_3)d)$. Hence, the overall time complexity is $O(d_1^3 + d_2^3 + d_3^3 + (d_2 + d_3)d)$ which again is much more efficient than the naive computation with $O(d^3))$ complexity. Accordingly, we can compute the values of all the required derivatives on $\mathcal{M}$ highly efficiently.

E Step. With this efficient computational method, we perform EP-EM steps for model estimation. In the E step, we estimate $p(\mathbf{s}, \mathbf{w} | \mathcal{D}, \widehat{\mathbf{y}})$ given the current estimate of $\mathcal{U}$. The joint distribution is

$$p(\mathbf{w}, \mathbf{s}, \mathcal{D}, \widehat{\mathbf{y}}) \propto \mathcal{N}(\mathbf{h} | \boldsymbol{\Phi}\mathbf{w}, \tau\mathbf{I}) \cdot \prod_{j=1}^{A} \operatorname{Bern}(s_j | \rho_0) \left(\mathbf{s}_j \mathcal{N}(w_j | 0, \sigma_0^2) + (1 - s_m)\delta(w_j)\right). \tag{8}$$

We can see that the problem is reduced to the inference for Bayesian linear regression with the spike-and-slab prior. We therefore use a similar approach to (Fang et al., 2020) to develop an efficient EP method. EP approximates each non-exponential-family factor in the joint distribution with an exponential-family term. Then from the approximate joint distribution, we can obtain a closed-form posterior, since the exponential family is closed under multiplication. EP iteratively updates each factor approximation until convergence. We leave the details in Appendix Section A.1. After our EP inference, we obtain a posterior approximation,

$$p(\mathbf{s}, \mathbf{w} | \mathcal{D}, \widehat{\mathbf{y}}) \approx q(\mathbf{s}, \mathbf{w}) = \prod_{j=1}^{A} \operatorname{Bern}(s_j | \sigma(\widehat{\rho}_j)) \cdot \mathcal{N}(\mathbf{w} | \boldsymbol{\beta}, \boldsymbol{\Sigma}), \tag{9}$$

where $\sigma(\cdot)$ is the sigmoid function. We then prune the operators according to $q(\mathbf{s})$ to decide the currently discovered equation.

M Step. In the M step, we maximize the expected log model likelihood under $q(\mathbf{w})$ over the remaining operators to update $\mathcal{U}$ and the kernel parameters,

$$\mathcal{L} = -\frac{1}{2}\operatorname{vec}(\mathcal{U})^\top \mathbf{K}_{\mathcal{M}\mathcal{M}}^{-1} \operatorname{vec}(\mathcal{U}) - \frac{1}{2v}\|\mathbf{y} - \mathbf{f}\|^2 - \frac{1}{2\tau}\mathbb{E}_q\left[\left\|\mathbf{h} - \widehat{\boldsymbol{\Phi}}\widehat{\mathbf{w}}\right\|^2\right] + \text{const}, \tag{10}$$

Algorithm 1 KBASS ($\mathcal{D}$, $\mathcal{O}$)

1: Learn a kernel regression model from $\mathcal{D}$ to obtain an initial estimate of $\mathcal{U}$ and the kernel parameters $\boldsymbol{\theta}$.
2: **repeat**
3: E-step: Fix $\mathcal{U}$ and $\boldsymbol{\theta}$, run EP to estimate $q(\mathbf{s}, \mathbf{w})$ in (9). Prune the operators with $q(s_j = 1) < 0.5$.
4: M-step: Fix $q(\mathbf{s}, \mathbf{w})$, run ADAM for 100 steps to maximize (10) to update $\mathcal{U}$ and $\boldsymbol{\theta}$.
5: **until** the maximum iteration number is reached or the relative change of $\mathcal{U}$ is less than a tolerance level.
6: **return** $q(\mathbf{s}, \mathbf{w})$, $\mathcal{U}$ and the kernel parameters $\boldsymbol{\theta}$.

where $\widehat{\boldsymbol{\Phi}}$ and $\widehat{\mathbf{w}}$ are the evaluation and weights of the remaining operators, respectively. Note that the first term is the RKHS norm of the target function estimate (3), which regularizes the learning of $u(\cdot)$. With the Kronecker product and tensor algebra, the computation of the RKHS norm is highly efficient. We can use any gradient based optimization to maximize $\mathcal{L}$.

Algorithm Complexity. The time complexity of our model estimation algorithm is $O(\sum_{j=1}^{3} d_j^3 + (N + \sum_j d_j)d)$, where $d = d_1 d_2 d_3$. The space complexity is $O(\sum_{j=1}^{3} d_j^2 + d + A^2)$, which is to store the kernel matrices at each dimension, $\mathcal{U}$ and $q(\mathbf{s}, \mathbf{w})$.

5 Related Work

The recent break-through SINDy (Brunton et al., 2016) uses sparse linear regression to select PDE operators from a pre-specified dictionary. The sparsity is fulfilled by a sequential threshold ridge regression (STRidge) method, which repeatedly conducts least-mean-square estimation and weight truncation. While SINDy was originally developed to discover ODEs, by augmenting the dictionary with partial derivatives over spatial variables, SINDy can be extended for discovering PDEs for spatial-temporal systems (Rudy et al., 2017; Schaeffer, 2017). The SINDy family of approaches use numerical differentiation to evaluate candidate operators, and it can suffer from scarce and noisy data. To alleviate this issue, recent works (Berg and Nyström, 2019; Both et al., 2021; Chen et al., 2021b) use deep NNs to approximate the solution, applying automatic differentiation to evaluate and select the operators. These methods can therefore be viewed as instances/extension of the PINNs (Raissi et al., 2019; Krishnapriyan et al., 2021). In (Berg and Nyström, 2019), a deep NN is first used to fit the data, and then a sparse linear regression with L_1 regularization is applied to select the operators. In (Both et al., 2021), the NN and sparse linear regression are jointly trained. In (Chen et al., 2021b), a joint training strategy is also used, but it performs an alternating optimization of the NN loss and sparse regression, and it uses STRidge for weight truncation. The most recent work (Sun et al., 2022) develops a Bayesian spline learning (BSL) method, which uses splines to approximate the solution and a student-t prior, *i.e.*, relevance vector machines (Tipping, 2001), for sparse linear regression. It uses a similar alternating optimization strategy to (Chen et al., 2021b). To quantify the uncertainty, BSL then applies Stochastic Weight Averaging-Gaussian (SWAG) (Maddox et al., 2019) for posterior approximation of the weights.

Our work uses kernel methods to estimate the solution function for PDE discovery. GP/kernel methods have been applied for solving differential equations for a long time. For example, Graepel (2003); Raissi et al. (2017) used GPs to solve linear PDEs with the noisy source term measurements. The recent work (Chen et al., 2021a) developed a general kernel method to solve linear/nonlinear PDEs. However, all these methods assume the equations are given, and thus they cannot discover the equations from data. GPs have also been used to model the equation components or incorporate the equations for better training, such as (Heinonen et al., 2018; Alvarez et al., 2009; Barber and Wang, 2014; Macdonald et al., 2015; Wenk et al., 2019, 2020). The GP community has realized the computational advantage of the Kronecker product (Saat**c**ci, 2012), and there has been work in leveraging the Kronecker product properties to improve the training speed and scalability, such as high-dimension output regression (Zhe et al., 2019) and sparse approximation based on inducing points (Wilson and Nickisch, 2015). However, in typical machine learning applications, the data are not observed at a grid and the Kronecker product has a limited usage. By contrast, for PDE solving, it is natural to estimate the solution values on a mesh (which is consistent with the practice of traditional numerical methods). This opens the possibility of using Kronecker products for efficient computation. To our knowledge, our work is first to realize this point and use the Kronecker product structure to enable efficient PDE discovery and solution estimation.

6 Empirical Results

For evaluation, we considered several benchmark ODEs/PDEs in the literature of data-driven equation discovery (Chen et al., 2021b; Sun et al., 2022). We examined wether KBASS can recover these equations from sparse, noisy measurements. We used the SE kernel and implemented KBASS with Jax (Frostig et al., 2018). We compared with state-of-the-art data-driven equation discovery methods, including SINDy (Brunton et al., 2016), PINN-SR (Chen et al., 2021b) and BSL (Sun et al., 2022). For the competing methods, we used the original implementation released by the authors. For a fair comparison, we carefully tuned every method to achieve the best performance (to our

best effort). We leave the details about the settings and tuning of all the methods in Appendix Section A.2.

Van del Pol (VDP) oscillator. We first tested the discovery of a nonlinear ODE system, $x_t = y, y_t = \mu(1-x^2)y - x$ where $\mu = 2.5$ and $x(0) = y(0) = 1$. We used two test settings. The first setting employed only 10 evenly-spaced training examples in $[0, 8]$ while the second setting used 25 such examples. For each setting, we considered three noise levels: 0%, 1% and 20%. The dictionary includes the combination of the polynomials of each state variable up to 4th order, which gives 24 candidates in total. To evaluate the discovery performance, we followed (Sun et al., 2022) to examine the normalized root-mean-square error (RMSE) of the weight estimation (including the relevant and irrelevant operators), and the precision and the recall of the selected operators.

Lorenz 96. We next tested on Lorenz 96, a well-known chaotic ODE system, $\frac{\mathrm{d}x_i}{\mathrm{d}t} = (x_{i+1} - x_{i-2})x_{i-1} - x_i + F$, where $t \in [0, 5]$, $1 \le i \le n$, $x_0 = x_n$, $x_{-1} = x_{n-1}$ and $x_{n+1} = x_1$. We set $n = 6$ to have six state variables and set forcing term $F = 8$. The initial condition is 2 for every state. The dictionary includes the polynomials of the state variables up to 3rd order, leading to 84 candidates in total. We performed two tests: one used 12 evenly-spaced examples, and the other used 50 such examples in the domain.

