entropy

MDPI

Article

Learning Interactions in Reaction Diffusion Equations by Neural Networks

Sichen Chen [1], Nicolas J-B. Brunel [2,3], Xin Yang [4] and Xinping Cui [1,*]

[1] Department of Statistics, University of California, Riverside, CA 92521, USA
[2] ENSIIE & Laboratoire de Mathématiques et Modélisation d'Evry, Université Paris Saclay, 91025 Evry, France
[3] Quantmetry, 75008 Paris, France
[4] Department of Mathematics, University of California, Riverside, CA 92521, USA
* Correspondence: xinping.cui@ucr.edu

Abstract: Partial differential equations are common models in biology for predicting and explaining complex behaviors. Nevertheless, deriving the equations and estimating the corresponding parameters remains challenging from data. In particular, the fine description of the interactions between species requires care for taking into account various regimes such as saturation effects. We apply a method based on neural networks to discover the underlying PDE systems, which involve fractional terms and may also contain integration terms based on observed data. Our proposed framework, called Frac-PDE-Net, adapts the PDE-Net 2.0 by adding layers that are designed to learn fractional and integration terms. The key technical challenge of this task is the identifiability issue. More precisely, one needs to identify the main terms and combine similar terms among a huge number of candidates in fractional form generated by the neural network scheme due to the division operation. In order to overcome this barrier, we set up certain assumptions according to realistic biological behavior. Additionally, we use an L^2-norm based term selection criterion and the sparse regression to obtain a parsimonious model. It turns out that the method of Frac-PDE-Net is capable of recovering the main terms with accurate coefficients, allowing for effective long term prediction. We demonstrate the interest of the method on a biological PDE model proposed to study the pollen tube growth problem.

Keywords: neural networks; deep learning; non-linear reaction–diffusion equations; model discovery; sparse regression; multiple testing

Citation: Chen, S.; Brunel, N.J-B.; Yang, X.; Cui, X. Learning Interactions in Reaction Diffusion Equations by Neural Networks. *Entropy* **2023**, *25*, 489. https://doi.org/10.3390/e25030489

Academic Editors: Nicolas Bousquet and Patrick Gallinari

Received: 6 February 2023
Revised: 7 March 2023
Accepted: 8 March 2023
Published: 11 March 2023

1. Introduction

Two-component reaction–diffusion systems often model the interaction of two chemicals, leading to the formation of non-uniform spatial patterns of chemical concentration or morphogenesis under certain conditions due to chemical reactions and spreading. Since Turing's groundbreaking work [1], reaction–diffusion systems have been extensively used in developmental biology modeling. For example, let $u = u(x, y, t)$ and $v = v(x, y, t)$ represent the concentration of two chemical species, which may either enhance or suppress each other depending on the context. The system of u and v can be modeled as follows:

$$\begin{cases} \partial_t u &= d_0 \Delta u + N_1(u, v), \\ \partial_t v &= d_1 \Delta v + N_2(u, v), \end{cases} \tag{1}$$

where $\Delta = \partial_x^2 + \partial_y^2$ denotes the Laplacian operator, N_1 and N_2 are interactions between u and v. The functions N_1 and N_2 are sums of various reaction terms that can be derived from physical or chemical principles such as mass-action laws, Michaelis–Menten kinetics, or products that represent some competition, cooperation effects. We refer the readers to ([2], Section 2.2) for more discussions. Hence, N_1 and N_2 are sums of meaningful

functions that represent specific mechanisms: if we are able to identify these terms and discover the explicit formulas for N_1 and N_2, then we can learn more about the nature of the interactions and predict future behaviors well. This situation arises commonly in biological applications such as chemotaxis, pattern formation in developmental biology, and also the cell polarity phenomenon [3,4].

Cell polarity plays a vital role in cell growth and function for many cell types, affecting cell migration, proliferation, and differentiation. A classic example of polar growth is pollen tube growth, which is controlled by the Rho GTPase (ROP1) molecular switch. Recent studies have revealed that the localization of active ROP1 is regulated by both positive and negative feedback loops, and calcium ions play a role in ROP1's negative feedback mechanism. Initially, ROP1 is inside the membrane. During positive feedback (rate k_{pf}), some of the ROP1 enters the membrane. At the same time, negative feedback (rate k_{nf}) causes some of it to return inside the membrane while the rest diffuse on the membrane (rate D_r). Calcium ions follow a similar process with positive rate k_{ac}, negative rate k_{dc}, and diffusion rate D_c. In [5,6], the following 2D reaction–diffusion system (2) is introduced:

$$\begin{cases} R_t = k_{pf} R^{\alpha} \left(R_{tot} - \int_{-L}^{L} R(x,t)dx \right) - k_{nf}\, g(C)\, R + D_r R_{xx}, \\ C_t = k_{ac} R - k_{dc} C + D_c C_{xx}, \\ R_x(-L,t) = R_x(L,t) = 0, \quad C_x(-L,t) = C_x(L,t) = 0, \\ R(x,0) = R_0(x), \quad C(x,0) = C_0(x). \end{cases} \tag{2}$$

with suitable initial and boundary conditions being proposed to quantitatively describe such spatial and temporal connection between ROP1 and calcium ions, leading to rapid oscillations in their distributions on the cell membrane. Here, $R = R(x,t)$, $C = C(x,t)$, and R_t, C_t, R_x, R_{xx}, C_x and C_{xx} are abbreviated notations for partial derivatives with respect to the time t or to the spatial variable x. Moreover, the non-linear function $g(C)$ characterizes how calcium ions play a role in ROP1's negative feedback loop. Specifically, the active ROP1 causes an increase in Ca^{2+} levels, leading to a reduction in ROP1 activity and a decrease in its levels. Meanwhile, the flow of Ca^{2+} slows down as ROP1 drops. Ref. [6] proposed the equation $g(C) = \frac{C^2}{C^2 + k_c^2}$ to describe such spatial–temporal patterns of calcium, where k_c is a positive constant. Based on this model, ref. [6] developed a modified gradient matching procedure for parameter estimation, including k_{nf} and k_c. However, it requires that $g(C)$ in (2) is a known function. In this work, we propose to apply neural network methods to uncover the function $g(C)$ or more broadly, to learn interaction terms N_1 and N_2 in general reaction-diffusion PDEs (1), which may contain fractional expressions (Figure 1).

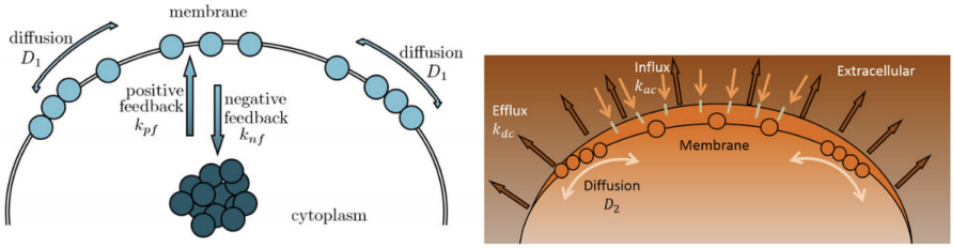


Figure 1. ROP1 and Ca^{2+} polarization dynamics. **Left**: ROP1 dynamics; **Right**: Ca^{2+} dynamics.

In the past decade, the artificial intelligence community has focused increasingly on neural networks, which have become crucial in many applications, especially PDEs. Deep learning-based approaches to PDEs have made substantial progress and are well-studied, both for forward and inverse problems. For forward problems with appropriate initial and boundary conditions in various domains, several methods have been developed to accurately predict dynamics (e.g., [7–17]). For inverse problems, there are two classes of approaches. The first class of approaches focuses on inferring coefficients from known data (e.g., [7,10,12,15,18,19]). An example of this is the widely known PINN (Physics-informed

Neural Networks) method [10], which uses PDEs in the loss function of neural networks to incorporate scientific knowledge. Ref. [7] improved the efficiency of PINNs with the residual-based adaptive refinement (RAR) method and created a library of open-source codes for solving various PDEs, including those with complex geometry. However, this method is only capable of estimating coefficients for fixed known terms in PDEs, and may not work well for discovering hidden PDE models. Although [9] extended the PINN method to find unknown dynamic systems, the nonlinear learner function remains a black-box and no explicit expressions of the discovered terms in the predicted PDE are available, making it difficult to interpret their physical meaning. The second class of approaches not only estimates coefficients, but also discovers hidden terms (e.g., [16,17,20–26]). An example is the PDE-Net method [16], which combines numerical approximations of convolutional differential operators with symbolic neural networks. PDE-Net can learn differential operators through convolution kernels, a natural method for solving PDEs that has been well-studied in [27]. This approach is capable of recovering terms in PDE models with explicit expressions and relatively accurate coefficients, but often produces many noisy terms that lack interpretation. In order to produce parsimonious models, refs. [25,26] proposed to create a regression model with the response variable $\partial_t u$, and a matrix Θ with a collection of spatial and polynomial derivative functions (e.g., $u, \partial_x u, u\partial_x u$): $\partial_t u = \Theta\xi$. The estimation of differential equations by modeling the time variations of the solution is known to produce consistent estimates [28]. In addition, the Ridge regression with hard thresholding can be used to approximate the coefficient vector ξ. This sparse regression-based method generally results in a PDE model with accurately predicted terms and high accuracy coefficients. However, few existing studies have focused on effectively recovering interaction terms in the fractional form (say one polynomial term divided by another polynomial term) in hidden partial differential equations, which is the focus of this paper.

Previous methods for identifying the hidden terms in reaction–diffusion partial differential equation models have mostly focused on polynomial forms. However, as indicated in Equation (2), the model for ROP1 and calcium ion distribution also involves fractional and integral forms, which can pose identifiability issues when combined with polynomial forms. Furthermore, we want to attain a parsimonious model, as the interpretability of the PDE model is important for biologists to comprehend biological behavior and phenomena revealed by the model.

In this paper, we utilize a combination of a modified PDE-Net method (which adds fractional and integration terms to the original PDE-Net approach), an L^2 norm term selection criterion, and an appropriate sparsity regression. This combination proves to produce meaningful and stable terms with accurate estimation of coefficients. For ease of reference to this combination, we call it Frac-PDE-Net.

The paper is organized as follows. In Section 2, we explain the main idea and the framework of our proposed method Frac-PDE-Net. In Section 3, we apply Frac-PDE-Net to discover some biological PDE models based on simulation data. Then, in Section 4, we make some predictions to test the effectiveness of the models learned in Section 3. Finally, we summarize our findings and present some possible future works in Section 5.

2. Methodology

The main idea of the PDE-Net method, as described in [16], is to use a deep convolutional neural network (CNN) to study generic nonlinear evolution partial differential equations (PDEs) as shown below:

$$\partial_t u = F(\mathbf{z}, u, \nabla u, \nabla^2 u, \dots), \quad \mathbf{z} \in \Omega, \ t \in [0, T], \tag{3}$$

where $u = u(\mathbf{z}, t)$ is a function (scalar valued or vector valued) of the space variable $\mathbf{z}$ and the temporal variable t. Its architecture is a feed-forward network that combines the forward Euler method in time with the second-order finite difference method in space through the implementation of special filters in the CNN that imitate differential operators. The network is trained to approximate the solution to the above PDEs and then the network

is used to make predictions for the subsequent time steps. The authors of [16] show that this approach is effective for solving a range of PDEs and can achieve satisfactory accuracy and computational efficiency compared to traditional numerical methods. In this paper, we follow a similar framework to PDE-Net, but with modifications on a symbolic network framework ($SymNet_m^k$) to better align with biological models.

2.1. PDE-Net Review

The feed-forward network consists of several Δt-blocks, all of which use the same parameters optimized through minimizing a loss function. For simplicity, we will only show one Δt-block for two-dimensional PDEs, as repeating it generates multiple Δt-blocks, and the concept can easily be extended to higher-dimensional PDEs.

Denote the space variable $\mathbf{z}$ in (3) to be $\mathbf{z} = (x, y)$ since we are dealing with the two-dimensional case. Let $t_0 = 0$ and $\tilde{u}(\cdot, t_0)$ be the given initial data. For $i \geq 0$, $\tilde{u}(\cdot, t_{i+1})$ denotes the predicted value of u at time t_{i+1} calculated from the predicted (true) value of $\tilde{u}$ at time t_i using the following procedure:

$$\tilde{u}(\cdot, t_{i+1}) = \tilde{u}(\cdot, t_i) + (\Delta t)\,\mathrm{SymNet}(x, y, D_{00}u, D_{10}u, D_{01}u, D_{20}u, \dots),$$

where SymNet is an approximation operator of F. Here, the operators D_{ij} are convolution operators with the underlying filters q_{ij}, i.e., $D_{ij}u := \frac{1}{(\Delta x)^i(\Delta y)^j}\, q_{ij} \otimes u$. These operators approximate differential operators:

$$D_{ij}u \approx \frac{d^{i+j}u}{d^i x d^j y}.$$

For a general $N \times N$ filter $q = \big(q[k_1, k_2]\big)$, where $-\frac{N-1}{2} \leq k_1, k_2 \leq \frac{N-1}{2}$,

$$q \otimes u(x, y) := \sum_{k_1, k_2} q[k_1, k_2]\, u(x + k_1\Delta x, y + k_2\Delta y). \tag{4}$$

By Taylor expansion,

$$q \otimes u(x, y) = \sum_{i,j=0}^{N-1} m_{ij}(\Delta x)^i(\Delta y)^j \frac{\partial^{i+j}u}{\partial^i x \partial^j y}\bigg|_{(x,y)} + O\big(|\Delta x|^N\big) + O\big(|\Delta y|^N\big),$$

where

$$m_{ij} := \frac{1}{i!j!} \sum_{k_1, k_2} k_1^i k_2^j q[k_1, k_2], \quad \forall\, 0 \leq i, j \leq N - 1.$$

In particular, if choosing $\Delta x = \Delta y = \delta$, then

$$q \otimes u(x, y) = \sum_{i,j=0}^{N-1} m_{ij}\, \delta^{i+j} \frac{\partial^{i+j}u}{\partial^i x \partial^j y}\bigg|_{(x,y)} + O\big(\delta^N\big), \tag{5}$$

As a result, the training of q can be performed through the training of $M := (m_{ij})$ since the moment matrix $M = M(q)$. It is important to note that the trainable filters M (or q) must be carefully constrained to match differential operators.