Burger's equation. Third, we tested on 1D Burger's equations,

$$u_t + uu_x - \nu u_{xx} = 0, \quad (x, t) \in [0, 10] \times [0, 8], \quad (11)$$

where the initial condition is $u(x, 0) = \exp(-(x-4)^2)$. We tested with three viscosity values, $\nu = 0.1$, $\nu = 0.01$, and $\nu = 0.005$. The dictionary consists of the polynomials and derivatives of u up to 4th order and their combinations, which in total includes 24 candidate operators. For $\nu = 0.1$, we first tested with 100 training examples on an evenly-spaced 10×10 grid in the domain. We then tested with a densor dataset, including 400 examples at a 20×20 evenly-spaced grid. Next, for $\nu = 0.01$ and $\nu = 0.005$, we used training examples at a 50×50 evenlly-spaced grid.

Kuramoto-Sivashinsky (KS) equation. Fourth, we tested the discovery of a KS equation,

$$u_t + uu_x + u_{xx} + u_{xxxx} = 0, \quad (x, t) \in [0, 32\pi] \times [0, 150].$$

We used a periodic boundary condition $u(x, 0) = \cos(x/16)$ for data generation. The dictionary includes polynomials and derivatives of u up to 4th order and their combinations, which in total gives 24 candidate operators. We used training examples at an evenly spaced 40×40 grid in the domain.

Allen-cahn equation. Fifth, we tested on an Allen-cahn equation with a very small diffusion rate,

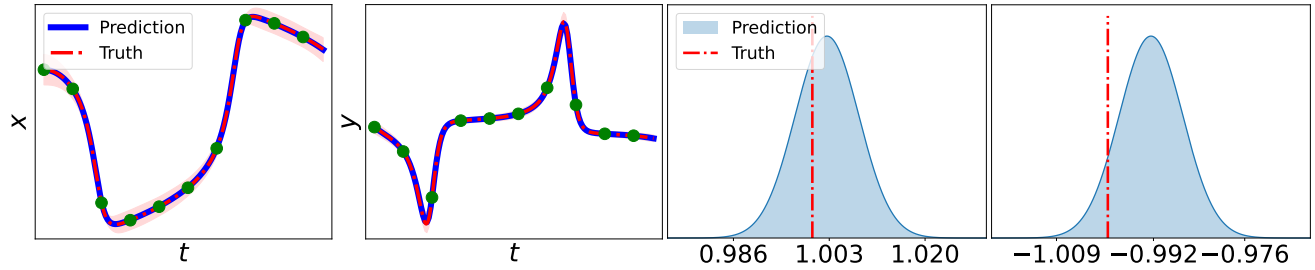
$$u_t = 10^{-4}u_{xx} + 5u - 5u^3, \quad (x, t) \in [-1, 1] \times [0, 1],$$

where $u(x, 0) = x^2 \cos(\pi x)$, $u(-1, t) = u(1, t)$ and $u_x(-1, t) = u_x(1, t)$. The dictionary consists of the polynomials and derivatives of u up to 3rd order and their combinations, which in total includes 16 candidate operators. We used training examples at a 26×101 grid in the spatial-temporal domain.

Real-world application. Finally, we ran our method on the data of a real-world predator-prey system. The dataset and experimental details are given in Appendix Section A.3.

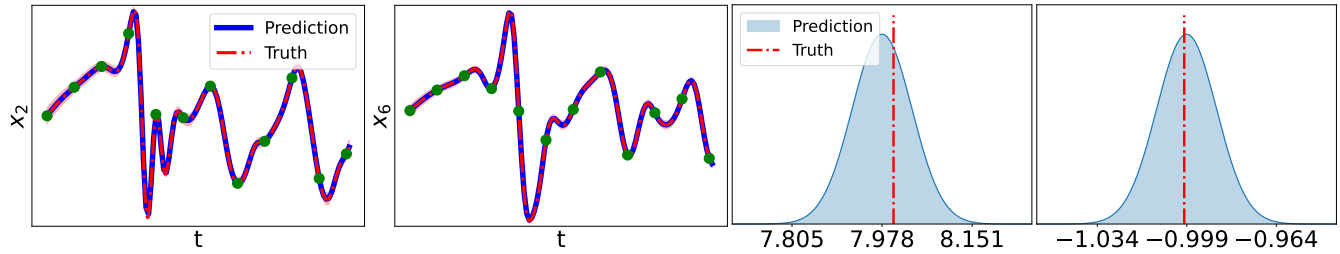
Results and discussion. We report the discovery performance of each method in Table 1, 2, 3, and 4 and Appendix Table 6. We can see that in all the settings, KBASS successfully recovered the equations, and the error of the operator weight estimate is small. As a comparison, SINDy failed to find the equations for every case. This might be due to that the sampled measurements (*i.e.*, training examples) are too sparse. We found that SINDy started to successfully recover VDP, Lorenz 96, Burger's ($\nu = 0.1$), and KS equations when we increased to 50, 250, 1.6K and 384.5K examples respectively, which takes 5x and 21x, 16x and 240x of the sample size needed by KBASS. Nonetheless, the accuracy of the operator weight estimation is still much worse than KBASS; see the results in Appendix Table 7. PINN-SR failed to discover all the ODEs. This is consistent with the observation in (Sun et al., 2022). Though PINN-SR can exactly recover the Burger's equation ($\nu = 0.1$) with 10×10 and 20×20 examples, when adding a tiny amount of noise (1%), PINN-SR immediately failed; see Table 3a. Similarly, BSL can exactly recover the Burger's equation ($\nu = 0.1$) using 20×20 examples, with zero or 1% noise. However, it failed with 20% noise. Hence, it illustrates that KBASS is much more robust to data noise. In addition, both BSL and PINN-SR failed to find the Burger's equation with $\nu = 0.01$ and $\nu = 0.005$, which are more challenging; see Table 3b. So too did they for the KS and Allen-cahn equations; see Table 4. Note that although in (Chen et al., 2021b), PINN-SR successfully recovers a KS equation, it uses 320×101 examples — 20x of the examples used by KBASS. We were not able to use PINN-SR to recover the KS equation tested in our experiment with the same amount of data. This might relate to our usage of a much larger domain, $[0, 32\pi] \times [0, 150]$ versus $[0, 32\pi] \times [0, 100]$.

Next, we showcased our solution prediction and posterior estimate of the operator weights in Fig. 1, 2, 3, 4, and Appendix Fig. 5, 6, 7. The shaded region in the solution prediction indicates the predictive standard



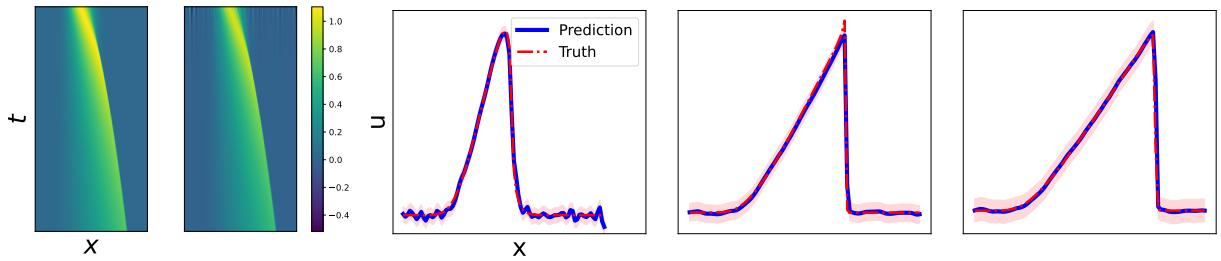
(a) (b) (c) (d)

Figure 1: Solution and weight posterior estimation for the VDP equation with 10 training examples (marked as green); (c) and (d) show the weight posterior for x and $-x$, respectively. Their posterior selection probabilities were both estimated as 1.0.



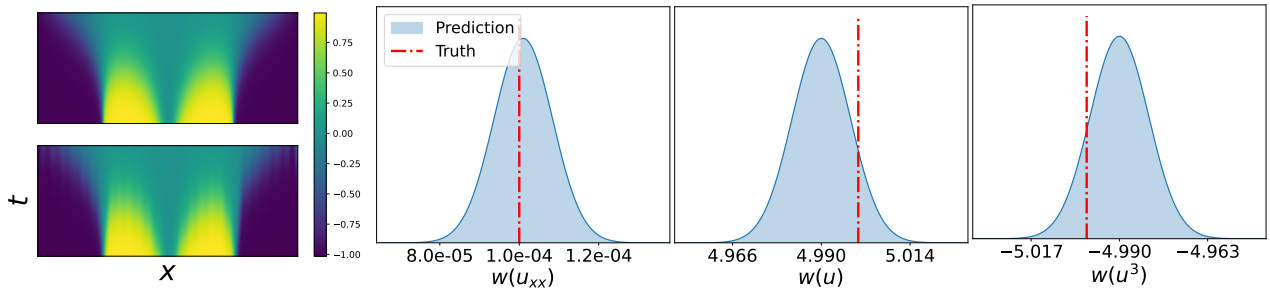
(a) (b) (c) (d)

Figure 2: Solution and weight posterior estimation for Lorenz 96 using 12 training examples; (c) and (d) show the weight posterior for the force term F and x_5. The posterior selection probabilities were estimated as 1.0.



(a) Left: truth; right: prediction (b) $t = 1.1$ (c) $t = 4.0$ (d) $t = 7.0$

Figure 3: Solution estimate for Burger's equation with $\nu = 0.005$ with 20% noise on the training data.



(a) Top: truth; bottom: prediction (b) $q(s(u_{xx}) = 1) = 1.0$ (c) $q(s(u) = 1) = 1.0$ (d) $q(s(u^3) = 1) = 1.0$

Figure 4: Solution and weight posterior estimation on Allen-cahn equation; $w(\cdot)$ and $s(\cdot)$ denote the weight and selection indicator of the operator.

deviation (scaled by a factor to be more clear). We used the Laplace's approximation (see details in Appendix Section A.4) to estimate the posterior of the solution prediction.

Method	10 examples			25 examples		
	(noise) 0%	1%	20%	0%	1%	20%
SINDy	F/0.5/1	F/0.5/1	F/0.5/1	F/0.75/0.6	F/0.75/0.6	F/0.75/0.75
PINN-SR	F/1/0.67	F/0.5/0.13	F/0.75/0.23	F/0.25/0.33	F/0.5/0.33	F/0/0
BSL	F/0.5/1	F/0.5/1	F/0.5/0.12	0/1/1	0.0029/1/1	F/0.75/1
KBASS	**0.0086/1/1**	**0.032/1/1**	**0.0995/1/1**	**0/1/1**	**0.0029/1/1**	**0.042/1/1**

Table 1: Performance of discovering VDP oscillators ($t \in [0, 8]$), with 0%, 1% and 20% noises in the training data. Each entry is normalized RMSE/Recall/Precision and "F" means Failed.

Method	12 examples			50 examples		
	(noise) 0%	1%	10%	0%	1%	10%
SINDy	F/0.083/0.29	F/0.125/0.38	F/0.083/0.33	F/0.17/0.8	F/0.17/0.8	F/0.083/0.5
PINN-SR	F/0/0	F/0/0	F/0/0	F/0/0	F/0/0	F/0/0
BSL	F/0.17/0.5	F/0.083/0.33	F/0.083/0.33	0.0011/1/1	0.0049/1/1	F/0.67/0.25
KBASS	**0.0068/1/1**	**0.0125/1/1**	**0.134/1/1**	**0/1/1**	**0.0035/1/1**	**0.054/1/1**

Table 2: Performance of discovering Lorenz 96 systems with 6 state variables and $t \in [0, 5]$.

Method	10×10 examples			20×20 examples		
	(noise) 0%	1%	20%	0%	1%	20%
SINDy	F/0.5/0.2	F/0.5/0.2	F/0.5/0.67	F/1/0.4	F/1/0.4	F/0/0
PINN-SR	0/1/1	F/1/0.4	F/1/0.4	0/1/1	F/1/0.4	F/1/0.13
BSL	F/0.5/1	F/0/0	F/0/0	0/1/1	0/1/1	F/0/0
KBASS	**0/1/1**	**0.0032/1/1**	**0.038/1/1**	**0/1/1**	**0/1/1**	**0.038/1/1**

(a) $\nu = 0.1$

Method	$\nu = 0.01$			$\nu = 0.005$		
	0%	10%	20%	0%	10%	20%
SINDy	F/0.5/0.2	F/0.5/0.2	F/0.5/0.2	F/0.5/0.2	F/1/0.4	F/1/0.4
PINN-SR	F/1/0.14	F/0.5/0.2	F/0.5/0.5	F/0.5/0.2	F/0.5/0.2	F/0/0
BSL	F/0.5/0.33	F/0/0	F/0/0	F/0.5/0.25	F/0/0	F/0/0
KBASS	**0.003/1/1**	**0.007/1/1**	**0.008/1/1**	**0.0013/1/1**	**0.0026/1/1**	**0.0047/1/1**

(b) $\nu = 0.01$ and $\nu = 0.005$ on 50×50 examples.

Table 3: Performance of discovering a 1D Burger's equation, with $(x, t) \in [0, 10] \times [0, 8]$.

Method	KS equation			Allen-cahn equation		
	(noise) 0%	10%	20%	0%	5%	10%
SINDy	F/0.33/0.2	F/0.33/0.2	F/0.33/0.2	F/0.67/1	F/0.67/1	F/0.67/1
PINN-SR	F/0/0	F/0/0	F/0.33/0.33	F/0.67/0.33	F/0.67/0.33	F/0.67/0.33
BSL	F/0/0	F/0/0	F/0/0	F/0.67/0.67	F/0.67/1	F/0.67/0.5
KBASS	**0.013/1/1**	**0.023/1/1**	**0.025/1/1**	**0.0023/1/1**	**0.0071/1/1**	**0.045/1/1**

Table 4: Performance of discovering a 1D KS equation with $(x, t) \in [0, 32\pi] \times [0, 150]$, and 1D Allen-cahn equation with $(x, t) \in [-1, 1] \times [0, 1]$.

We can see that the solution prediction by KBASS is quite accurate compared to the ground-truth, even when the training data is very sparse and noisy, *e.g.*, the prediction for Burger's equation with $\nu = 0.005$; see Fig 3. This might be (partly) due to that the accurate discovery of the equation in our E step can simultaneously guide the solution learning (M step). From the posterior estimate for the operator weights, we can see that the posterior mean is close to the ground-truth, while the Gaussian distribution allows us to further quantify the uncertainty. It is particularly interesting to see that in Fig. 4b, KBASS successfully identifies the operator u_{xx} that has a very small weight 10^{-4} and provides a Gaussian posterior approximation. Note that the posterior selection probability $q(s(u_{xx}) = 1) = 1.0$, reflecting that KBASS has a very high confidence to select u_{xx}, even though the scale of the weight is very small. Hence, KBASS does not need to use a weight threshold for operator selection.

Computational efficiency. We examined the running time of each method. The results are given in Appendix Section A.5. As shown in Appendix Table 8, KBASS takes much less time than BSL. For example, on Burger's and KS , PINN-SR takes 9x and 20x running time but still failed to recover the KS equation. Together this shows our method also has a significant advantage in computational efficiency.

7 Conclusion

We have presented KBASS, an efficient kernel learning method with Bayesian spike-and-slab priors for data-driven equation discovery. Our method has higher sample efficiency and is more resistant to data noise than existing methods. KBASS has shown significant advantages on a series of PDE/ODE discovery tasks. Currently, KBASS is limited to a pre-specified operator dictionary. In the future, we plan to develop methods that can dynamically expand the dictionary to further improve the accuracy and robustness.

Acknowledgement

This work has been supported by MURI AFOSR grant FA9550-20-1-0358 and NSF CAREER Award IIS-2046295.

References

Alvarez, M., Luengo, D., and Lawrence, N. D. (2009). Latent force models. In Artificial Intelligence and Statistics, pages 9–16.

Barber, D. and Wang, Y. (2014). Gaussian processes for Bayesian estimation in ordinary differential equations. In International Conference on Machine Learning, pages 1485–1493.

Berg, J. and Nyström, K. (2019). Data-driven discovery of pdes in complex datasets. Journal of Computational Physics, 384:239–252.

Both, G.-J., Choudhury, S., Sens, P., and Kusters, R. (2021). DeepMoD: Deep learning for model discovery in noisy data. Journal of Computational Physics, 428:109985.

Brunton, S. L., Proctor, J. L., and Kutz, J. N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proceedings of the national academy of sciences, 113(15):3932–3937.

Chen, Y., Hosseini, B., Owhadi, H., and Stuart, A. M. (2021a). Solving and learning nonlinear PDEs with Gaussian processes. arXiv preprint arXiv:2103.12959.

Chen, Z., Liu, Y., and Sun, H. (2021b). Physics-informed learning of governing equations from scarce data. Nature communications, 12(1):1–13.

Fang, S., Zhe, S., Lee, K.-c., Zhang, K., and Neville, J. (2020). Online Bayesian sparse learning with spike and slab priors. In 2020 IEEE International Conference on Data Mining (ICDM), pages 142–151. IEEE.

Frostig, R., Johnson, M. J., and Leary, C. (2018). Compiling machine learning programs via high-level tracing. Systems for Machine Learning, 4(9).

Graepel, T. (2003). Solving noisy linear operator equations by Gaussian processes: Application to ordinary and partial differential equations. In ICML, pages 234–241.

Heinonen, M., Yildiz, C., Mannerström, H., Intosalmi, J., and Lähdesmäki, H. (2018). Learning unknown ODE models with Gaussian processes. In International Conference on Machine Learning, pages 1959–1968.

Ishwaran, H. and Rao, J. S. (2005). Spike and slab variable selection: Frequentist and Bayesian strategies. The Annals of statistics, 33(2):730–773.

Kolda, T. G. (2006). Multilinear operators for higher-order decompositions. Technical report, Sandia National Laboratories (SNL).

Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., and Mahoney, M. W. (2021). Characterizing possible failure modes in physics-informed neural networks. Advances in Neural Information Processing Systems, 34:26548–26560.

Lagergren, J. H., Nardini, J. T., Michael Lavigne, G., Rutter, E. M., and Flores, K. B. (2020). Learning partial differential equations for biological transport models from noisy spatio-temporal data. Proceedings of the Royal Society A, 476(2234):20190800.

Macdonald, B., Higham, C., and Husmeier, D. (2015). Controversy in mechanistic modelling with Gaussian processes. Proceedings of Machine Learning Research, 37:1539–1547.

MacKay, D. J. (2003). Information theory, inference and learning algorithms. Cambridge university press.

Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. (2019). A simple baseline for Bayesian uncertainty in deep learning. Advances in neural information processing systems, 32.