For example, to approximate $\frac{\partial u}{\partial x}$ by $D_{10}u$, or equivalently by $\frac{1}{\Delta x}\, q_{10} \otimes u$ for a 3×3 filter q_{10}, we may choose

$$M_1(q_{10}) = \begin{pmatrix} 0 & 0 & * \\ 1 & * & * \\ * & * & * \end{pmatrix} \quad \text{or} \qquad M_2(q_{10}) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & * \\ 0 & * & * \end{pmatrix}, \tag{6}$$

where $*$ means no constraint on the corresponding entry. Generally, the fewer instances of $*$ present, the more restrictions are imposed, leading to increased accuracy. In this example

of (6), the choice of M_1 ensures the 1st order accuracy and the choice of M_2 guarantees the 2^{nd} order accuracy. More precisely, if we plug M_1 into (5) with $\Delta x = \Delta y = \delta$, then

$$q_{10} \otimes u(x,y) = \delta \frac{\partial u}{\partial x} + O(\delta^2),$$

which implies $\frac{1}{\Delta x} q_{10} \otimes u(x,y) = \frac{\partial u}{\partial x} + O(\Delta x)$. Similarly, if we plug M_2 into (5), then $\frac{1}{\Delta x} q_{10} \otimes u(x,y) = \frac{\partial u}{\partial x} + O((\Delta x)^2)$. In PDE-Net 2.0, all moment matrices are trained as subject to partial constraints so that the accuracy is at least 2nd order.

The $SymNet_m^k$ network, modeled after CNNs, is employed to approximate the multivariate nonlinear response function F. It takes a m-dimensional vector as input and consists of k layers. As depicted in Figure 2, the $SymNet_m^2$ network has two hidden layers, where each f_i unit performs a dyadic multiplication and the output is added to the $(i+1)$th hidden layer.

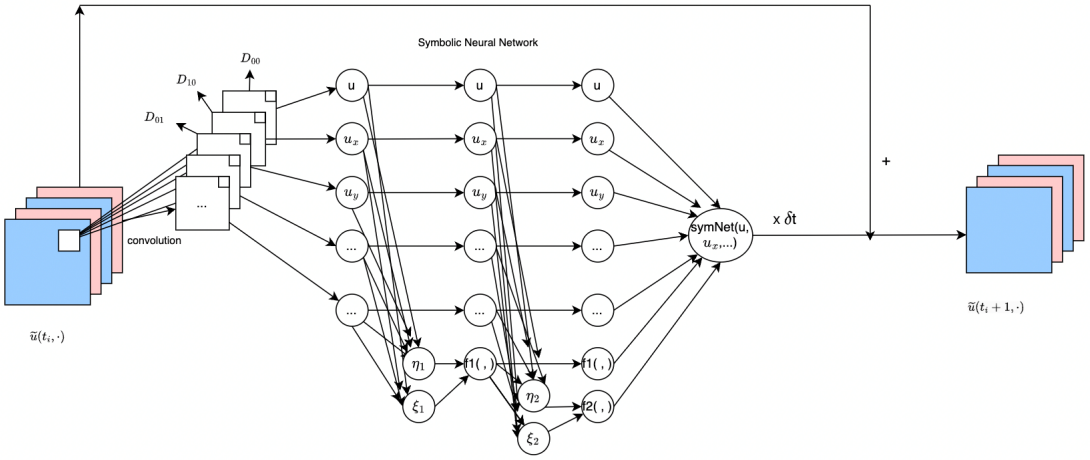


Figure 2. The scheme of one Δt.

The loss function for this method has three components and is defined as follows:

$$L = L^{data} + \lambda_M L^{moment} + \lambda_S L^{SymNet}. \tag{7}$$

Here, L^{data} measures the difference between the true data and the prediction. Consider the data set $\{u_j(\cdot, t_i) \in R^{N_s \times N_s} : 1 \le i \le n, 1 \le j \le N\}$, where n is the number of Δt blocks, N is the total number of samples, and N_s is the number of space grids. The index j indicates the jth solution path with a certain initial condition of the unknown dynamics, and the index i represents the solution at time t_i. Then, we define

$$L^{data} = \frac{1}{nN(\Delta t)^2} \sum_{i=1}^{n} \sum_{j=1}^{N} \ell_{ij}.$$

Here, $\ell_{ij} := ||u_j(t_i, \cdot) - \widetilde{u}_j(t_i, \cdot)||_2^2$, where u_j represents the real data and $\widetilde{u}_j$ denotes the predicted data. For a given threshold s, recall the Huber's loss function $\ell_1^{(s)}$ defined as

$$\ell_1^{(s)}(x) = \begin{cases} |x| - \frac{s}{2} & \text{if} \quad |x| > s, \\ \frac{x^2}{2s} & \text{if} \quad |x| \le s. \end{cases} \tag{8}$$

We then define the following:

$$L^{moment} = \sum_{i,j} \sum_{i_1, j_1} \ell_1^{(s)}(M(q_{ij})[i_1, j_1]),$$

where q_{ij}s are filters and $M(q_{ij})$ is the moment matrix of q_{ij}. Using the same Huber loss function as in (8), we define

$$L^{SymNet} = \sum_{i,j} \ell_1^{(s)}(w_{ij}),$$

where w_{ij}s are the parameters in SymNet. The coefficients λ_M and λ_S in Equation (7) serve as regularization terms to help control the magnitude of the parameters, preventing them from becoming too large and overfitting to the training data.

2.2. mPDE-Net (Modified PDE-Net)

In mPDE-Net, we do not include multiplications between derivatives of u and v, as these interactions are not commonly present in biological phenomena. Additionally, to handle interactions in fractional or integral forms, such as those in Equation (2), mPDE-Net incorporates integral terms and division operations into $SymNet_m^k$. However, there was a challenge with identifiability using mPDE-Net. For instance, consider a two-component input vector u and v. mPDE-Net may produce results such as $\frac{u^2}{u+\epsilon}$ or $\frac{uv}{v+\epsilon}$, where ϵ is a small number due to noise. Although both of these terms essentially represent the same term u, the mPDE-Net is unable to automatically identify them as such. Keeping all similar terms such as $\frac{u^2}{u+\epsilon}$, $\frac{uv}{v+\epsilon}$ and u at the same time would result in a complex model and the real fractional term would not be effectively trained.

To address the identifiability issue, restrictions were imposed on the nonlinear interaction term $N(u,v)$ by assuming that $N(u,v) = g(u)h(v)$, where either g or h is linear and the other one can contain a fractional term with the order of the denominator larger than that of the numerator. For instance, the terms $\frac{u^2}{u+\epsilon}$ and $\frac{uv}{v+\epsilon}$ are further decomposed as follows:

$$\frac{u^2}{u+\epsilon} = u - \epsilon + \frac{\epsilon^2}{u+\epsilon}, \qquad \frac{uv}{v+\epsilon} = u - u\,\frac{\epsilon}{v+\epsilon}.$$

As seen, the main part of the above two terms is u while the rest, such as ϵ, $\frac{\epsilon^2}{u+\epsilon}$ and $u\frac{\epsilon}{v+\epsilon}$, are considered as perturbations since ϵ is very small. This allows the mPDE-Net to identify and combine the main parts of terms, resulting in a compact model.

Figure 3 presents an example of a system involving the derivatives of u and v up to the second order. The symbolic neural network in this example has five hidden layers, referred to as $SymNet_{10}^5$. The operators f_i are multiplication functions, i.e., $f_i(\eta_i, \xi_i) = \eta_i\xi_i$, for $i = 1,4,5$; and f_j are division functions, i.e., $f_j(\eta_j, \xi_j) = \frac{\eta_j}{\xi_j}$, for $j = 2,3$. Additionally, a term u^α is included to incorporate fractional powers, such as the term R^α in (2). The algorithm corresponding to this example is outlined in Algorithm 1, where $L_1 = (u, u_x, u_{xx}, v, v_x, v_{xx}, u^2, v^2, u^\alpha, I)^T, L_2 = (L_1^T, f_1)^T, L_3 = (L_2^T, f_2)^T, L_4 = (L_3^T, f_3)^T$, $L_5 = (L_4^T, f_4)^T, L_6 = (L_5^T, f_5)^T$.

Algorithm 1 Scheme of mPDE-Net.

Input: $u, u_x, u_{xx}, v, v_x, v_{xx}, u^2, v^2, u^\alpha, I$, where I represents $\int u(x,t)\,dx$,
$(\eta_1, \xi_1)^T = W^{(1)}L_1 + b^{(1)}, \quad W^{(1)} \in \mathbb{R}^{2\times 10}, L_1 \in \mathbb{R}^{10}, b^{(1)} \in \mathbb{R}^2,$
$(\eta_2, \xi_2)^T = W^{(2)}L_2 + b^{(2)}, \quad W^{(2)} \in \mathbb{R}^{2\times 11}, L_2 \in \mathbb{R}^{11}, b^{(2)} \in \mathbb{R}^2,$
$(\eta_3, \xi_3)^T = W^{(3)}L_3 + b^{(3)}, \quad W^{(3)} \in \mathbb{R}^{2\times 12}, L_3 \in \mathbb{R}^{12}, b^{(3)} \in \mathbb{R}^2,$
$(\eta_4, \xi_4)^T = W^{(4)}L_4 + b^{(4)}, \quad W^{(4)} \in \mathbb{R}^{2\times 13}, L_4 \in \mathbb{R}^{13}, b^{(4)} \in \mathbb{R}^2,$
$(\eta_5, \xi_5)^T = W^{(5)}L_5 + b^{(5)}, \quad W^{(5)} \in \mathbb{R}^{2\times 14}, L_5 \in \mathbb{R}^{14}, b^{(5)} \in \mathbb{R}^2,$
Output: $F = W^{(6)}L_6 + b^{(6)}, \quad W^{(6)} \in \mathbb{R}^{1\times 15}, L_6 \in \mathbb{R}^{15}, b^{(6)} \in \mathbb{R}^1.$

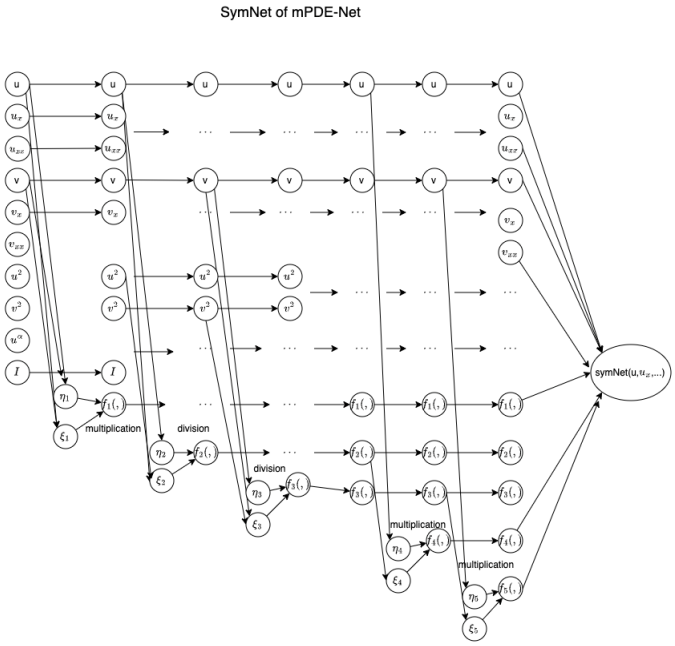


Figure 3. The scheme of mPDE-Net.