Minka, T. P. (2000). Old and new matrix algebra useful for statistics.

Minka, T. P. (2001a). Expectation propagation for approximate Bayesian inference. In Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence, pages 362–369.

Minka, T. P. (2001b). A family of algorithms for approximate Bayesian inference. PhD thesis, Massachusetts Institute of Technology.

Mitchell, T. J. and Beauchamp, J. J. (1988). Bayesian variable selection in linear regression. Journal of the american statistical association, 83(404):1023–1032.

Mohamed, S., Heller, K. A., and Ghahramani, Z. (2012). Bayesian and l1 approaches for sparse unsupervised learning. In Proceedings of the 29th International Coference on International Conference on Machine Learning, pages 683–690.

Mojgani, R., Balajewicz, M., and Hassanzadeh, P. (2022). Lagrangian PINNs: A causality-conforming solution to failure modes of physics-informed neural networks. arXiv preprint arXiv:2205.02902.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2017). Machine learning of linear differential equations using Gaussian processes. Journal of Computational Physics, 348:683–693.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378:686–707.

Ritter, H., Botev, A., and Barber, D. (2018). A scalable laplace approximation for neural networks. In 6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings, volume 6. International Conference on Representation Learning.

Rudy, S. H., Brunton, S. L., Proctor, J. L., and Kutz, J. N. (2017). Data-driven discovery of partial differential equations. Science advances, 3(4):e1602614.

Saatcci, Y. (2012). Scalable inference for structured Gaussian process models. PhD thesis, Citeseer.

Schaeffer, H. (2017). Learning partial differential equations via data discovery and sparse optimization. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 473(2197):20160446.

Sun, L., Huang, D., Sun, H., and Wang, J.-X. (2022). Bayesian spline learning for equation discovery of nonlinear dynamics with quantified uncertainty. Advances in Neural Information Processing Systems, 35:6927–6940.

Tipping, M. E. (2001). Sparse Bayesian learning and the relevance vector machine. Journal of machine learning research, 1(Jun):211–244.

Wainwright, M. J., Jordan, M. I., et al. (2008). Graphical models, exponential families, and variational inference. Foundations and Trends® in Machine Learning, 1(1–2):1–305.

Walker, A. M. (1969). On the asymptotic behaviour of posterior distributions. Journal of the Royal Statistical Society: Series B (Methodological), 31(1):80–88.

Wenk, P., Abbati, G., Osborne, M. A., Schölkopf, B., Krause, A., and Bauer, S. (2020). ODIN: ODE-informed regression for parameter and state inference in time-continuous dynamical systems. In AAAI, pages 6364–6371.

Wenk, P., Gotovos, A., Bauer, S., Gorbach, N. S., Krause, A., and Buhmann, J. M. (2019). Fast Gaussian process based gradient matching for parameter identification in systems of nonlinear ODEs. In The 22nd International Conference on Artificial Intelligence and Statistics, pages 1351–1360. PMLR.

Williams, C. K. and Rasmussen, C. E. (2006). Gaussian processes for machine learning, volume 2. MIT press Cambridge, MA.

Wilson, A. and Nickisch, H. (2015). Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In International conference on machine learning, pages 1775–1784. PMLR.

Zhang, J. and Ma, W. (2020). Data-driven discovery of governing equations for fluid dynamics based on molecular simulation. Journal of Fluid Mechanics, 892.

Zhe, S., Xing, W., and Kirby, R. M. (2019). Scalable high-order Gaussian process regression. In The 22nd International Conference on Artificial Intelligence and Statistics, pages 2611–2620. PMLR.

A SUPPLEMENTARY MATERIAL

A.1 EP Spike-and-Slab Inference

A.1.1 EP Tutorial

We first give a brief introduction to the expectation propagation (EP) framework (Minka, 2001a). Consider a general probabilistic model with latent parameters $\boldsymbol{\theta}$. Given the observed data $\mathcal{D} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$, the joint probability distribution is

$$p(\boldsymbol{\theta}, \mathcal{D}) = p(\boldsymbol{\theta}) \prod_{n=1}^{N} p(\mathbf{y}_n|\boldsymbol{\theta}). \tag{12}$$

Our goal is to compute the posterior $p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\boldsymbol{\theta},\mathcal{D})}{p(\mathcal{D})}$. However, it is usually infeasible to compute the exact marginal distribution $p(\mathcal{D})$, because the complexity of the likelihood and/or prior makes the integration or marginalization intractable. This further makes the exact posterior distribution infeasible to compute. EP therefore seeks to approximate each term in the joint distribution by an exponential-family term,

$$p(y_n|\boldsymbol{\theta}) \approx c_n f_n(\boldsymbol{\theta}), \quad p(\boldsymbol{\theta}) \approx c_0 f_0(\boldsymbol{\theta}), \tag{13}$$

where c_n and c_0 are constants to ensure the normalization consistency (they will get canceled in the inference, so we do not need to calculate them), and

$$f_n(\boldsymbol{\theta}) \propto \exp(\boldsymbol{\lambda}_n^\top \boldsymbol{\phi}(\boldsymbol{\theta})) \ \ (0 \leq n \leq N),$$

where $\boldsymbol{\lambda}_n$ is the natural parameter and $\boldsymbol{\phi}(\boldsymbol{\theta})$ is sufficient statistics. For example, if we choose a Gaussian term, $f_n = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n)$, then the sufficient statistics is $\boldsymbol{\phi}(\boldsymbol{\theta}) = \{\boldsymbol{\theta}, \boldsymbol{\theta}\boldsymbol{\theta}^\top\}$. The moment is the expectation of the sufficient statistics.

We therefore approximate the joint distribution with

$$p(\boldsymbol{\theta}, \mathcal{D}) = p(\boldsymbol{\theta}) \prod_{n=1}^{N} p(\mathbf{y}_n|\boldsymbol{\theta}) \approx f_0(\boldsymbol{\theta}) \prod_{n=1}^{N} f_n(\boldsymbol{\theta}) \cdot \text{const}. \tag{14}$$

Because the exponential family is closed under multiplication and division, we can immediately obtain a closed-form approximate posterior $q(\boldsymbol{\theta}) \approx p(\boldsymbol{\theta}|\mathcal{D})$ by merging the approximation terms in the R.H.S of (14), which is still a distribution in the exponential family.

Then the task amounts to optimizing those approximation terms $\{f_n(\boldsymbol{\theta})|0 \leq n \leq N\}$. EP repeatedly conducts four steps to optimize each f_n.

- **Step 1.** Obtain the calibrated distribution that integrates the context information of f_n,

$$q^{\backslash n}(\boldsymbol{\theta}) \propto \frac{q(\boldsymbol{\theta})}{f_n(\boldsymbol{\theta})}, \tag{15}$$

  where $q(\boldsymbol{\theta})$ is the current posterior approximation.

- **Step 2.** Construct a tilted distribution to combine the true likelihood,

$$\widetilde{p}(\boldsymbol{\theta}) \propto q^{\backslash n}(\boldsymbol{\theta}) \cdot p(\mathbf{y}_n|\boldsymbol{\theta}). \tag{16}$$

  Note that if $n = 0$, we have $\widetilde{p}(\boldsymbol{\theta}) \propto q^{\backslash n}(\boldsymbol{\theta}) \cdot p(\boldsymbol{\theta})$.

- **Step 3.** Project the tilted distribution back to the exponential family,

$$q^*(\boldsymbol{\theta}) = \underset{q}{\text{argmin}} \ \ \text{KL}(\widetilde{p}\|q), \tag{17}$$

where $\mathrm{KL}(\cdot\|\cdot)$ is the Kullback-Leibler divergence, and q belongs to the exponential family. This can be done by moment matching,

$$\mathbb{E}_{q^*}[\boldsymbol{\phi}(\boldsymbol{\theta})] = \mathbb{E}_{\widetilde{p}}[\boldsymbol{\phi}(\boldsymbol{\theta})]. \tag{18}$$

That is, we compute the expected moment under $\widetilde{p}$, with which to set the parameters of q^*. For example, if $q^*(\boldsymbol{\theta})$ is a Gaussian distribution, then we need to compute $\mathbb{E}_{\widetilde{p}}[\boldsymbol{\theta}]$ and $\mathbb{E}_{\widetilde{p}}[\boldsymbol{\theta}\boldsymbol{\theta}^\top]$, with which to obtain the mean and covariance for $q^*(\boldsymbol{\theta})$. Accordingly, we obtain $q^*(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbb{E}_{\widetilde{p}}[\boldsymbol{\theta}], \mathbb{E}_{\widetilde{p}}[\boldsymbol{\theta}\boldsymbol{\theta}^\top] - \mathbb{E}_{\widetilde{p}}[\boldsymbol{\theta}]\mathbb{E}_{\widetilde{p}}[\boldsymbol{\theta}]^\top)$.

- **Step 4.** Update the approximation term by

$$f_n(\boldsymbol{\theta}) \propto \frac{q^*(\boldsymbol{\theta})}{q^{\setminus n}(\boldsymbol{\theta})}. \tag{19}$$

In practice, EP often updates all the f_n's in parallel. It iteratively runs the four steps until convergence. In essence, this is a fixed point iteration to optimize an energy function (a mini-max problem) (Minka, 2001a,b; Wainwright et al., 2008).

A.1.2 EP Method for Our Model

We now present how we develop an EP method to estimate the posterior distribution of the spike-and-slab variables, $\mathbf{s}$ and $\mathbf{w}$, for our model. Let us write down the joint distribution here (see (8) of the main paper),

$$p(\mathbf{s}, \mathbf{w}, \mathcal{D}) \propto \prod_{j=1}^{A} \mathrm{Bern}(s_j|\rho_0) \left(s_j \mathcal{N}(w_j|0, \sigma_0^2) + (1 - s_j)\delta(w_j)\right) \cdot \mathcal{N}(\mathbf{h}|\boldsymbol{\Phi}\mathbf{w}, \tau\mathbf{I}). \tag{20}$$

The mixture factors in the spike-and-slab prior (20) do not belong to the exponential family, and thereby make the posterior distribution analytically intractable. So we introduce an approximation,

$$s_j \mathcal{N}(w_j|0, \sigma_0^2) + (1 - s_j)\delta(w_j) \approx c_j \mathrm{Bern}(s_j|\sigma(\rho_j))\mathcal{N}(w_j|\mu_j, v_j). \tag{21}$$

where $\sigma(\cdot)$ is the sigmoid function. Substituting the above into (20), we can obtain the approximate posterior,

$$q(\mathbf{s}, \mathbf{w}) = \prod_{j=1}^{A} \mathrm{Bern}\left(s_j|\sigma(\widehat{\rho}_j)\right) \cdot \mathcal{N}(\mathbf{w}|\boldsymbol{\beta}, \boldsymbol{\Sigma}), \tag{22}$$

where

$$\begin{aligned} &\widehat{\rho}_j = \sigma^{-1}(\rho_0) + \rho_j, \;\; \boldsymbol{\Sigma} = \left(\mathrm{diag}^{-1}(\mathbf{v}) + \tau^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\right)^{-1}, \\ &\boldsymbol{\beta} = \boldsymbol{\Sigma}\left(\frac{\boldsymbol{\mu}}{\mathbf{v}} + \tau^{-1}\boldsymbol{\Phi}^\top\mathbf{h}\right), \end{aligned} \tag{23}$$

where $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_A]^\top$ and $\mathbf{v} = [v_1, \ldots, v_A]^\top$.

We optimize the approximation factor (21) for every operator j. To this end, we first compute the calibrated distribution (STEP 1; see (15)),

$$q^{\setminus j}(s_j, w_j) \propto \frac{q(s_j, w_j)}{\mathrm{Bern}(s_j|\sigma(\rho_j))\mathcal{N}(w_j|\mu_j, v_j)} = \mathrm{Bern}(s_j|\sigma(\rho^{\setminus j}))\mathcal{N}(w_j|\mu^{\setminus j}, v^{\setminus j}), \tag{24}$$

where according to the exponential family properties, we have

$$\begin{aligned} \rho^{\setminus j} &= \widehat{\rho}_j - \rho_j, \\ \left(v^{\setminus j}\right)^{-1} &= [\boldsymbol{\Sigma}]_{jj}^{-1} - (v_j)^{-1}, \\ \left(v^{\setminus j}\right)^{-1}\mu^{\setminus j} &= \frac{\beta_j}{[\boldsymbol{\Sigma}]_{jj}} - \frac{\mu_j}{v_j}. \end{aligned} \tag{25}$$

Note that β_j and $[\mathbf{\Sigma}]_{jj}$ are the marginal mean and variance of w_j, respectively, under the global posterior approximation $q(\mathbf{s}, \mathbf{w})$.

In STEP 2 (see (16)), we construct the titled distribution,

$$\begin{aligned}
&\widetilde{p}(s_j, w_j) \\
&\propto q^{\setminus j}(s_j, w_j) \cdot \left(s_j \mathcal{N}(w_j|0, \sigma_0^2) + (1 - s_j)\delta(w_j)\right) \\
&= \text{Bern}(s_j|\sigma(\rho^{\setminus j}))\mathcal{N}(w_j|\mu^{\setminus j}, v^{\setminus j}) \left(s_j \mathcal{N}(w_j|0, \sigma_0^2) + (1 - s_j)\delta(w_j)\right).
\end{aligned} \tag{26}$$

In STEP 3 (see (17)), we perform moment matching. That is, we compute the moment of s_j and w_j under $\widetilde{p}$. To this end, we use the following identity about the convolution of two Gaussians,

$$\mathcal{N}(\boldsymbol{\theta}|\mathbf{m}_1, \Sigma_1)\mathcal{N}(\boldsymbol{\theta}|\mathbf{m}_2, \Sigma_2) = \mathcal{N}(\boldsymbol{\theta}|\hat{\mathbf{m}}, \hat{\Sigma})\mathcal{N}(\mathbf{m}_1|\mathbf{m}_2, \Sigma_1 + \Sigma_2)$$

where

$$\hat{\Sigma} = \left(\Sigma_1^{-1} + \Sigma_2^{-1}\right)^{-1}, \quad \hat{\mathbf{m}} = \hat{\Sigma}(\Sigma_1^{-1}\mathbf{m}_1 + \Sigma_2^{-1}\mathbf{m}_2).$$

It is therefore straightforward to obtain the marginal tilted distribution,

$$\widetilde{p}(s_j) \propto \left(\sigma(\rho^{\setminus j})\mathcal{N}(\mu^{\setminus j}|0, v^{\setminus j} + \sigma_0^2)\right) s_j + \left((1 - \sigma(\rho^{\setminus j}))\mathcal{N}(\mu^{\setminus j}|0, v^{\setminus j})\right)(1 - s_j). \tag{27}$$

We need to normalize the coefficients to get the probabilities, so we have

$$\widetilde{p}(s_j) = \text{Bern}(s_j|\rho^*), \tag{28}$$

where

$$\rho^* = \rho^{\setminus j} + \log \frac{\mathcal{N}(\mu^{\setminus j}|0, v^{\setminus j} + \sigma_0^2)}{\mathcal{N}(\mu^{\setminus j}|0, v^{\setminus j})}. \tag{29}$$

Now, we need to compute the first and second moments of w_j in (26). We first marginalize out s_j,

$$\widetilde{p}(w_j) = \frac{1}{Z}\left(\sigma(\rho^{\setminus j})\mathcal{N}(w_j|\mu^{\setminus j}, v^{\setminus j})\mathcal{N}(w_j|0, \sigma_0^2) + (1 - \sigma(\rho^{\setminus j}))\mathcal{N}(w_j|\mu^{\setminus j}, v^{\setminus j})\delta(w_j)\right). \tag{30}$$

Look at the normalizer, it is the same as the one in (27). Then we can obtain the analytical form,

$$\widetilde{p}(w_j) = \sigma(\rho^*)\mathcal{N}(w_j|\widetilde{\mu}, \widetilde{v}) + (1 - \sigma(\rho^*))\frac{\mathcal{N}(w_j|\mu^{\setminus j}, v^{\setminus j})}{\mathcal{N}(\mu^{\setminus j}|0, v^{\setminus j})}\delta(w_j), \tag{31}$$

where

$$\begin{aligned}
\widetilde{v}^{-1} &= {v^{\setminus j}}^{-1} + \sigma_0^{-2}, \\
\widetilde{v}^{-1}\widetilde{\mu} &= \frac{\mu^{\setminus j}}{v^{\setminus j}}.
\end{aligned} \tag{32}$$

Now, we can compute the moments of (31), which is straightforward:

$$\begin{aligned}
\mu_j^* \triangleq \mathbb{E}_{\widetilde{p}}[w_j] &= \sigma(\rho^*)\widetilde{\mu}, \\
\mathbb{E}_{\widetilde{p}}[(w_j)^2] &= \sigma(\rho^*)\left(\widetilde{v} + \widetilde{\mu}^2\right), \\
v_j^* \triangleq \text{cov}_{\widetilde{p}}(w_j) &= \sigma(\rho^*)\left(\widetilde{v} + (1 - \sigma(\rho^*))\widetilde{\mu}^2\right).
\end{aligned} \tag{33}$$

Then, we obtain the global posterior,

$$q^*(s_j, w_j) = \text{Bern}(s_j|\sigma(\rho^*))\mathcal{N}(w_j|\mu_j^*, v_j^*). \tag{34}$$

The approximation factor in (21) is updated by $\frac{q^*(s_j,w_j)}{q^{\setminus j}(s_j,w_j)}$ (STEP 4; see (19)), which gives

$$\begin{aligned}\rho_j &= \rho^* - \rho^{\setminus j},\\ (v_j)^{-1} &= \left(v_j^*\right)^{-1} - \left(v^{\setminus j}\right)^{-1},\\ \frac{\mu_j}{v_j} &= \frac{\mu_j^*}{v_j^*} - \frac{\mu^{\setminus j}}{v^{\setminus j}}.\end{aligned} \tag{35}$$

In each iteration, we in parallel update the approximation factor (21) for every operator j. Initially, we set $\rho_j = 0$, $\mu_j = 0$ and $v_j = 10^6$ (so at the beginning, these factors are uninformative and nearly constant one). We repeat the EP iterations until convergence, and then return the posterior approximation (22).

A.2 Experimental Details

SINDy. We used the PySINDy library (`https://pysindy.readthedocs.io/en/latest/index.html#id4`) in the experiment. PySINDy not only includes the original implementation of SINDy with sequential threshold ridge regression (STRidge), but it also supports other sparse promoting techniques, including L0 and L1 thresholding. The current library includes six optimizers: STLSQ, SR3-L0, SR3-L1, SSR, SSR-residual, and FROLS. We tried all the six optimizers and tuned the hyperparmeters to achieve the best performance with our best efforts. Specifically, we tuned the tolerance level in the range $[10^{-15}, 10^{-1}]$, regularization strength (for L0, L1 and L2) from range $[10^{-1}, 10^3]$, normalization (on/off), the max number of iterations from range $[10^4, 10^5]$, and threshold for STLSQ from the range $[10^{-5}, 0.5]$.