To further demonstrate the mPDE-Net approach, we present a concrete example. To simplify the notation, we introduce the row vector e_i with a 1 in the ith component and 0 in all other components, i.e.,

$$e_i = (0, 0, \cdots, 0, 1, 0 \cdots, 0),$$

where the number "1" is on the ith position. Then, we set

$$W^{(1)} = \begin{pmatrix} e_1 + e_4 \\ e_1 + e_4 \end{pmatrix}, \quad W^{(2)} = \begin{pmatrix} e_4 \\ 4e_4 + e_8 \end{pmatrix}, \quad W^{(3)} = \begin{pmatrix} 0.5e_1 \\ 0.2e_1 + e_7 \end{pmatrix},$$

$$W^{(4)} = \begin{pmatrix} 0.2e_1 \\ e_{12} \end{pmatrix}, \quad W^{(5)} = \begin{pmatrix} 0.2e_4 \\ e_{13} \end{pmatrix}, \quad W^{(6)} = 0.1e_1 + 0.3e_3 + 6e_4 + e_{11} + 2e_{14} + 3e_{15},$$

$$b^{(1)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad b^{(2)} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \quad b^{(3)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad b^{(4)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad b^{(5)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad b^{(6)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

According to Algorithm 1 for $1 \le i \le 5$,

$$W^{(1)}L_1 + b^{(1)} = \begin{pmatrix} u + v + 1 \\ u + v \end{pmatrix}, \quad f_1 = f_1(\eta_1, \xi_1) = \eta_1\xi_1 = (u + v + 1)(u + v),$$

$$W^{(2)}L_2 + b^{(2)} = \begin{pmatrix} v + 0.5 \\ 4v + v^2 + 0.5 \end{pmatrix}, \quad f_2 = f_2(\eta_2, \xi_2) = \frac{\eta_2}{\xi_2} = \frac{v + 0.5}{v^2 + 4v + 0.5},$$

$$W^{(3)}L_3 + b^{(3)} = \begin{pmatrix} 0.5u + 1 \\ 0.2u + u^2 \end{pmatrix}, \quad f_3 = f_3(\eta_3, \xi_3) = \frac{\eta_3}{\xi_3} = \frac{0.5u + 1}{u^2 + 0.2u},$$

$$W^{(4)}L_4 + b^{(4)} = \begin{pmatrix} 0.2u \\ f_2 \end{pmatrix}, \quad f_4 = f_4(\eta_4, \xi_4) = \eta_4\xi_4 = 0.2u\, f_2 = 0.2u\, \frac{v + 0.5}{v^2 + 4v + 0.5},$$

$$W^{(5)}L_5 + b^{(5)} = \begin{pmatrix} 0.2v \\ f_3 \end{pmatrix}, \quad f_5 = f_5(\eta_5, \xi_5) = \eta_5\xi_5 = 0.2v\, f_3 = 0.2v\, \frac{0.5u + 1}{u^2 + 0.2u},$$

Therefore,

$$\begin{aligned} SymNet_{10}^{5} &= W^{6}L_{6} + b^{(6)} = 0.1u + 0.3u_{xx} + 6v + f_1 + 2f_4 + 3f_5 \\ &= 0.3u_{xx} + u^2 + 2uv + v^2 + 1.1u + 7v + 0.4u\,\frac{v+0.5}{v^2+4v+0.5} + 0.6v\,\frac{0.5u+1}{u^2+0.2u}. \end{aligned}$$

Let $\mathcal{L}$ denote the library for PDE-Net 2.0 and $\mathcal{L}_f$ denote the library for mPDE-Net. It is clear that $\mathcal{L}$ and $\mathcal{L}_f$ are distinct. Typically, $\mathcal{L}$ only seeks to identify multiplication terms and has the form:

$$\mathcal{L} = \left\{\lambda(U_{xx} + U_{yy}) + f_1(U) :\ \lambda \in \mathbb{R},\, U = (u, v),\, f_1 \in \mathcal{P}\right\},$$

where

$$\mathcal{P} := \{\text{Polynomials of } U \text{ up to a certain degree}\}.$$

Conversely, $\mathcal{L}_f$ is engineered to learn both multiplication terms and fractional terms, subject to certain constraints. In our paper, we make the choice of

$$\begin{aligned} \mathcal{L}_f = \Big\{ &\lambda(U_{xx} + U_{yy}) + f_1(U) + \frac{f_2(u)}{f_3(u)} f_4(v) + f_5(u)\frac{f_6(v)}{f_7(v)} : \\ &\lambda \in \mathbb{R},\, U = (u, v),\, \{f_i\}_{i=1}^{7} \subset \mathcal{P},\, \deg f_2 < \deg f_3,\, \deg f_6 < \deg f_7 \Big\}, \end{aligned}$$

which is much larger than $\mathcal{L}$. Therefore, our framework of neural networks, built upon $\mathcal{L}_f$, is more challenging to implement than the original framework, which is based on $\mathcal{L}$.

2.3. Optimizing Hyperparameters

In this section, we will explain the process of tuning hyperparameters λ_M and λ_S in the loss function (7). Firstly, the range of spatial and temporal variables in the training set are defined as $[-L, L]$ and $[0, T]$, respectively. Then, using the finite difference method, we generate a dataset that acts as the "true data". Additionally, we consider M initial conditions. The time interval is determined by $dt/\tilde{dt}$, where $\tilde{dt}$ is the time step size for computing the "true data" and dt represents the time step size for selecting the "observational data". Typically, $\tilde{dt}$ is chosen to be much smaller than dt. The solution corresponding to the mth initial condition is denoted as $u_m(\cdot, \cdot)$, where the first "$\cdot$" refers to the spatial variable and the second "$\cdot$" represents the temporal variable. If the solution is evaluated at the kth time step, it is written as $u_m(\cdot, t_k)$, with "$\cdot$" representing the spatial variable.

The M initial values from M initial conditions are divided into three separate groups, resulting in $M = M_1 + M_2 + M_3$, where M_1, M_2, and M_3 represent the sizes of the training set, validation set, and test set, respectively. The solutions produced by these initial values are designated as follows:

Training set: $u_1(\cdot, \cdot), \cdots, u_{M_1}(\cdot, \cdot)$;

Validation set: $u_{M_1+1}(\cdot, \cdot), \cdots, u_{M_1+M_2}(\cdot, \cdot)$;

Testing set: $u_{M_1+M_2+1}(\cdot, \cdot), \cdots, u_{M_1+M_2+M_3}(\cdot, \cdot)$.

We use the training set to train our models, the validation set to find the best parameters, and the testing set to evaluate the performance of the trained models.

Assume we divide the time range $[0, T]$ into K blocks, with cutting points denoted as t_k for $1 \le k \le K$. Then, for any $1 \le m \le M$ and for any $1 \le k \le K$, we define

$$\ell_{km} = ||u_m(\cdot, t_k) - \tilde{u}_m(\cdot, t_k)||_2^2,$$

where $\|\cdot\|_2$ denotes the L^2 norm with respect to the space variable on $[-L, L]$, u_m is the "true solution", and $\tilde{u}_m$ is the "predicted solution" by a neural network. Based on this, the training loss, validation loss and the testing loss are defined as follows:

- Training loss:

$$L_{train} := \frac{1}{M_1 K (dt)^2} \sum_{k=1}^{K} \sum_{m=1}^{M_1} \ell_{km}. \tag{9}$$

- Validation loss:

$$L_{valid} := \frac{1}{M_2 K (dt)^2} \sum_{k=1}^{K} \sum_{m=M_1+1}^{M_1+M_2} \ell_{km}. \tag{10}$$

- Testing loss:

$$L_{test} := \frac{1}{M_3 K (dt)^2} \sum_{k=1}^{K} \sum_{m=M_1+M_2+1}^{M} \ell_{km}. \tag{11}$$

We choose the hyperparameters λ_M and λ_S in the loss function (7) using the validation sets. Let $B_{mk} = u_m(\cdot, t_k)$ and $B_{jk} = u_j(\cdot, t_k)$, where $1 \le m \le M_1$, $M_1 + 1 \le j \le M_1 + M_2$ and $1 \le k \le K$. We define the training number by N_t. We then gradually increase the time points of the training and validation sets. For instance, if K = 15 and $N_t = 5$, the training and validation sets can be selected as follows. The performance metric is the same as the validation loss in (10).

Training	Validation	Validation Loss
$B_{m1}, \cdots, B_{m3}$	$B_{j1}, \cdots, B_{j3}$	$L_{valid}^{(1)}$
$B_{m1}, \cdots, B_{m6}$	$B_{j1}, \cdots, B_{j6}$	$L_{valid}^{(2)}$
$B_{m1}, \cdots, B_{m9}$	$B_{j1}, \cdots, B_{j9}$	$L_{valid}^{(3)}$
$B_{m1}, \cdots, B_{m12}$	$B_{j1}, \cdots, B_{j12}$	$L_{valid}^{(4)}$
$B_{m1}, \cdots, B_{m15}$	$\tilde{B}_{j1}, \cdots, B_{j15}$	$L_{valid}^{(5)}$

Furthermore, we tune the hyperparameters using Hyperopt [29], which uses Bayesian optimization to explore the hyperparameter space more efficiently than a brute-force grid search. Specifically, the mPDE-Net is nested in the objective function of Hyperopt, which will optimize the average validation loss L_{avl} of models.

$$L_{avl} = \frac{1}{5} \sum_{i=1}^{5} L_{valid}^{(i)}.$$

The selection procedure is described in Algorithm 2.

Algorithm 2 Optimizing Hyperparameters using Hyperopt

1: Initialize the search spaces for λ_M and λ_S;
2: Define the objective function (to be optimized) as the average testing loss obtained from mPDE-Net, implemented using PyTorch;
3: Set the optimization algorithm, specify the number of trials, and initialize the results list.
4: **for** $i = 1$ to number of trials **do**
5: Sample a set of hyperparameters from the search spaces, evaluate the objective function with the sampled hyperparameters, and set a list called the Validation loss.
6: **for** $r = 1$ to $\frac{K}{N_t}$ **do**
7: Train model of mPDE-Net on $B_{m1}, \cdots, B_{mr}$, test it on $B_{j1}, \cdots, B_{jr}$ to get a validation loss, and then append the validation loss to the Validation loss.
8: **end for**
9: Get an average validation loss from the Validation loss, append the hyperparameters and the average validation loss to the results list, and then update the search space based on the results so far.
10: **end for**
11: **return** the hyperparameters with the minimum objective function value.

2.4. Frac-PDE-Net

We have noted that mPDE-Net accurately fits data and recovers terms, but it may not always simplify the learned PDE, making it challenging to interpret. To address this, we implement sparsity-encouraging methods such as the Lasso approach. However, even with Lasso and hyperparameters chosen from the validation sets, the predicted equation still had redundant terms. This is likely due to correlated data and linear dependencies in the data, which prevent Lasso from fully shrinking the extra coefficients to zero. To overcome this, we employ two approaches. The first, called the L^2 norm-based term selection criterion, weakens or eliminates linear dependencies in the data. The second, called sequential threshold ridge regression (STRidge), creates concise models through strong thresholding. We will discuss these approaches in more detail below.

L^2 norm based term selection criterion. Consider the underlying PDE in the form of

$$\partial_t u = \Theta(u)\xi, \tag{12}$$

where

$$\Theta(u) = \big(\Theta_1(u), \Theta_2(u), \ldots, \Theta_p(u)\big), \quad \xi = (\xi_1, \xi_2, \ldots, \xi_p)^T.$$

To address the issue of excessive terms in the learned PDE, we apply the L^2 norm based term selection criterion. This involves normalizing the columns of $\Theta(u)$ to obtain $\Phi_k(u)$

$$\Theta(u)\xi = \sum_{k=1}^{p} \Theta_k(u)\xi_k = \sum_{k=1}^{p} \Phi_k(u)\eta_k,$$

where

$$\Phi_k(u) = \frac{\Theta_k(u)}{\|\Theta_k(u)\|_2}, \quad \eta_k = \xi_k \|\Theta_k(u)\|_2, \quad \forall\, 1 \le k \le p,$$

and adjusting the coefficients ξ to $\tilde{\xi}$,

$$\tilde{\xi}_k = \begin{cases} 0, & \text{if} \quad |\eta_k| < \delta \max_j |\eta_j|, \\ \xi_k, & \text{otherwise}, \end{cases} \qquad \forall\, 1 \le k \le p.$$

By removing the terms in $\Theta(u)$ whose adjusted coefficients η_k are significantly smaller than the largest one, we shorten the vector $\tilde{\xi}$ to $\xi^{(s)}$. The corresponding columns in $\Theta(u)$ form a new matrix $\Theta^{(s)}(u)$ with reduced linear dependency between its columns. This results in a simplified approximation of the PDE:

$$\partial_t u \approx \Theta^{(s)}(u)\, \xi^{(s)}. \tag{13}$$

Sparse regression: STRidge. After using the L^2 norm-based term selection criterion to select terms, as discussed previously, we move on to consider sparse regression to further improve the compactness of the representation for the hidden PDE model (13). Here, a tolerance threshold "tol" is introduced to select coefficients for sparse results. Coefficients smaller than "tol" will be discarded, and the remaining ones will be utilized until the number of terms stabilizes. The sparsity regression process is outlined in Algorithm 3. For further information, see [25].

To summarize, the mPDE-Net approach allows us to achieve relatively accurate predictions for the function and its derivatives. We then employ an L^2 norm-based term selection criterion and sparse regression to obtain a concise model, which we refer to as Frac-PDE-Net. Algorithm 4 summarizes this procedure.