PINN-SR. We used the implementation (`https://github.com/isds-neu/EQDiscovery`) by the authors. PINN-SR first performs a pre-training of the NN from the training data, and then conducts alternating direction optimization (ADO) for joint equation discovery and solution learning. We tuned the number of NN layers from the range $[3, 8]$, the NN width from range $[20, 100]$, the number of iterations for ADO from range $[12, 20]$, regularization strength from $[10^{-7}, 10^{-1}]$, L-BFGS pre-training iterations from [10K, 80K], L-BFGS ADO training iterations from [1K, 2K], learning rate of ADAM from $[10^{-4}, 10^{-3}]$, and d_tol hyper-parameter from $[10^{-5}, 5]$. We varied the number of collocation points from [10K, 160K].

BSL. We used the original implementation shared by the authors. We tuned the number of knots from range [30, 300] for each input dimension. We tuned the number of collocation points per dimension from [50, 300]. The weight value threshold was tuned from range $[10^{-5}, 0.5]$, and the regularization strength from range $[10^{-6}, 10^{-1}]$. We also tuned the pre-training iteration number from $[10^4, 10^5]$ and the number of ADO iterations from $[8, 20]$. The d_tol hyper-parmeter was tuned from $[10^{-5}, 0.5]$.

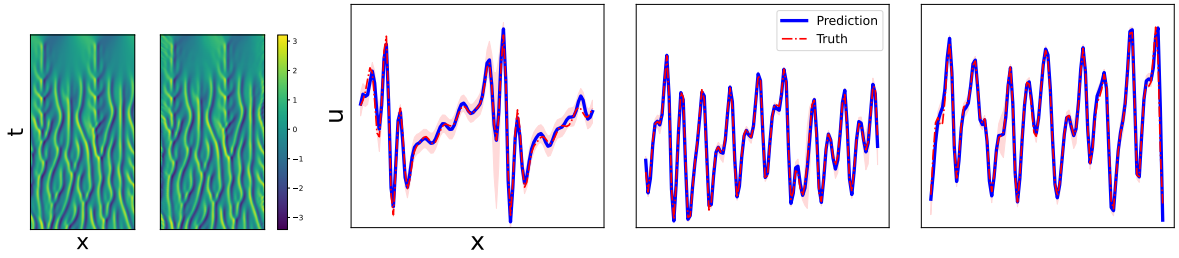
KBASS. We slightly tuned two hyper-parameters, the slab variance σ_0^2 from 0.5 to 5000 (very flat Gaussian slab), and the equation likelihood variance τ from 10^{-4} to 1. The data likelihood variance v in (6) (of the main paper) is obtained via cross-validation in the initial training (see the first step in Algorithm 1). The performance is not sensitive to other hyperparameters and so we fixed them. Specifically, the maximum number of EP-EM iteration was set to 200K and the tolerance level is 10^{-8}. We also used relative change of natural parameters less than 10^{-5} as the condition to stop the EP inference at the E step. The solution estimation $\mathcal{U}$ was initialized as zero. We used an evenly-spaced mesh for each testing case, and the size is summarized in Table 5.

Test equation	Mesh size
VDP	200
Lorenz 96	300
Burger's ($\nu = 0.1$)	160×160
Burger's ($\nu = 0.01$)	180×180
Burger's ($\nu = 0.005$)	180×180
KS	200×200
Allen-cahn	200×200
Predator-Prey	500

Table 5: The mesh size used by KBASS.

Method	21 examples		
	RMSE	Recall	Precision
SINDy	F	0.75	0.6
PINN-SR	F	0.25	0.5
BSL	0.0304	1	1
KBASS	**0.0244**	**1**	**1**

Table 6: Performance of discovering a real-world predator-prey system.



(a) Left: truth; right: prediction (b) t = 24.1 (c) t = 75.4 (d) t = 128.1

Figure 5: Solution estimate for the KS equation with 20% noise on training data.

A.3 Real-world Predator-Prey Data

We followed (Sun et al., 2022) to test the equation discovery from a real-world dataset collected from a predator-prey system between lynx and hares. The dataset is the population of the lynx and hares from 1900 to 1920 at Hudson Bay Company, presented in Table 11 of (Sun et al., 2022). The dataset is very noisy. The governing equation from mathematical analysis is given by

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= 0.4807x - 0.0248xy, \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= -0.9272y + 0.0276xy, \end{aligned} \tag{36}$$

which we used as the golden standard. We used the same setting as in (Sun et al., 2022) to test KBASS. The performance of the discovery is reported in Table 6. As we can see, KBASS not only correctly discovered the equation form, but also it achieves the smallest RMSE in the operator weight estimation. We also show the solution prediction of KBASS in Fig. 7.

A.4 Laplace's approximation for Posterior Estimation

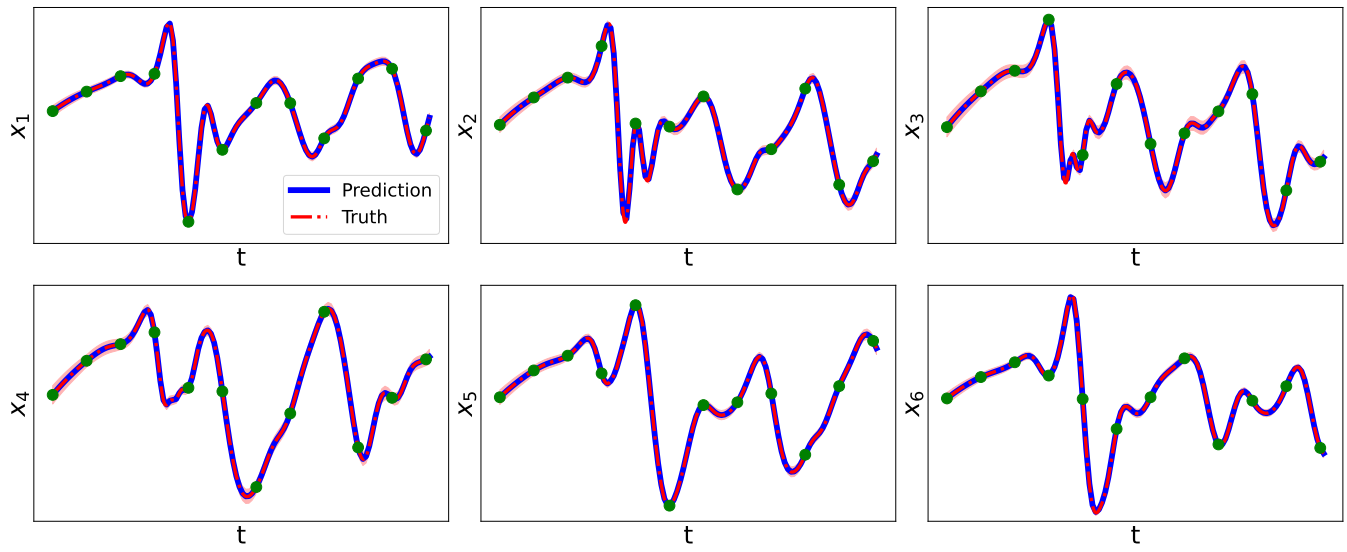
Given the loss L and parameters $\boldsymbol{\beta}$, the Laplace's approximation (Walker, 1969; MacKay, 2003) first minimizes the loss function to obtain the optimal parameter estimate $\boldsymbol{\beta}^*$, and then constructs a multi-variate Gaussian posterior approximation,

$$q(\boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta}|\boldsymbol{\beta}^*, \mathbf{H}^{-1}), \tag{37}$$

where

$$\mathbf{H} = \left.\frac{\partial^2 L}{\partial \boldsymbol{\beta}^2}\right|_{\boldsymbol{\beta}=\boldsymbol{\beta}^*} \tag{38}$$

is the Hessian matrix at $\boldsymbol{\beta}^*$. The approximation can be derived via the second-order Taylor approximation of L at the optimum $\boldsymbol{\beta}^*$. For the ODE systems, we applied the standard Laplace's approximation to obtain the posterior estimate of our solution prediction $\mathcal{U}$. For the other equations, like Burger's and KS equations, since $\mathrm{vec}(\mathcal{U})$ is high-dimensional, computing the full Hessian matrix is very costly. Therefore, we use the block-diagonal Hessian (Ritter et al., 2018). We compute the Hessian for each time slice and then approximate the posterior for the solution at that slice.



(a) (b) (c)

Figure 6: Solution estimate for the Lorenz 96 system using 12 training examples.

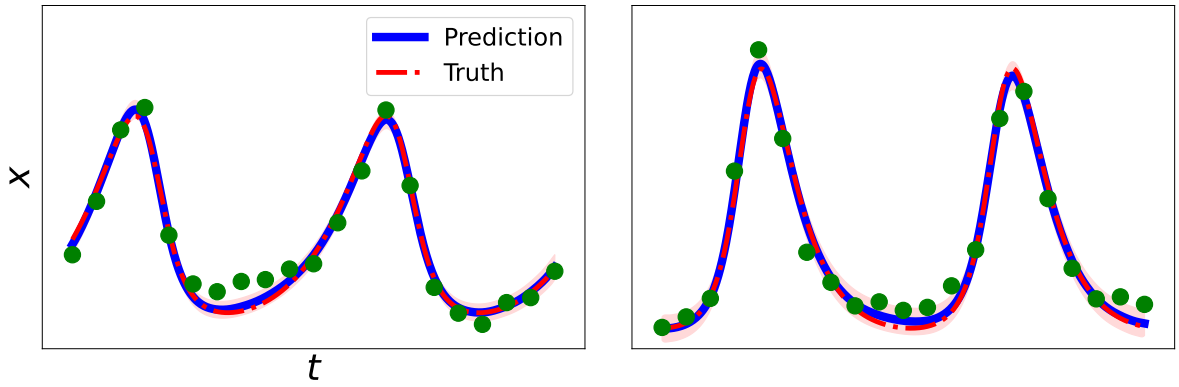


Figure 7: Solution estimate for the real-world predator-prey system.