Algorithm 3 : STRidge($\Theta^{(s)}$, U_t, λ, tol, iters)

$\hat{\xi}^{(s)} = argmin_{\tilde{\xi}^{(s)}} ||\Theta^{(s)}\xi^{(s)} - U_t||_2^2 + \lambda||\xi^{(s)}||_2^2$ ▷ ridge regression

bigcoeffs=$\{j : |\hat{\xi}_j^{(s)}| \geq tol\}$ ▷ select large coefficients

$\hat{\xi}^{(s)}[\sim \text{bigcoeffs}] = 0$ ▷ apply hard threshold

$\hat{\xi}^{(s)}[\text{bigcoeffs}] = STRidge(\Theta^{(s)}[:, \text{bigcoeffs}], U_t, \lambda, \text{tol}, \text{iters-1})$ ▷ recursive call with fewer coefficients

return $\hat{\xi}^{(s)}$

Algorithm 4 :L^2 Norm selection criterion+ STRidge($\hat{\Theta}$, $\hat{u}_t$, λ, tol, iters)

$\hat{\Theta}\hat{\xi} = \sum_{k=1}^{p} \hat{\Theta}_k\hat{\xi}_k = \sum_{k=1}^{p} \frac{\Theta_k(\hat{u})}{\|\Theta_k(\hat{u})\|_2}\left(\hat{\xi}_k\|\Theta_k(\hat{u})\|_2\right) = \sum_{k=1}^{p} \Phi_k(\hat{u})\eta_k$ ▷ Adjusted coefficients

bigcoeffs=$\{k : |\eta_k| \geq \delta \max_j |\eta_j|\}$ ▷ Select large coefficients

$\hat{\xi}[\sim \text{bigcoeffs}] = 0$

$\Theta^{(s)} = \hat{\Theta}[:, \text{bigcoeffs}]$ and $\xi^{(s)} = \hat{\xi}[\text{bigcoeffs}]$

$\hat{\xi}^{(s)} = argmin_{\tilde{\xi}^{(s)}} ||\Theta^{(s)}\xi^{(s)} - \hat{u}_t||_2^2 + \lambda||\xi^{(s)}||_2^2$ ▷ ridge regression

bigcoeffs=$\{j : |\hat{\xi}_j^{(s)}| \geq tol\}$ ▷ select large coefficients

$\hat{\xi}^{(s)}[\sim \text{bigcoeffs}] = 0$ ▷ apply hard threshold

$\hat{\xi}^{(s)}[\text{bigcoeffs}] = STRidge(\Theta^{(s)}[:, \text{bigcoeffs}], \hat{u}_t, \lambda, \text{tol}, \text{iters-1})$ ▷ recursive call with fewer non-zero coefficients

return $\hat{\xi}^{(s)}$

2.5. Kolmogorov-Smirnov Test

After applying the Frac-PDE-Net procedure, a simplified, interpretable model has been created. Our next goal is to determine if this model can be further compressed. We designate Model 1 as the system learned by Frac-PDE-Net, and Model 2 as the system obtained by removing the interaction term with the smallest L^2 norm from Model 1. To determine if Model 1 and Model 2 come from the same distribution, we use the Kolmogorov–Smirnov test (K-S test).

Since our examples involve systems of two PDEs, a two-dimensional K-S test is appropriate. The time range is $[0, T]$ with time step size dt, giving $n := \frac{T}{dt}$ time grids denoted as $\{t_i\}_{i=1}^n$, where $t_i = i(dt)$, and $1 \leq i \leq n$. At a fixed time t_i, we aim to test the proximity of two samples Y_{t_i} and $\widetilde{Y}_{t_i}$, which are associated with Model 1 and Model 2, respectively, at time t_i. For each t_i, we specify:

Hypothesis 1 (Null). *The two sets* $\{Y_{t_i}\}_{i=1}^n$ *and* $\{\widetilde{Y}_{t_i}\}_{i=1}^n$ *come from a common distribution.*

Hypothesis 2 (Alternative). *The two sets* $\{Y_{t_i}\}_{i=1}^n$ *and* $\{\widetilde{Y}_{t_i}\}_{i=1}^n$ *do not come from a common distribution.*

Let $H_{t_i,0}$ and $\hat{p}_{t_i}$ denote null hypotheses and the corresponding p-values, respectively, for $1 \leq i \leq n$. In this paper, we employed Bonerroni [30], Holm [31] and Benjamini–Hochberg (B-H) [32] methods for multiple testing adjustment. Note that the Bonferroni method is the most conservative one among these three methods. Under the complete null hypothesis of a common distribution across all time points, no more than 5% of the total time points can be rejected.

3. Numerical Studies: Convection-Diffusion Equations with the Neumman Boundary Condition

In this section, we showcase numerical examples to demonstrate the efficacy of Frac-PDE-Net, our proposed method. The training, validation, and testing data are generated

based on the underlying governing equation. Our aim is to use Frac-PDE-Net on these data to obtain a concise and interpretable model for the PDE. The governing PDEs under consideration in this paper are of the following form:

$$\begin{cases} \partial_t u = F_1(u,v), \\ \partial_t v = F_2(u,v), \end{cases} \tag{14}$$

where

$$F_1(u,v) = d_1\Delta u + P_1(u,v) + R_1(u,v), \quad F_2(u,v) = d_2\Delta v + P_2(u,v) + R_2(u,v). \tag{15}$$

Here, d_1 and d_2 are positive diffusion coefficients, R_1 and R_2 represent fractional functions of (u,v), and P_1 and P_2 denote combinations of power functions and integration operators of (u,v) through addition and multiplication. For example, $R_1(u,v)$ can be $\frac{u-2}{v^2-v+3}$, and $P_1(u,v)$ can be $1 + u^{1.5} - v^2 + u^{1.5}\int u\,dx$.

3.1. Example 1: A 2-Dimensional Model

Our first example is taken from (Equation (2.8) in Section 2.2 in [2]). In this example, we consider (14) under the Neumann boundary condition on a 2-dimensional domain $D_1 := [-5,5] \times [-5,5]$ with $d_1 = 0.3$, $d_2 = 0.4$, $P_1(u,v) = 1 - u$, $P_2(u,v) = 0.4 - 0.2v$, and

$$R_1(u,v) = R_2(u,v) = -2\,\frac{uv}{1+u+4u^2} = -\frac{1}{2}\,\frac{uv}{u^2+0.25u+0.25}.$$

Thus, Equation (14) is reduced to

$$\begin{cases} \partial_t u = F_1(u,v), \\ \partial_t v = F_2(u,v), \\ \partial_x u(-5,y,t) = \partial_x u(5,y,t) = \partial_y u(x,-5,t) = \partial_y u(x,5,t) = 0, \end{cases} \tag{16}$$

with $(x,y,t) \in [-5,5] \times [-5,5] \times [0,0.15]$ and

$$\begin{cases} F_1(u,v) &= 0.3(\partial_x^2 u + \partial_y^2 u) + 1 - u - \dfrac{1}{2}\,\dfrac{uv}{u^2+0.25u+0.25}, \\ F_2(u,v) &= 0.4(\partial_x^2 v + \partial_y^2 v) + 0.4 - 0.2v - \dfrac{1}{2}\,\dfrac{uv}{u^2+0.25u+0.25} \end{cases} \tag{17}$$

The observations are generated with Equations (16) and (17), and then split into training data, validation data and testing data. The PDE is solved by applying a finite difference scheme to a 64×64 spatial mesh grid with the central difference scheme for $\Delta := \partial_x^2 + \partial_y^2$, and with a temporal discretization of second-order Runge–Kutta (see [16]), using a time step size of $\frac{1}{1600}$.

In addition, the observations are obtained from various initial values: this implies an extra variability in the datasets, that is necessary if we want to be able to generalize well to any initial conditions. We assume that we have $N_{Init} = 12$ different solutions, coming from different initial values w_0. The functions are random, defined through random parameters $a_{i,j}$, $b_{i,j}$, $c_{i,j}$, $d_{i,j}$, $a_{k,l}$, $b_{k,l}$, $c_{k,l}$ and $d_{k,l}$, which follow from the standard normal distribution $\mathcal{N}(0,1)$, c_1 and c_2, which follow from uniform distributions: $c_1 \sim \mathcal{U}(-0.5,0.5)$ and $c_2 \sim \mathcal{U}(0.5,1.5)$. Then, we generate the 12 initial values (u_0, v_0) by setting

$$u_0(x,y) = \frac{w_0(x,y)}{\max|w_0|} + c_1, \qquad v_0(x,y) = \frac{\widetilde{w}_0(x,y)}{\max|\widetilde{w}_0|} + c_2, \tag{18}$$

where

$$w_0(x,y) = \sum_{|i|,|j|\le 13} \Big\{ a_{i,j}\cos(2ix)\cos(2jy) + b_{i,j}\sin[(2i+1)x]\sin[(2j+1)y] + c_{i,j}\sin[(2i+1)x]\cos(2jy) + d_{i,j}\cos(2ix)\sin[(2j+1)y] \Big\},$$

$$\widetilde{w}_0(x,y) = \sum_{|k|,|l|\le 13} \Big\{ a_{k,l}\cos(2kx)\cos(2ly) + b_{k,l}\sin[(2k+1)x]\sin[(2l+1)y] + c_{k,l}\sin[(2k+1)x]\cos(2ly) + d_{k,l}\cos(2kx)\sin[(2l+1)y] \Big\}.$$

For any given initial data (u_0, v_0), we denote the corresponding solution to be (u^*, v^*). When noise is allowed, we assume the perturbed data to be

$$u(x,y,t) = u^*(x,y,t) + n_l Q_1, \quad v(x,y,t) = v^*(x,y,t) + n_l Q_2,$$

where n_l is the level of Gaussian noise added, and Q_1 and Q_2 are random variables, which follow from the normal distribution: $Q_i \sim \mathcal{N}(0, \sigma_i^2)$ for $i = 1, 2$, where σ_1 (or σ_2 resp.) is the standard deviation of the true data u^* (or v^* resp.).

Since the time is from 0 to 0.15, there are 15 time blocks and we denote $N_{Time} = 15$. For spatial variables, we have $N_{Space} = 64$, where N_{Space} represents the number of space grids. Therefore, the dataset is

$$\{(u_{t,k}, v_{t,k}) : 1 \le t \le N_{Time},\ 1 \le k \le N_{Init}\},$$

where both $u_{t,k}$ and $v_{t,k}$ are matrices in $\mathbb{R}^{N_{Space} \times N_{Space}}$. The following Tables 1 and 2 show a summary of parameters for Frac-PDE-Net.

Table 1. Fixed parameters for Frac-PDE-Net.

Parameter	Value
t	[0, 0.15]
dt	0.01
x & y	$[-5, 5]$
dx & dy	$\frac{10}{64}$
N_{Init}	12
N_{Time}	15
N_{Space}	64

Table 2. Hyper-parameters selected by validation procedure Section 2.3 for Frac-PDE-Net.

Parameter	Value
λ_M (5% noise level)	3.28×10^{-5}
λ_S (5% noise level)	4.93×10^{-5}

Our goal is to discover the terms $F_1(u,v)$ and $F_2(u,v)$ on the right hand side of (16) and the true expressions are given by (17). For convenience of notation, we denote $\widehat{F}_1$ and $\widehat{F}_2$ to be our predicted operators for F_1 and F_2. Based on some existing models (see, e.g., Section 2.2 in [2]), we adopt some assumptions before discovering $\widehat{F}_1$ and $\widehat{F}_2$. More precisely, we assume that

$$\widehat{F}_1(u,v) = \widehat{d}_1\Delta u + \widehat{P}_1(u,v) + \widehat{R}_1(u,v), \quad \widehat{F}_2(u,v) = \widehat{d}_2\Delta v + \widehat{P}_2(u,v) + \widehat{R}_2(u,v), \tag{19}$$

where $\widehat{d}_1$ and $\widehat{d}_2$ are positive constants, $\widehat{P}_1$ and $\widehat{P}_2$ are polynomials of (u,v) up to order 2, and both the fractional terms $\widehat{R}_1$ and $\widehat{R}_2$ are in the form $l(u)r(v)$ or $r(u)l(v)$, where l means

a linear function and r denotes a fractional function in which the numerator is linear and the denominator is quadratic.

Based on these assumptions, we consider the following library $\{u,\ u_{xx},\ u_{yy},\ v,\ v_{xx},\ v_{yy}\}$ for training our model.

The filters q (as defined in (4)) are selected to be of size 5×5. The total number of parameters in $W^{(i)}$ (as defined in Algorithm 1) for approximating F_1 and F_2 is 56, and the number of trainable parameters in the moment matrices M (as defined in (6)) is 52. To optimize the parameters, we use the BFGS algorithm instead of the Adam or SGD optimizers since the BFGS algorithm is faster and also stable.

In the following, we outline the notation used and summarize the key steps of our framework.

1. $\widehat{F}_i^{mPDE-Net}$ denotes the result of applying the modified PDE-Net on our model.
2. Next, we utilize the L^2 norm-based selection criterion and sparse regression on $\widehat{F}_i^{mPDE-Net}$ to obtain a more concise and interpretable model, referred to as $\widehat{F}_i^{smPDE-Net}$. The "s" in $\widehat{F}_i^{smPDE-Net}$ represents the application of sparse regression.
3. Subsequently, we fix the terms in $\widehat{F}_i^{smPDE-Net}$ and retrain its coefficients to produce a final model named $\widehat{F}_i^{rsmPDE-Net}$. This is the end result of our Frac-PDE-Net scheme. The "r" in $\widehat{F}_i^{rsmPDE-Net}$ signifies the process of retraining the coefficients.
4. Finally, to verify that no further terms can be eliminated after Frac-PDE-Net, we compare two models: Model 1, generated by Frac-PDE-Net; and Model 2, which is identical to Model 1 but removes the term with the smallest L^2 norm from $\widehat{F}_1$ and $\widehat{F}_2$. The coefficients in Model 2 are retrained, and the resulting model is referred to as $\widehat{F}_i^{PHrsmPDE-Net}$. "PH" in $\widehat{F}_i^{PHrsmPDE-Net}$ represents the Post-hoc selection in Model 2. The comparison between Model 1 and Model 2 will be conducted using the Kolmogorov–Smirnov test as outlined in Section 2.5.

For this case, we added 5% noise to the generated data to form the observational data. The results are displayed in Table 3. Table 3 shows that $\widehat{F}_i^{mPDE-Net}$ (modified PDE-Net framework) accurately identifies the terms in Example 1 and estimates their corresponding coefficients. However, it also produces unnecessary terms with low weights after training. By applying the L^2 norm-based selection and sparse regression (L^2+SP), we successfully remove these extra terms in $\widehat{F}_i^{rsmPDE-Net}$. After the terms in $\widehat{F}_1$ and $\widehat{F}_2$ are identified, we retrain the model with these fixed terms to obtain the final coefficients in $\widehat{F}_i^{rsmPDE-Net}$.

Table 3. PDE model discovery with 5% noise.

True F_1^*	$0.3\Delta u + 1 - u - 0.5\,\frac{uv}{u^2+0.25u+0.25}$
$\widehat{F}_1^{mPDE-Net}$	$0.305\Delta u + 0.992 - 0.988u - 0.510\,\frac{uv}{u^2+0.259u+0.265}$ $-0.003\,\frac{v}{u^2+0.259u+0.265} + 0.003v$
$\widehat{F}_1^{smPDE-Net}$	$0.305\Delta u + 1.00 - 0.981u - 0.532\,\frac{uv}{u^2+0.259u+0.265}$
$\widehat{F}_1^{rsmPDE-Net}$ (Frac-PDE-Net)	$0.304\Delta u + 0.975 - 0.982u - 0.501\,\frac{uv}{u^2+0.256u+0.260}$
$\widehat{F}_1^{PHrsmPDE-Net}$	$0.278\Delta u + 0.969 - 0.993u - 0.514\,\frac{uv}{u^2+0.301u+0.271}$
True F_2^*	$0.4\Delta v + 0.4 - 0.2v - 0.5\,\frac{uv}{u^2+0.25u+0.25}$
$\widehat{F}_2^{mPDE-Net}$	$0.398\Delta v + 0.412 - 0.195v - 0.510\,\frac{uv}{u^2+0.254u+0.263}$ $-0.005\,\frac{v}{u^2+0.254u+0.263} - 0.010u$
$\widehat{F}_2^{smPDE-Net}$	$0.398\Delta v + 0.424 - 0.199v - 0.542\,\frac{uv}{u^2+0.254u+0.263}$
$\widehat{F}_2^{rsmPDE-Net}$ (Frac-PDE-Net)	$0.400\Delta v + 0.385 - 0.202v - 0.490\,\frac{uv}{u^2+0.243u+0.256}$
$\widehat{F}_2^{PHrsmPDE-Net}$	$0.344\Delta v + 2.116 - 0.815v$

To test whether Model 1 ($\widehat{F}_i^{rsmPDE-Net}$) and Model 2 ($\widehat{F}_i^{PHrsmPDE-Net}$) are similar or not, we compare their predictions by using the finite difference scheme. Consider the time range to be $[0, 0.5]$ with time step size $dt = 0.01$. Hence, there are 50 time grids, which are denoted to be $\{t_i\}_{i=1}^{50}$, where $t_i = 0.01i$, $1 \le i \le 50$. Fix a time t_i, we introduce the residuals $E_{t_i} := Y_{t_i} - Y_{t_i}^*$ and $\widetilde{E}_{t_i} := \widetilde{Y}_{t_i} - Y_{t_i}^*$, where $Y_{t_i}^*$ represents the true solution, and Y_{t_i} and $\widetilde{Y}_{t_i}$ denote the predicted solutions based on Model 1 and Model 2, respectively, at time t_i. We will test if the residuals $\{E_{t_i}\}_{i=1}^{50}$ and $\{\widetilde{E}_{t_i}\}_{i=1}^{50}$ have similar distributions. The null hypothesis is $H_0^{(i)} : E_{t_i} \sim \widetilde{E}_{t_i}$ and the alternative hypothesis is $H_A^{(i)} : E_{t_i} \not\sim \widetilde{E}_{t_i}$. Applying Bonferroni method, Holm method and the B-H's procedure for multiple testing adjustment, discussed in Section 2.5, the test results are presented in the following Table 4.

Table 4. Hypothesis tests with 5% observation noise.

$H_0^{(i)}$ vs. $H_A^{(i)}, 1 \le i \le 50$	Number of Rejections
Bonferroni	49
Holm	49
B-H	49

The results in Table 4 show that Model 1 (Frac-PDE-Net) is significantly different from Model 2, meaning all terms in Model 1 should be kept. Hence, the final discovered terms for $\widehat{F}_1$ and $\widehat{F}_2$ are represented by Model 1 (Frac-PDE-Net) in Table 4.

To assess the stability of the results shown above, we repeated the experiments 100 times and the results are presented in Figures 4 and 5. The process of merging similar terms is outline in Appendix A.1. The plots show that there are some instances where the three methods fail to eliminate certain redundant terms. However, these instances are rare, as the median of these terms is 0, indicating that they appear infrequently.

3.2. Example 2: A 1-Dimensional Model

Our second example is taken from [6]. In this example, we consider (14) under the Neumann boundary condition on a one-dimensional domain $D_1 := [-\frac{5\pi}{2}, \frac{5\pi}{2}]$ with $d_1 = 0.1, d_2 = 10$,

$$P_1(u,v) = 3.6u^{1.5} - 3.6u - 0.229u^{1.5}\int_{-2.5\pi}^{2.5\pi} u\,dx, \quad P_2(u,v) = u - 0.4v,$$

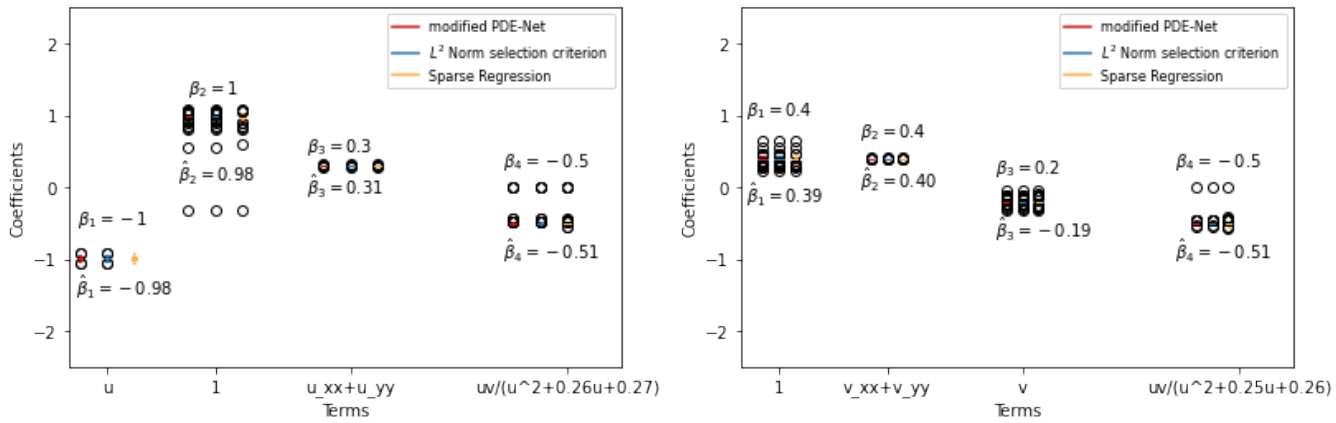
$$R_1(u,v) = 0.081\,\frac{u}{v^2 + 0.0215}, \quad R_2(u,v) = 0.$$

Thus, Equation (14) is reduced to

$$\begin{cases} \partial_t u = F_1(u,v), \\ \partial_t v = F_2(u,v), \\ \partial_x u(-2.5\pi, t) = \partial_x u(2.5\pi, t) = \partial_x v(-2.5\pi, t) = \partial_x v(2.5\pi, t) = 0, \end{cases} \tag{20}$$

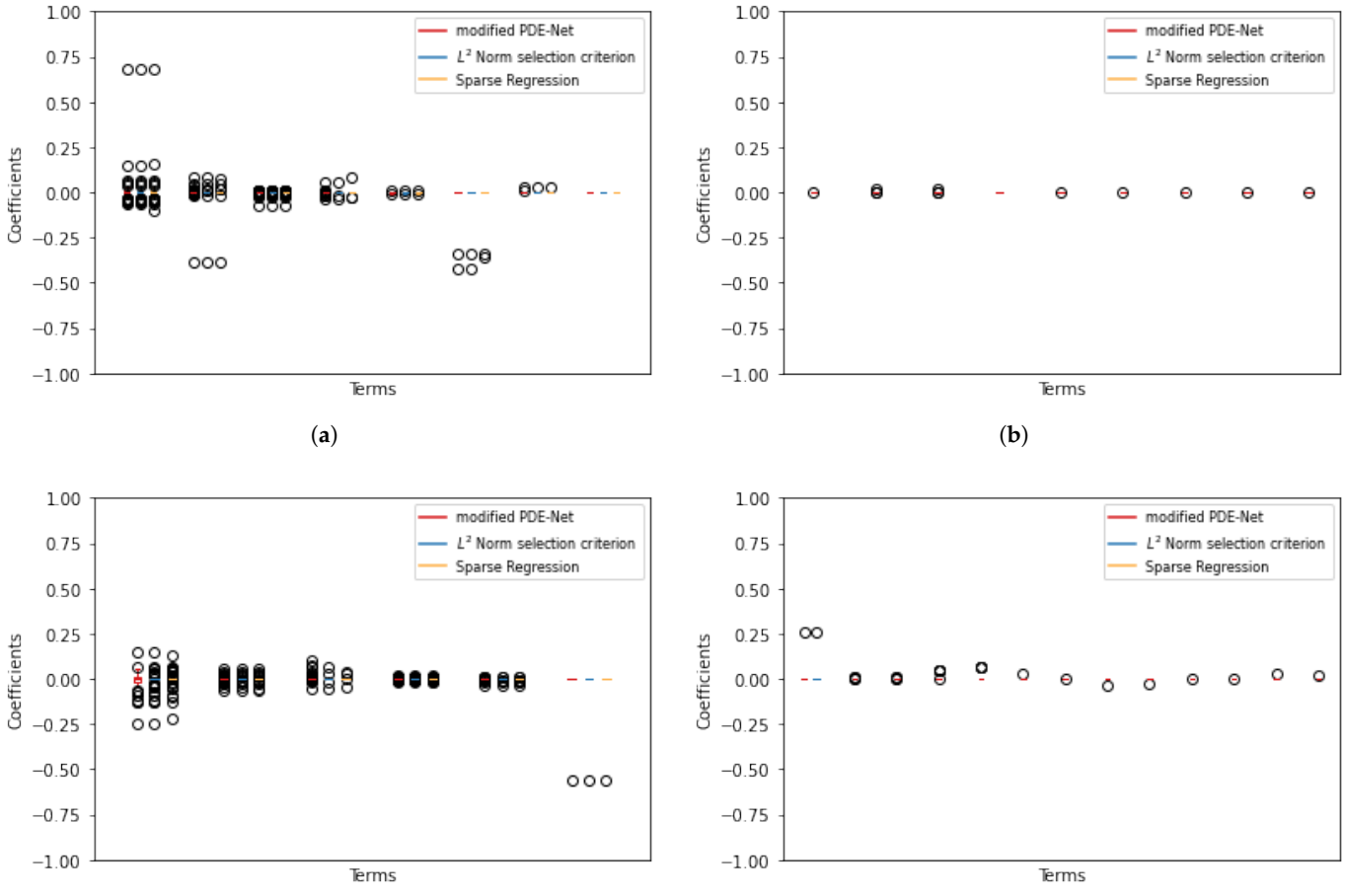
with $(x,t) \in [-5,5] \times [0, 0.75]$ and

$$\begin{cases} F_1(u,v) &= 0.1\partial_x^2 u + 3.6u^{1.5} - 3.6u - 0.229u^{1.5}\int\limits_{-2.5\pi}^{2.5\pi} u\,dx + 0.081\,\frac{u}{v^2+0.0215}, \\ F_2(u,v) &= 10\partial_x^2 v + u - 0.4v. \end{cases} \tag{21}$$



(**a**) (**b**)

Figure 4. Simulation results for true positive discovering with 5% noise. (**a**) $\widehat{F}_1$. (**b**) $\widehat{F}_2$.