A.5 Running Time

We examined the running time of each method on a Linux workstation with a NVIDIA GeForce RTX 3090 GPU, with a memory size of 24576 MB. We tested on the discovery of VDP, Lorenz 96, Burger's and KS equations. The results are reported in Table 8. It can be seen that KBASS is consistently faster than BSL: 4.2x, 1.9x, 7.3x and 2.7x faster in the four testing cases, respectively. Although PINN-SR is faster than KBASS in VDP and Lorenz 96, it failed to discover the correct equations. For KS, PINN-SR used 19.5x time and but still failed to recover the equation. SINDy is the fastest among all the four methods. It is reasonable, because SINDy does not estimate the solution function, and only performs sparse linear regression.

Test case	num. of examples	RMSE
VDP	54	0.127
Lorenz 96	250	0.229
Burger's ($\nu = 0.1$)	40×40	0.0398
KS	512×751	0.0173

Table 7: SINDy with more training data for successful discovery. Note that the data is noise free.

Test case	KBASS	BSL	PINN-SR	SINDy
VDP	643	2700	421 (F)	0.5 (F)
Lorenz 96	3994	7438	2086 (F)	2 (F)
Burger's ($\nu = 0.1$)	99	720	853	2 (F)
KS	1174	3150 (F)	22934 (F)	0.5 (F)

Table 8: Running time (in seconds) on a Linux workstation with a NVIDIA GeForce RTX 3090 GPU with 24576 MB memory. The number of training examples used for each test case is 25, 50, 20×20 and 40×40 for VDP, Lorenz 96, Burger's ($\nu = 0.1$) and KS, respectively. The noise level is zero. "F" means failed to discover the equation.

A.6 Discovered Equations

We show the examples of discovered equations by KBASS and the competing methods in Table 9, 10, 11, 12, 13, 14, 15 and 16.

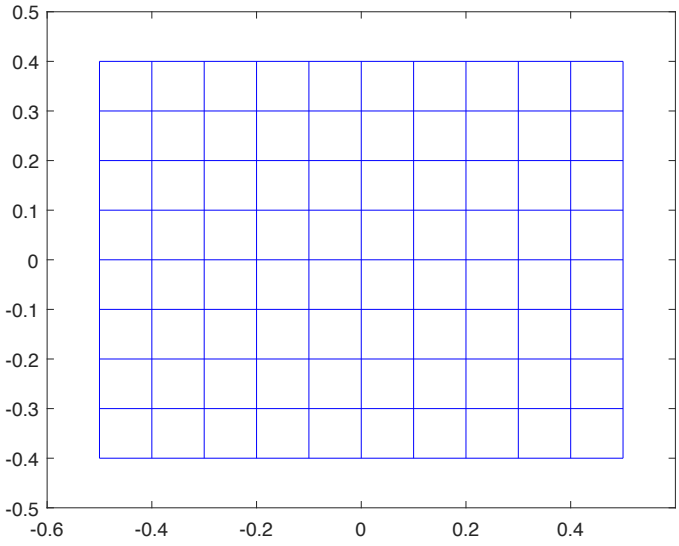
Name	Equation
True	$x_t = y$
	$y_t = 2.5y - x - 2.5x^2y$
SINDy	$x_t = 0.807y$
	$y_t = -0.504x$
PINN-SR	$x_t = 0.848y + 1.23y^2 + 0.424y^3$
	$y_t = 1.14y - 0.531x - 0.742x^2y$
BSL	$x_t = 1.236y$
	$y_t = -0.622x$
KBASS	$x_t = y$
	$y_t = 2.47y - 0.992x - 2.49x^2y$

(a) zero noise

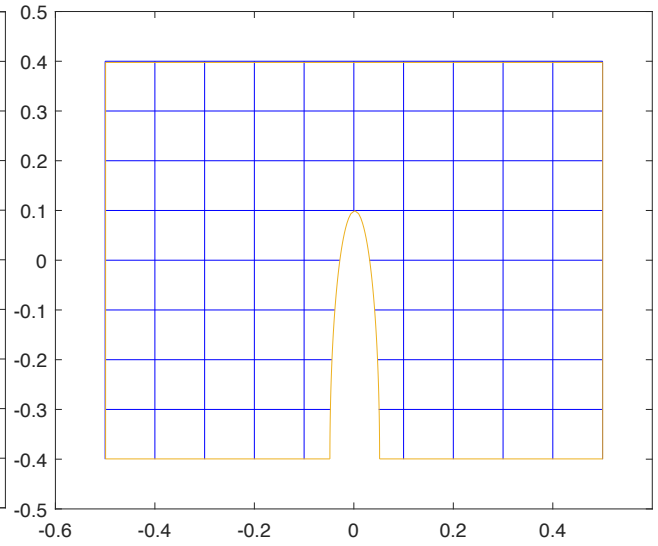
Name	Equation
True	$x_t = y$
	$y_t = 2.5y - x - 2.5x^2y$
SINDy	$x_t = 0.724y$
	$y_t = -0.524x$
PINN-SR	$x_t = 2.26y + 0.208y^3 - 0.188y^4 - 0.303x^2 + ...$
	$y_t = 2.38y - 1.5x^2y + 1.52y^2 - 0.662y^4 + ...$
BSL	$x_t = 0.505y + 0.937xy - 0.588x^2y + ...$
	$y_t = 2.35xy - 0.913x^2y - 0.427x^3 + ...$
KBASS	$x_t = 0.961y$
	$y_t = 2.13y - 1.04x - 2.44x^2y$

(b) 20% noise

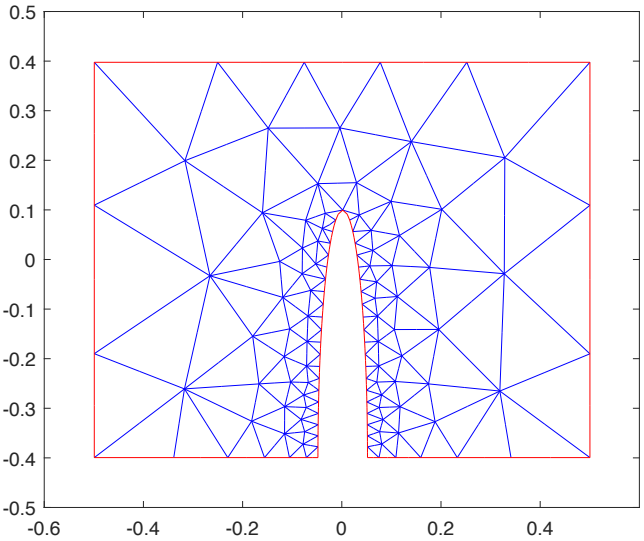
Table 9: Discovery result for the VDP system with 10 measurement examples.



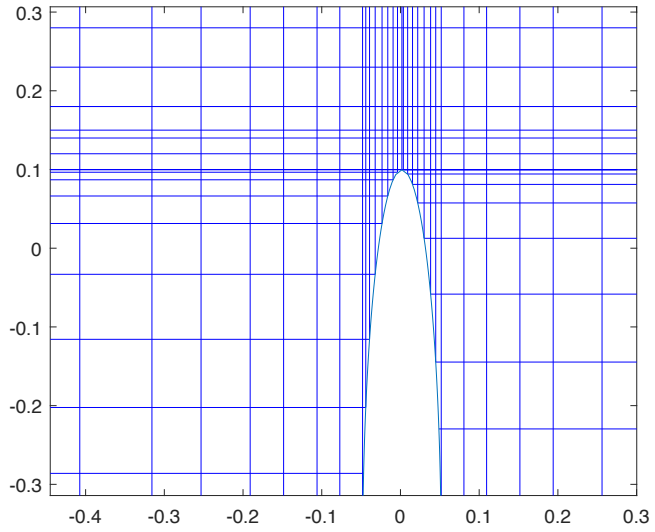
(a) Regular mesh



(b) Regular mesh on a special domain



(c) An example of finite element design.



(d) Unevenly-spaced mesh design for KBASS.

Figure 8: An example of the mesh design. By varying the density regions of the mesh (d), our method can adapt to the geometry of specific domains, as the traditional numerical methods do (c).