(**a**) (**b**)

(**c**) (**d**)

Figure 5. Simulation results for false positive discovering with 5% noise. (**a**) $\widehat{F}_1$. (**b**) $\widehat{F}_1$. (**c**) $\widehat{F}_2$. (**d**) $\widehat{F}_2$.

The training data, validation data and testing data are generated, based on (20), by applying a finite difference scheme to a 600 spatial mesh grid and then restricted to a 200 spatial mesh grid with the central difference scheme for $\Delta := \partial_x^2$, and with a temporal discretization of the implicit Euler scheme, using a time step size of 0.01. Furthermore, we evaluate 14 different initial values, 10 of which were selected from a set of solutions with periodic patterns. The remaining initial values were generated by combining elementary

functions. The reason for using different ways to produce initial values is to test if this method still works for periodical solutions.

We also add noise to the generated data in the following form:

$$u(x,y,t) = |u^*(x,y,t) + n_l Q_1|, \quad v(x,y,t) = |v^*(x,y,t) + n_l Q_2|$$

where n_l is the level of Gaussian noise added and Q_1 and Q_2 are random variables that follow from the normal distribution: $Q_i \sim \mathcal{N}(0, \sigma_i^2)$ for $i = 1, 2$, where σ_1 (or σ_2 resp.) is the standard deviation of u^* (or v^* resp.). The reason of imposing the absolute value sign is to avoid negative values, which may cause trouble to evaluate power functions with non-integer power, such as $u^{1.5}$.

We choose 15 blocks for the time on the interval $[0, 0.75]$ and denote $N_{Time} = 15$. For spatial variables, we set $N_{Space} = 200$, where N_{Space} represents the number of space grids. Therefore, the dataset is

$$\{(u_{t,k}, v_{t,k}) : 1 \le t \le N_{Time},\ 1 \le k \le N_{Init}\},$$

where both $u_{t,k}$ and $v_{t,k}$ are matrices in $\mathbb{R}^{N_{Space} \times N_{Space}}$. The following Tables 5 and 6 show a summary of parameters for Frac-PDE-Net.

In [6], some assumptions are made on the model based on existing experimental knowledge of the biological behavior. For example, it is assumed that the operator $F_2(u, v)$ is linear in both u and v, while $F_1(u, v)$ is nonlinear in both u and v. As the form in (15),

$$F_1(u, v) = d_1 \Delta u + P_1(u, v) + R_1(u, v).$$

In [6], the nonlinear dependence of $P_1(u, v)$ on u is via the combination of the power function u^α and the integration operator $\int u\, dx$, where α is further restricted to the range $[1, 2]$. On the other hand, $R_1(u, v)$ is assumed to be linear in u, but nonlinear in v and the nonlinear dependence on v is via a fractional function whose denominator is a quadratic polynomial. Thanks to these a priori constraints, we consider the library $\{u, u_x, u_{xx}, v, v_x, v_{xx}, I, u^\alpha\}$ for $\widehat{F}_1(u, v)$ and the library $\{u, u_x, u_{xx}, v, v_x, v_{xx}\}$ for $\widehat{F}_2(u, v)$, where α takes the form $\alpha = 1.5 + 0.5 \sin(\eta)$ for $\eta \in \mathbb{R}$ to ensure that $\alpha \in [1, 2]$.

Table 5. Fixed parameters for Frac-PDE-Net.

Parameter	Value
t	[0, 0.75]
dt	0.05
x	$[-2.5\pi, 2.5\pi]$
dx	$\frac{5\pi}{200}$
N_{Init}	14
N_{Time}	15
N_{Space}	200

Table 6. Hyper-parameters selected for Frac-PDE-Net by the validation procedure as in Section 2.3.

Parameter	Value
λ_M (1% noise level)	1.88×10^{-7}
λ_S (1% noise level)	1.62×10^{-6}

The filters q are of size 1×19. The total number of parameters for approximating F_1 and F_2 is 29, and the number of trainable parameters in the moment matrices M is 32. To optimize the parameters, we again use the BFGS algorithm.

For this case, we added 1% noise to the generated data to form the observational data. The results are displayed in Table 7, in which the notations are consistent with those in Table 3.

Table 7. PDE model discovery with 1% noise level.

True F_1^*	$0.1\partial_x^2 u + 3.6u^{1.5} - 3.6u - 0.229u^{1.5}\int_{-2.5\pi}^{2.5\pi} u\,dx$ $+0.081\,\frac{u}{v^2+0.0215}$
$\widehat{F}_1^{mPDE-Net}$	$0.118\partial_x^2 u + 3.959u^{1.361} - 3.871u$ $-0.223u^{1.361}\int_{-2.5\pi}^{2.5\pi} u\,dx + 0.0749\,\frac{u}{(v+0.005)^2+0.0211}$ $0.0002\,\frac{uv}{(v+0.005)^2+0.0211} - 0.0029v$
$\widehat{F}_1^{smPDE-Net}$	$0.117\partial_x^2 u + 3.893u^{1.361} - 3.976u$ $-0.223u^{1.361}\int_{-2.5\pi}^{2.5\pi} u\,dx + 0.0750\,\frac{u}{(v+0.005)^2+0.0211}$
$\widehat{F}_1^{rsmPDE-Net}$ (Frac-PDE-Net)	$0.0899\partial_x^2 u + 3.441u^{1.508} - 3.363u - 0.244u^{1.508}\int_{-2.5\pi}^{2.5\pi} u\,dx$ $+0.0714\,\frac{u}{(v+0.0002)^2+0.0209}$
$\widehat{F}_1^{PHrsmPDE-Net}$	$-0.026\partial_x^2 u + 0.628u^{1.500} - 2.333u + 0.0393\,\frac{u}{(v-0.0479)^2+0.0154}$
True F_2^*	$10.0\partial_x^2 v + u - 0.4v$
$\widehat{F}_2^{mPDE-Net}$	$9.388\partial_x^2 v + 0.963u - 0.400v$
$\widehat{F}_2^{smPDE-Net}$	$9.388\partial_x^2 v + 0.963u - 0.400v$
$\widehat{F}_1^{rsmPDE-Net}$	$9.588\partial_x^2 v + 0.969u - 0.403\,v$
$\widehat{F}_1^{PHrsmPDE-Net}$	$8.145\partial_x^2 v + 0.937u - 0.387v$

Similar to the post hoc selection procedure we performed in Example 1, we also need to compare Model 1 ($\widehat{F}_1^{rsmPDE-Net}$) and Model 2 ($\widehat{F}_1^{PHrsmPDE-Net}$), and determine whether they differ significantly. Consider the time range to be $[0, 10]$ with time step size $dt = 0.05$. Hence, there are 200 time grids which are denoted to be $\{t_i\}_{i=1}^{200}$, where $t_i = 0.05i$, $1 \le i \le 200$. At each time t_i, we introduce the residuals $E_{t_i} := Y_{t_i} - Y_{t_i}^*$ and $\widetilde{E}_{t_i} := \widetilde{Y}_{t_i} - Y_{t_i}^*$, where Y_{t_i} and $\widetilde{Y}_{t_i}$ are associated to Model 1 and Model 2, respectively. We will test if the residuals $\{E_{t_i}\}_{i=1}^{200}$ and $\{\widetilde{E}_{t_i}\}_{i=1}^{200}$ have similar distributions or not. Analogous to the previous case, we see from Table 7 that the coefficient in front of the term $\partial_x^2 u$ in Model 2 ($\widehat{F}_1^{PHrsmPDE-Net}$) is a negative number -0.026, which leads to rapid concentration rather than diffusion effect. With this being said, Model 2 is essentially different from Model 1 and the distributions of $\{E_{t_i}\}_{i=1}^{200}$ and $\{\widetilde{E}_{t_i}\}_{i=1}^{200}$ are totally different.

To assess the stability of the results shown above, we repeated the experiments 100 times and the results are presented in Figures 6 and 7. The plots show that there are some instances where the three methods fail to eliminate certain redundant terms. However, these instances are rare, as the median of these terms is 0, indicating that they appear infrequently.

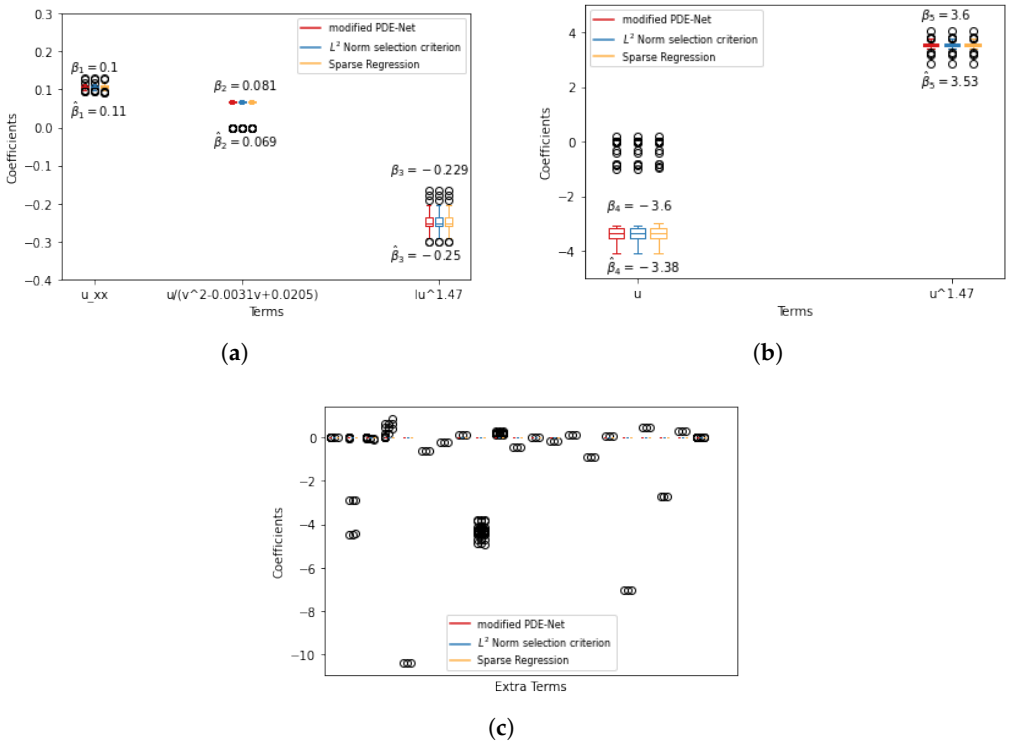


Figure 6. Simulation results for $\widehat{F}_1(u, v)$ with 1% noise. (**a**) True positive discovering. (**b**) True positive discovering. (**c**) False positive discovering.

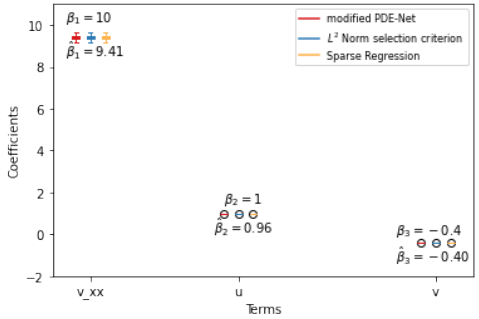


Figure 7. Simulation results for $\widehat{F}_2(u, v)$ with 1% noise. True positive discovering.

4. Prediction

4.1. Example 1: The 2-Dimensional Model

In this section, we validate the robustness of the model discovered by Frac-PDE-Net in Example 1 by performing predictions with non-typical initial values u_0 and v_0,

$$u_0 = \frac{50y^2 - y^4 + 4}{800[1.2 - \cos(\frac{\pi}{5}y)]} + 4, \qquad v_0 = \frac{1}{800}(50y^2 - y^4 + 4)\left[2 + \cos\left(\frac{\pi}{5}x\right)\right] + 1.$$

We use the finite difference method to generate the "true data" in the forward direction using the known coefficients and terms in (16) and (17). The spatial step sizes (dx and dy) are set to $\frac{10}{64}$ and the time step size (dt) is $\frac{1}{1600}$. We then simulate the data using the trained model from Table 3 up to $t = 0.5$.

In Figure 8, both the true solution (u, v) and the predicted solution $(\tilde{u}, \tilde{v})$ of the trained model by Frac-PDE-Net are plotted at different time instances: $t \in \{0.4, 0.6, 0.8, 1\}$. One can see from Figure 8 that the predicted solution is very close to the true one.