Name	Equation
True	$x_{1t} = x_2x_6 - x_5x_6 - x_1 + 8$, $x_{2t} = x_1x_3 - x_1x_6 - x_2 + 8$
	$x_{3t} = x_2x_4 - x_1x_2 - x_3 + 8$, $x_{4t} = x_3x_5 - x_2x_3 - x_4 + 8$
	$x_{5t} = x_4x_6 - x_3x_4 - x_5 + 8$, $x_{6t} = x_1x_5 - x_4x_5 - x_6 + 8$
SINDy	$x_{1t} = 0.688x_1 + 0.016x_4^2x_5 - 0.06x_3x_4^2$, $x_{2t} = 0.795x_3$
	$x_{3t} = 0$, $x_{4t} = 0$
	$x_{5t} = 1.33$, $x_{6t} = 1.38x_1 - 0.067x_3x_4^2$
PINN-SR	$x_{1t} = -0.0288x_1x_2x_6$, $x_{2t} = -0.01371x_2x_6$
	$x_{3t} = -0.0161x_1x_2x_6$, $x_{4t} = -0.015x_1x_2x_6$
	$x_{5t} = -0.0117x_1x_2x_6$, $x_{6t} = -0.0117x_1x_2x_6$
BSL	$x_{1t} = 0.944x_2$, $x_{2t} = 0$
	$x_{3t} = -2.74 - 1.76x_1 + 1.24x_3 + ...$, $x_{4t} = 6.35$
	$x_{5t} = -1.8$, $x_{6t} = -4.52$
KBASS	$x_{1t} = x_2x_6 - x_5x_6 - 0.991x_1 + 7.96$, $x_{2t} = 0.999x_1x_3 - 0.999x_1x_6 - x_2 + 7.94$
	$x_{3t} = x_2x_4 - x_1x_2 - 0.985x_3 + 7.92$, $x_{4t} = x_3x_5 - x_2x_3 - 0.995x_4 + 7.98$
	$x_{5t} = x_4x_6 - x_3x_4 - 0.999x_5 + 7.97$, $x_{6t} = 0.995x_1x_5 - 0.993x_4x_5 - 1.01x_6 + 7.93$

(a) zero noise

Name	Equation
True	$x_{1t} = x_2x_6 - x_5x_6 - x_1 + 8$, $x_{2t} = x_1x_3 - x_1x_6 - x_2 + 8$
	$x_{3t} = x_2x_4 - x_1x_2 - x_3 + 8$, $x_{4t} = x_3x_5 - x_2x_3 - x_4 + 8$
	$x_{5t} = x_4x_6 - x_3x_4 - x_5 + 8$, $x_{6t} = x_1x_5 - x_4x_5 - x_6 + 8$
SINDy	$x_{1t} = 0.169x_1 + 0.714x_2 - 0.053x_3x_4^2$, $x_{2t} = 0$
	$x_{3t} = 0$, $x_{4t} = 0$
	$x_{5t} = 3.53 - 0.041x_2x_4^2$, $x_{6t} = -0.044x_3x_4^2$
PINN-SR	$x_{1t} = 0.587x_1x_4 - 2.85x_2x_4 - 0.206x_1x_2^2 + ...$, $x_{2t} = 0.0997x_1x_5 + 0.779x_2x_6 + 0.209x_1x_3^2 + ...$
	$x_{3t} = -0.157x_1x_5 + 1.1x_2x_6 + 0.0907x_1x_2^2 + ...$, $x_{4t} = 0.375x_1x_4 - 0.187x_2x_5 + 0.0273x_1^2x_2 + ...$
	$x_{5t} = -0.923x_1x_4 + 0.07x_1x_2^2 + 0.263x_1x_3^2 + ...$, $x_{6t} = -0.923x_1x_4 + 0.07x_1x_2^2 + 0.263x_1x_3^2 + ...$
BSL	$x_{1t} = 10.8x_2$, $x_{2t} = -18.7x_6$
	$x_{3t} = 0.578$, $x_{4t} = 15.9x_6$
	$x_{5t} = 2.1$, $x_{6t} = 13.8x_2$
KBASS	$x_{1t} = 0.933x_2x_6 - 0.932x_5x_6 - 1.16x_1 + 9.22$, $x_{2t} = 1.06x_1x_3 - 0.969x_1x_6 - 1.28x_2 + 7.26$
	$x_{3t} = 0.906x_2x_4 - 0.988x_1x_2 - 0.594x_3 + 6.75$, $x_{4t} = 1.1x_3x_5 - 1.05x_2x_3 - 1.26x_4 + 8.37$
	$x_{5t} = 0.919x_4x_6 - 0.947x_3x_4 - 0.81x_5 + 7.47$, $x_{6t} = 0.985x_1x_5 - 0.992x_4x_5 - 1.02x_6 + 9.66$

(b) 10% noise

Table 10: Discovery result for the Lorenz 96 system with 12 measurement examples.

Name	Equation
True	$u_t = -uu_x + 0.1u_{xx}$
SINDy	$u_t = -0.135u_x - 0.186uu_x - 0.486uu_{xx} + ...$
PINN-SR	$u_t = -uu_x + 0.1u_{xx}$
BSL	$u_t = -0.218uu_x$
KBASS	$u_t = -uu_x + 0.1u_{xx}$

(a) zero noise

Name	Equation
True	$u_t = -uu_x + 0.1u_{xx}$
SINDy	$u_t = -0.2u_x + 0.147u_{xx} - 0.715uu_x$
PINN-SR	$u_t = -1.03uu_x + 0.0313u_{xx} + 0.48u^3 + ...$
BSL	$u_t = -0.372u_x$
KBASS	$u_t = -1.04uu_x + 0.0873u_{xx}$

(b) 20% noise

Table 11: Discovery result for the Burgers' Equation ($\nu = 0.1$) with 10×10 measurement examples.

Name	Equation
True	$u_t = -uu_x + 0.01u_{xx}$
SINDy	$u_t = -0.177u_x - 1.101uu_x + 0.143uu_{xxx} + ...$
PINN-SR	$u_t = -uu_x + 0.0593u_{xx} - 0.0515u_x + ...$
BSL	$u_t = -1.72uu_x + 6.023u^3u_x - 5.38u^4u_x$
KBASS	$u_t = -0.998uu_x + 0.01u_{xx}$

(a) zero noise.

Name	Equation
True	$u_t = -uu_x + 0.01u_{xx}$
SINDy	$u_t = -0.192u_x - 0.745uu_x + 0.064uu_{xxx} + ...$
PINN-SR	$u_t = -0.697uu_x - 0.206u_x$
BSL	$u_t = -0.406u_x$
KBASS	$u_t = -0.992uu_x + 0.00963u_{xx}$

(b) 20% noise

Table 12: Discovery result for the Burgers' Equation ($\nu = 0.01$) with 50×50 measurement examples.

Name	Equation
True	$u_t = -uu_x + 0.005u_{xx}$
SINDy	$u_t = -0.293u_x - 0.651uu_x + 0.077uu_{xxx} + ...$
PINN-SR	$u_t = -0.958uu_x + 0.262uu_{xx} - 1.06u^2u_{xx} + ...$
BSL	$u_t = -3.24uu_x + 8.77u^2u_x - 9.55u^3u_x + ...$
KBASS	$u_t = -uu_x + 0.00634u_{xx}$

(a) zero noise.

Name	Equation
True	$u_t = -uu_x + 0.005u_{xx}$
SINDy	$u_t = -0.222u_x + 0.022u_{xx} - 0.266uu_x + ...$
PINN-SR	$u_t = -0.426u_x$
BSL	$u_t = -9.15u^2u_x + 21.4u^3u_x - 13.7u^4u_x$
KBASS	$u_t = -1.0044uu_x + 0.0066u_{xx}$

(b) 20% noise

Table 13: Discovery result for the Burgers' Equation ($\nu = 0.005$) with 50×50 measurement examples.

Name	Equation
True	$u_t = -uu_x - u_{xx} - u_{xxxx}$
SINDy	$u_t = -0.068uu_x + 0.049uu_{xxx} - 0.02u^2u_{xxxx} + ...$
PINN-SR	$u_t = 0.0856u_x + 0.0684uu_{xx} - 0.0566u^2u_{xxx}$
BSL	$u_t = -0.115u_x$
KBASS	$u_t = -0.988uu_x - 0.986u_{xx} - 0.988u_{xxxx}$

(a) zero noise

Name	Equation
True	$u_t = -uu_x - u_{xx} - u_{xxxx}$
SINDy	$u_t = -0.035u - 0.104uu_x + 0.024uu_{xxx} + ...$
PINN-SR	$u_t = 0.0717uu_{xxx} + 0.0374uu_{xx} - 0.034u_{xx}$
BSL	$u_t = -0.115u_x$
KBASS	$u_t = -0.97uu_x - 0.978u_{xx} - 0.977u_{xxxx}$

(b) 20% noise

Table 14: Discovery result for the KS Equation with 40×40 measurement examples.

Name	Equation
True	$u_t = 0.0001u_{xx} + 5u - 5u^3$
SINDy	$u_t = 4.99u - 4.99u^3$
PINN-SR	$u_t = 4.16u - 3.7u^3 + 0.043u^4 + ...$
BSL	$u_t = 4.53u - 4.33u^3 + 0.128u^2$
KBASS	$u_t = 0.0001u_{xx} + 4.99u - 4.99u^3$

(a) zero noise.

Name	Equation
True	$u_t = 0.0001u_{xx} + 5u - 5u^3$
SINDy	$u_t = 3.85u - 3.49u^3$
PINN-SR	$u_t = 2.94u - 2u^3 + 0.063u^4 + ...$
BSL	$u_t = -0.0099u_x + 4.55u - 4.42u^3 + ...$
KBASS	$u_t = 0.000104u_{xx} + 4.79u - 4.76u^3$

(b) 10% noise

Table 15: Discovery result for the Allen-cahn Equation with 26×101 measurement examples.

Name	Equation
True	$x_t = 0.48x - 0.0248xy$
	$y_t = -0.927y + 0.0276xy$
SINDy	$x_t = 0.581x - 0.0261xy$
	$y_t = 0.255x - 0.27y$
PINN-SR	$x_t = -13.9y$
	$y_t = -0.114y$
BSL	$x_t = 0.512x - 0.0266xy$
	$y_t = -0.926y + 0.0279xy$
KBASS	$x_t = 0.506x - 0.0255xy$
	$y_t = -0.925y + 0.0273xy$

Table 16: Discovery result for the real-world predator-prey system with 21 observed examples.