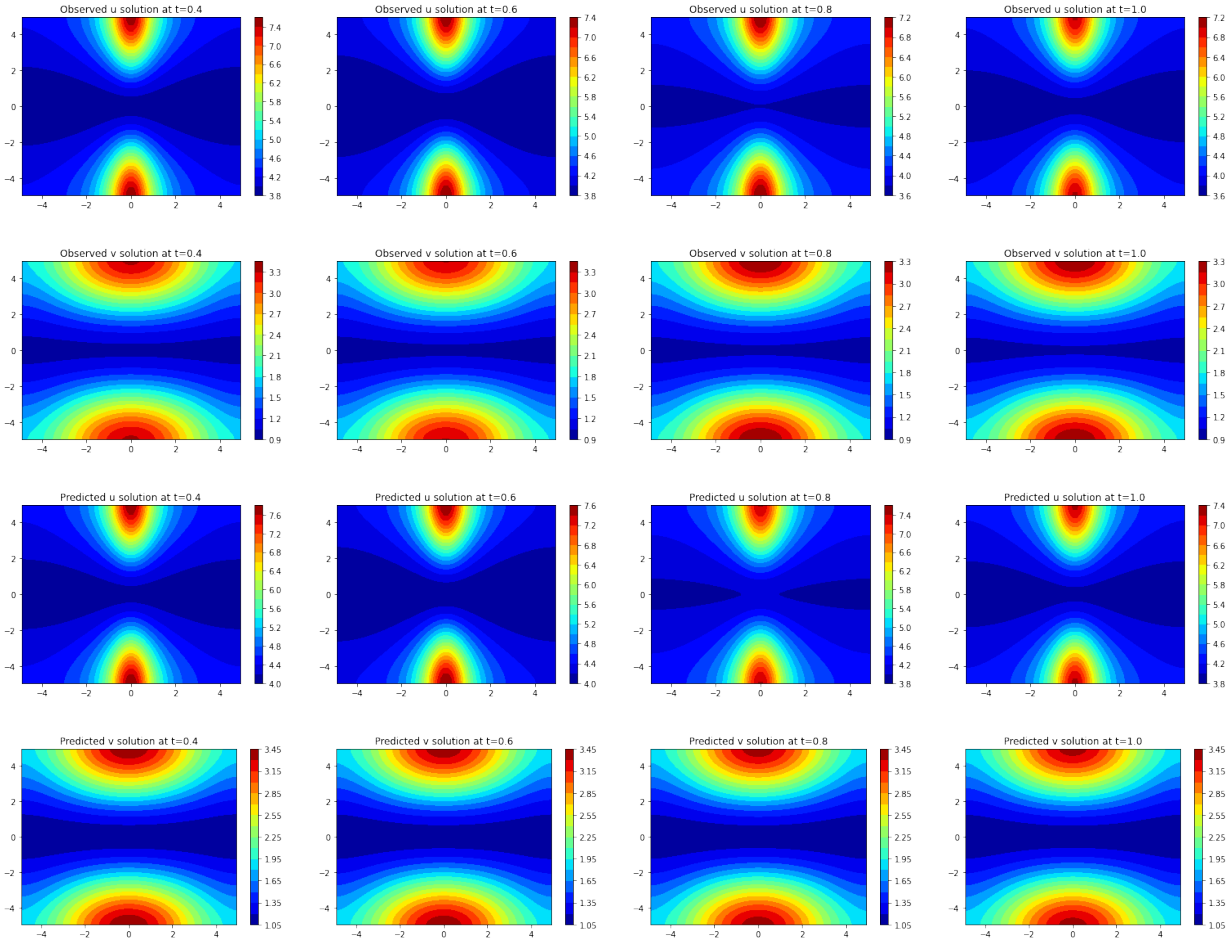


Figure 8. The first (second resp.) row shows the true dynamics of u (v resp.) at times $t = 0.4, 0.6, 0.8, and 1.0$. The third (fourth resp.) row shows the predicted dynamics of u (v resp.) with 5% noise level using Frac-PDE-Net.

The results of the comparison between Frac-PDE-Net and PDE-Net 2.0 are presented in both graphical and quantitative form. The model discovered by PDE-Net 2.0 is shown in Table 8, while the predicted solutions are displayed in Figure 9. Although PDE-Net 2.0 only utilizes polynomials, the predicted images still have a similar shape to the true ones. To further evaluate the performance, the predicted errors are analyzed quantitatively using the L^∞ norm and L^2 norm on the space domain $[-5, 5] \times [-5, 5]$, as seen in Table 9. The results show that Frac-PDE-Net has smaller errors compared to PDE-Net 2.0, highlighting its advantage.

Table 8. PDE model discovered by PDE-Net 2.0.

	Predicted Terms by PDE-Net 2.0 with 5% Noise
$\widehat{F}_1(u, v)$	$0.0457\Delta u - 1.765u + 0.0938v + 0.0008$
$\widehat{F}_2(u, v)$	$0.243\Delta v - 0.604u - 0.277v + 7(10^{-5})$

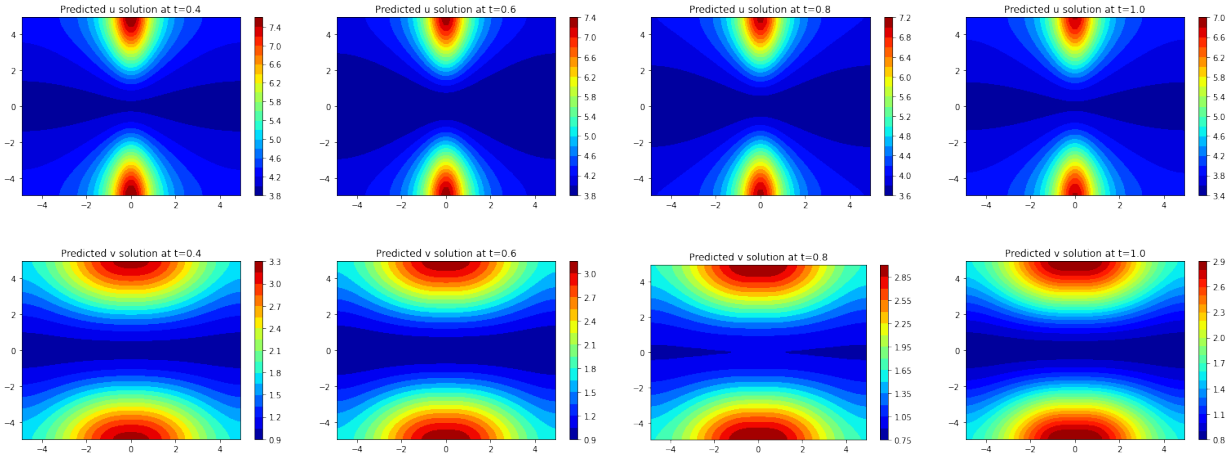


Figure 9. Images of the predicted dynamics using PDE-Net 2.0 with 5% noise level.

Table 9. Errors of predicted solutions for u and v by Frac-PDE-Net and PDE-Net 2.0.

	Noise	**Frac-PDE-Net**				**PDE-Net 2.0**			
$\lvert\tilde{u}-u\rvert$		$t=0.4$	$t=0.6$	$t=0.8$	$t=1$	$t=0.4$	$t=0.6$	$t=0.8$	$t=1$
L^∞	5%	0.007254	0.010806	0.014310	0.017765	0.227365	0.331438	0.429602	0.522157
L^2	5%	0.000106	0.000158	0.000209	0.000260	0.002720	0.003986	0.005192	0.006341
$\lvert\tilde{v}-v\rvert$		$t=0.4$	$t=0.6$	$t=0.8$	$t=1$	$t=0.4$	$t=0.6$	$t=0.8$	$t=1$
L^∞	5%	0.001503	0.002247	0.002988	0.003725	0.200241	0.293939	0.383577	0.469314
L^2	5%	0.000022	0.000033	0.000044	0.000054	0.001989	0.002930	0.003836	0.004708

4.2. Example 2: The One-Dimensional Model

In this section, we validate the robustness of the model discovered by Frac-PDE-Net in Example 2 in Section 3.2 by performing predictions with the following periodic initial values u_0 and v_0,

$$u_0(x) = 0.0259 + 0.01\sin(3x), \quad v_0(x) = 0.06475 + 0.01\sin(3x).$$

We use finite difference method to generate the "true" data in the forward direction using the known coefficients and terms in (20) and (21). The spatial step sizes (dx and dy) are set to $\frac{5\pi}{200}$ and the time step size (dt) is $\frac{5}{100}$. The time interval considered is $t \in [0, 10]$. We then simulate the data using the trained model from Table 7 over the time period $[0, 10]$. In Figure 10, both the true solution and the predicted solution of the trained model by Frac-PDE-Net are plotted for $t \in [0, 10]$. One can see from Figure 10 that the predicted solution is very close to the true one.

The results of the comparison between Frac-PDE-Net and PDE-Net 2.0 are presented in both graphical and quantitative form. The model discovered by PDE-Net 2.0 is shown in Table 10, while the predicted solutions are displayed in Figure 11. We can clearly see that the predicted images by PDE-Net 2.0 are far from satisfactory compared to the true one in Figure 10. To further evaluate the performance, the predicted errors are analyzed quantitatively using the L^∞ norm and L^2 norm on the space-time region $\left[-\frac{5\pi}{2}, \frac{5\pi}{2}\right] \times [0, 10]$ in Table 11. The results show that Frac-PDE-Net has much smaller errors compared to PDE-Net 2.0, highlighting its advantage.

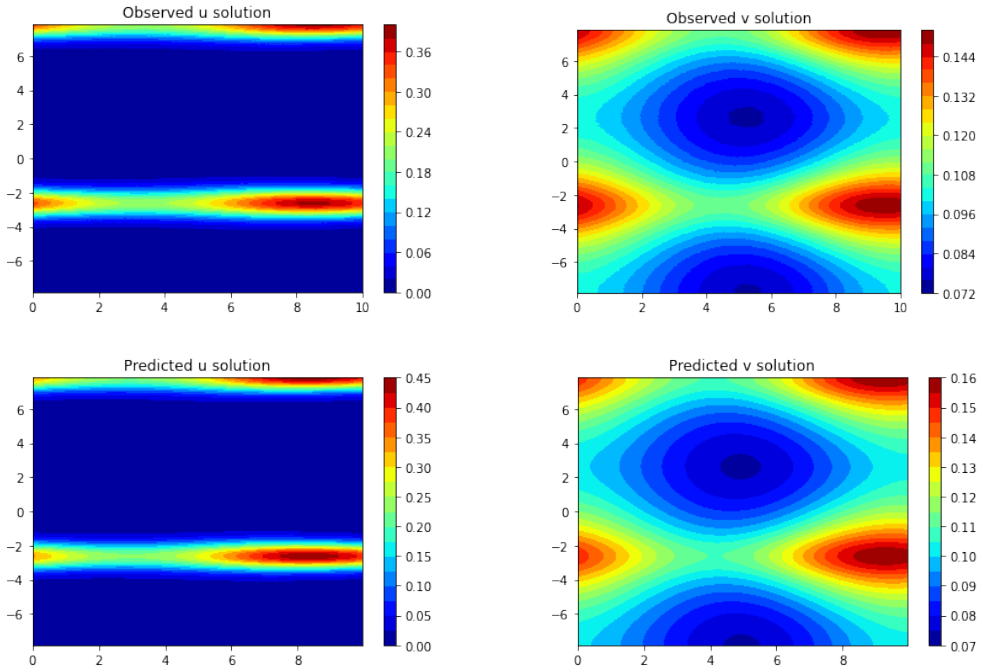


Figure 10. The first row shows the true dynamics of (u, v) for $(x, t) \in \left[-\frac{5\pi}{2}, \frac{5\pi}{2}\right] \times [0, 10]$. The second row presents the predicted dynamics of (u, v) with 1% noise level by Frac-PDE-Net.

Table 10. PDE model discovered by PDE-Net 2.0.

	Predicted Terms by PDE-Net 2.0 with 1% Noise
$\widehat{F}_1(u, v)$	$0.0001\partial_x^2 u - 3.95(10^{-5})u - 6.05(10^{-5})v - 0.0002$
$\widehat{F}_2(u, v)$	$5.22(10^{-5})\partial_x^2 v + 1.70(10^{-5})u + 8.19(10^{-6})v + 4.59(10^{-5})$

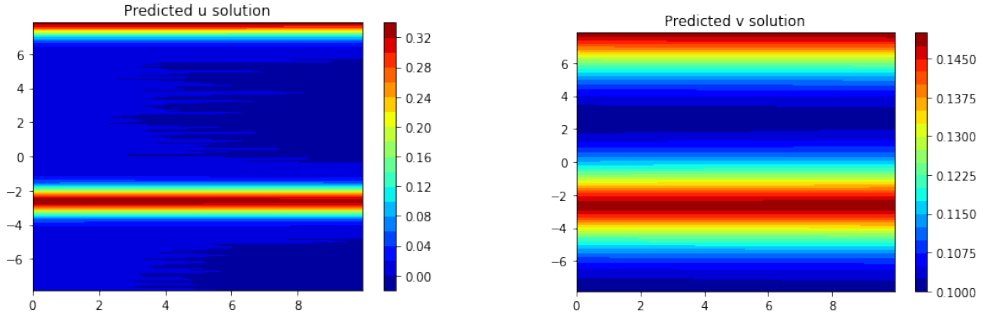


Figure 11. Images of the predicted dynamics of (u, v) for $(x, t) \in \left[-\frac{5\pi}{2}, \frac{5\pi}{2}\right] \times [0, 10]$ using PDE-Net 2.0 with 1% noise level.

Table 11. Errors of predicted solutions for u and v by Frac-PDE-Net and PDE-Net 2.0.

	Noise	**Frac-PDE-Net**	**PDE-Net 2.0**
$\lvert\tilde{u} - u\rvert$		$t \in [0, 10]$	$t \in [0, 10]$
L^∞	1%	0.062771	0.117773
L^2	1%	0.000029	0.000060
$\lvert\tilde{v} - v\rvert$		$t \in [0, 10]$	$t \in [0, 10]$
L^∞	1%	0.009434	0.039400
L^2	1%	0.000010	0.000056

5. Conclusions

Our approach, Frac-PDE-Net, builds on the symbolic approach developed in PDE-Net for addressing the discovery of realistic and interpretable PDE from data. While the neural network remains very efficient for generating and learning dictionaries of functions, typically polynomials, we have shown that if we enrich the dictionaries with large families of functions (typically uncountable), an extra-care is needed for selecting the important terms by penalization and by evaluating and testing the impact of a reaction term in the predicted solution. Quite remarkably, we can extract a sparse equation with readable terms and with good estimates of the associated parameters.

The introduction of rich families of functions, such as fractions (rational functions) is often necessary because they are well used by modelers, but also they can avoid the limitations of the approximation capacity of polynomials. Indeed, it might be necessary to have numerous terms in the expansion in order to have a correct approximation of the unknown reaction terms. As a matter of fact, we have introduced a very flexible family of fractions that avoid truncation based on powers $u^p, v^q, q, p \in \mathbb{N}$. While we learn then the numerator and denominator coefficients in $\mathbb{R}$, our approach is incorporated seamlessly in the symbolic differentiable neural network framework of PDE-Net by the introduction of extra layers.

Our work is originally motivated by the discovery and estimation of reaction–diffusion PDEs, with possibly complex terms such as fractions, non-integer powers, or non-local terms (such as an integral), as it has been introduced for the pollen tube growth problem [6]. Nevertheless, our selection approach could be used to handle other dictionaries, or in the presence of advection terms as our methodology does exploit the reaction–diffusion structure only for imposing some constraints on the dictionaries of interest, and because of the interpretability of each term in that case. As the next steps, the Frac-PDE-Net methodology can be improved by considering more advanced numerical schemes in time discretization, say implicit Euler or second-order Runge–Kutta. In that case, we expect to have a better accuracy and stability for model recovery and prediction. Another possible improvement would be to enrich the dictionaries of fractionals by replacing the current form $N(u, v) = g(u)h(v)$ by more rational functions with denominators that depends both on u and v, say $N(u, v) = \frac{u-v}{u^2-v^2+1}$. Finally, we put an emphasis on the fact that Frac-PDE-Net reaches a trade-off by discovering the main terms of the PDE, accurately estimating each coefficient in order to gain interpretability, while it also allows effective long-term prediction, even for unseen initial conditions.

Author Contributions: Conceptualization, N.J-B.B., X.C.; methodology, S.C., X.Y., N.J-B.B., X.C.; software, S.C., X.Y.; validation, S.C., X.Y., N.J-B.B., X.C.; formal analysis, S.C., X.Y., N.J-B.B., X.C.; writing—original draft preparation, S.C., X.Y., N.J-B.B., X.C.; writing—review and editing, S.C., X.Y., N.J-B.B., X.C.; supervision, N.J-B.B., X.C.; funding acquisition, X.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by United States Department of Agriculture (USDA) National Institute of Food and Agriculture (NIFA) Hatch Project AES-CE award (CA-R-STA-7132-H) and NSF DMS 1853698.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank all anonymous reviewers for their constructive comments and suggestions.

Conflicts of Interest: The authors declared no conflict of interest.

Appendix A

Appendix A.1. Term Combination after Simulation

During the process of simulation, if only the addition and the multiplication operators are involved, then it is not an issue to combine terms as the program can easily identify same terms and then add their coefficients together. However, combining similar terms can be difficult when fractional terms are present. To address this issue, we classify the simulation results into various groups before combining them.

As an example, we consider the scenario where the nonlinear term takes the form $g(u)h(v)$, and one of the following two structures is assumed.

(i) g is linear and h is a fractional function whose denominator is a second order polynomial:

$$(u + c_1)\,\frac{\alpha_1 v + \alpha_2}{v^2 + \beta_1 v + \beta_2}.$$

(ii) h is linear and g is a fractional function whose denominator is a second order polynomial:

$$\frac{\alpha_3 u + \alpha_4}{u^2 + \beta_3 u + \beta_4}\,(v + c_2).$$

Therefore, the outcomes have 32 possibilities if we only classify terms and signs:

- Numerator (4 possibilities): u, v, uv, 1.
- Denominator (2 possibilities): quadratic function in u or v.
- Signs (4 possibilities): the sign of β_1 or β_2 can be either positive or negative.

There are now 32 groups. In each of them, all members share the same main terms and same signs in the denominator while the coefficients are allowed to be different. For example, in the group with the form

$$\frac{\alpha_1 uv}{v^2 + \beta_1 v + \beta_2},$$

all members share the same term uv in the numerator, same terms v^2 and v in the denominator, and same signs of β_1 and β_2, while the specific values of α_1, β_1 and β_2 may vary.

Based on the above groups, we will adopt the following general principle to proceed. If two terms live in distinct groups, then they are considered to be different and will not be combined. If two terms live in the same group, then we will further quantify how close their coefficients in the denominator (say β_1 and β_2) are. If these coefficients are close enough, then we will regard them as the "same" term and combine them by adding their coefficients in the numerator (say α_1) together. So, the next question is how to quantify the distance between two members in the same group with possibly different coefficients (say β_1 and β_2).

We will illustrate the criterion in the following by studying a specific form $\frac{uv}{v^2+\beta_1 v+\beta_2}$. More precisely, suppose there are two terms T_1 and T_2 as below,

$$T_1 = \frac{\alpha_1^{(1)} uv}{v^2 + \beta_1^{(1)} v + \beta_2^{(1)}}, \quad T_2 = \frac{\alpha_1^{(2)} uv}{v^2 + \beta_1^{(2)} v + \beta_2^{(2)}},$$

then we define their distance to be

$$D\big[T_1, T_2\big] = \max_{i=1,2} \left\{ \frac{|\beta_i^{(2)} - \beta_i^{(1)}|}{\max\{|\beta_i^{(2)}|, |\beta_i^{(1)}|\}} \right\}. \tag{A1}$$

According to this concept, we combine T_1 and T_2 together if and only if $D[T_1, T_2] < 0.2$, that is when the relative difference between the coefficients is less than 0.2. In such a case, we add the coefficients $\alpha_1^{(1)}$ and $\alpha_1^{(2)}$ to obtain

$$T_1 + T_2 \approx T^* := \alpha^* \frac{uv}{v^2 + \beta_1^{(*)} v + \beta_2^{(*)}},$$

where

$$\alpha^* = \alpha_1^{(1)} + \alpha_1^{(2)}, \quad \beta_1^{(*)} = \frac{1}{2}\left[\beta_1^{(1)} + \beta_1^{(2)}\right], \quad \beta_2^{(*)} = \frac{1}{2}\left[\beta_2^{(1)} + \beta_2^{(2)}\right].$$

References

1. Turing, A.M. The Chemical Basis of Morphogenesis. *Philos. Trans. R. Soc. Lond. Ser. B, Biol. Sci.* **1952**, *237*, 37–72.
2. Murray, J.D. *Mathematical Biology, II*, 3rd ed.; Interdisciplinary Applied Mathematics; Springer: New York, NY, USA, 2003; Volume 18, p. xxvi+811.
3. Mori, Y.; Jilkine, A.; Edelstein-Keshet, L. Wave-pinning and cell polarity from a bistable reaction-diffusion system. *Biophys. J.* **2008**, *94*, 3684–3697. [CrossRef]
4. Mogilner, A.; Allard, J.; Wollman, R. Cell polarity: Quantitative modeling as a tool in cell biology. *Science* **2012**, *336*, 175–179. [CrossRef]
5. Tian, C. Parameter Estimation Procedure of Reaction Diffusion Equation with Application on Cell Polarity Growth. Ph.D. Thesis, UC Riverside, Riverside, CA, USA, 2018.
6. Tian, C.; Shi, Q.; Cui, X.; Guo, J.; Yang, Z.; Shi, J. Spatiotemporal dynamics of a reaction-diffusion model of pollen tube tip growth. *J. Math. Biol.* **2019**, *79*, 1319–1355. [CrossRef] [PubMed]
7. Lu, L.; Meng, X.; Mao, Z.; Karniadakis, G.E. DeepXDE: A deep learning library for solving differential equations. *arXiv* **2019**, arXiv:1907.04502.
8. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. *arXiv* **2017**, arXiv:1711.10561.
9. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *J. Mach. Learn. Res.* **2018**, *19*, 932–955.
10. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
11. Meng, X.; Li, Z.; Zhang, D.; Karniadakis, G.E. PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Comput. Methods Appl. Mech. Eng.* **2020**, *370*, 113250. [CrossRef]
12. Pang, G.; Lu, L.; Karniadakis, G.E. fPINNs: Fractional physics-informed neural networks. *SIAM J. Sci. Comput.* **2019**, *41*, A2603–A2626. [CrossRef]
13. Chen, Z.; Xiu, D. On generalized residual network for deep learning of unknown dynamical systems. *J. Comput. Phys.* **2021**, *438*, 110362. [CrossRef]
14. Wu, K.; Xiu, D. Data-driven deep learning of partial differential equations in modal space. *J. Comput. Phys.* **2020**, *408*, 109307. [CrossRef]
15. Zhou, Z.; Wang, L.; Yan, Z. Deep neural networks for solving forward and inverse problems of (2 + 1)-dimensional nonlinear wave equations with rational solitons. *arXiv* **2021**, arXiv:2112.14040.
16. Long, Z.; Lu, Y.; Dong, B. PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *J. Comput. Phys.* **2019**, *399*, 108925. [CrossRef]
17. Long, Z.; Lu, Y.; Ma, X.; Dong, B. PDE-Net: Learning pdes from data. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 3208–3216.
18. Pakravan, S.; Mistani, P.; Aragon-Calvo, M.; Gibou, F. Solving inverse-PDE problems with physics-aware neural networks. *J. Comput. Phys.* **2021**, *440*, 110414. [CrossRef]
19. Daneker, M.; Zhang, Z.; Karniadakis, G.; Lu, L. Systems Biology: Identifiability analysis and parameter identification via systems-biology informed neural networks. *arXiv* **2022**, arXiv:2202.01723.
20. Both, G.; Choudhury, S.; Sens, P.; Kusters, R. DeepMoD: Deep learning for model discovery in noisy data. *J. Comput. Phys.* **2021**, *428*, 109985. [CrossRef]
21. Xu, H.; Chang, H.; Zhang, D. DL-PDE: Deep-learning based data-driven discovery of partial differential equations from discrete and noisy data. *arXiv* **2019**, arXiv:1908.04463.
22. Chen, Y.; Luo, Y.; Liu, Q.; Xu, H.and Zhang, D. Symbolic genetic algorithm for discovering open-form partial differential equations (SGA-PDE). *Phys. Rev. Res.* **2022**, *4*, 023174. [CrossRef]
23. Zhang, Z.; Liu, Y. Robust data-driven discovery of partial differential equations under uncertainties. *arXiv* **2021**, arXiv:2102.06504.
24. Bhowmick, S.; Nagarajaiah, S. Data-driven theory-guided learning of partial differential equations using simultaneous basis function approximation and parameter estimation (SNAPE). *arXiv* **2021**, arXiv:2109.07471.

25. Rudy, S.H.; Brunton, S.L.; Kutz, J.N. Data-driven discovery of partial differential equations. *Sci. Adv.* **2017**, *3*, e1602614. [CrossRef] [PubMed]
26. Rudy, S.; Alla, A.; Brunton, S.L.; Kutz, J.N. Data-driven identification of parametric partial differential equations. *SIAM J. Appl. Dyn. Syst.* **2019**, *18*, 643–660. [CrossRef]
27. Cai, J.; Dong, B.; Osher, S.; Shen, Z. Image restoration: Total variation, wavelet frames, and beyond. *J. Amer. Math. Soc.* **2012**, *25*, 1033–1089. [CrossRef]
28. Brunel, N.J-B. Parameter estimation of ODE's via nonparametric estimators. *Electron. J. Statist.* **2008**, *2*, 1242–1267. [CrossRef]
29. Bergstra, J.; Yamins, D.; Cox, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Proceedings of the International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; pp. 115–123.
30. Dunn, O. Multiple comparisons among means. *J. Am. Stat. Assoc.* **1961**, *56*, 52–64. [CrossRef]
31. Holm, S. A simple sequentially rejective multiple test procedure. *Scand. J. Stat.* **1979**, *6*, 65–70.
32. Benjamini, Y.; Hochberg, Y. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *J. R. Stat. Soc. Ser. B* **1995**, *57*, 289–300. [CrossRef]