Discovering Nonlinear Dynamics Through Scientific Machine Learning

Lei Huang[1(✉)], Daniel Vrinceanu[2], Yunjiao Wang[2], Nalinda Kulathunga[2], and Nishath Ranasinghe[1]

[1] Department of Computer Science, Prairie View A&M University, Prairie View, TX 77446, USA
lhuang@pvamu.edu
[2] Texas Southern University, Houston, TX 77004, USA
https://computinglab.wixsite.com/computinglab

Abstract. Scientific Machine Learning (SciML) is a new multidisciplinary methodology that combines the data-driven machine learning models and the principle-based computational models to improve the simulations of scientific phenomenon and uncover new scientific rules from existing measurements. This article reveals the experience of using the SciML method to discover the nonlinear dynamics that may be hard to model or be unknown in the real-world scenario. The SciML method solves the traditional principle-based differential equations by integrating a neural network to accurately model the nonlinear dynamics while respecting the scientific constraints and principles. The paper discusses the latest SciML models and apply them to the oscillator simulations and experiment. Besides better capacity to simulate, and match with the observation, the results also demonstrate a successful discovery of the hidden physics in the pendulum dynamics using SciML.

Keywords: Scientific machine learning · Scientific simulation · Computational science · Nonlinear dynamics

1 Introduction

Scientific Machine Learning (SciML) [1] has recently emerged as a new method to solve the scientific computing problems using machine learning models. The method leverages the success of traditional scientific computational models and the advances in data-driven machine learning models to augment the efficiency and accuracy of scientific simulation and inversion. Moreover, it facilitates the scientific discovery by modeling both well-known scientific rules and the unknown patterns based on observed data.

The traditional scientific computational models mostly are developed to simulate the physics, chemistry, biology and other scientific phenomenons by using various numerical methods, such as the finite difference or finite element methods, to solve a variety of differential equations. These methods can achieve highly

K. Arai (Ed.): IntelliSys 2021, LNNS 294, pp. 261–279, 2022.
https://doi.org/10.1007/978-3-030-82193-7_17

accurate simulation results; however, they are also notoriously expensive in consuming computational resources. It is why scientific computing is typically conducted on the supercomputers using complex programming models. Moreover, the scientific computing highly depends on our understanding of the theoretical principles, which do not fully represent the complexity and nonlinear dynamics in many real-world phenomenons. Many times, the parameters and other constraints are not well known and simulation scientists have nothing better to rely on educated guesses.

In theory, SciML combines the deterministic scientific principles with the universal approximation of machine learning to thus provide a more efficient yet reliable and explainable model-based data-driven solution. SciML provides a sound scientific theoretic foundation to unveil the new scientific governing rules with a collection of data and models. For example, SciML may integrate a deep learning model into a partial differential equation (PDE) to fit the observed data, which models the well-known principles using the PDE and models the unknown portion such as noise and friction using the deep learning model.

The latest theoretical and practical advances in machine learning, especially deep learning, dramatically increase the capacity and accuracy of the universal approximation of nonlinear functions. Despite the progress, it is still not reliable and explainable to learn a complex system with nonlinear dynamics or chaos using deep learning along. Moreover, it requires huge mostly unrealistic big data sets to train a deep learning model to cover all possible features. Even if we can successfully train a deep learning-based surrogate model, the model's extrapolation is not questionable. It would be much more reliable and explainable if we can embed the physical principles to determine the nonlinear dynamics, and only leave the unknown functions to machine learning. SciML converges the computational science and data science disciplines powers to improve the accuracy, performance, and interpretability in scientific simulation. Moreover, it may unveil scientific rules hidden inside the nonlinear dynamics learned by the machine learning models.

In this paper, we present the latest four SciML models using a couple of physics experiments to report our experience, the benefits, and limitations of the SciML method. Section 2 describes the state-of-art of SciML models; Sect. 3 shows the physics experiments, simulation, and data collection; Sect. 4 discusses the results of the SciML models; and the Sect. 5 concludes the findings of the paper.

2 Scientific Machine Learning Models

Scientific machine learning is developed to facilitate the scientific computation either by developing a surrogate model to replace the numerical model or combine the data-driven model to achieve better accuracy and performance. In this paper, we applies the Physics-Informed Neural Networks (PINNs) [29], the Universal Differential Equation (UDE) method [27], the Hamiltonian Neural Networks (HNNs) [11], and the Neural Ordinary Differential Equation (NODE) [7] to learn the nonlinear dynamics in several physics experiments.

2.1 Physics-Informed Neural Networks

The Physics Informed Neural Networks (PINNs) is one of SciML solutions that solves the differential equations by modeling the latent solution $u(t, x)$ directly with a deep learning model and solves the differential equation by taking advantage of automatic differentiation functionalities [2] in machine learning (ML) software. The solution $u(t, x)$ is replaced by a neural network or other machine learning model and its derivatives satisfy the definition of the governing differential equation.

For example, the harmonic pendulum differential equation is defined as the Eq. (1).

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\sin(\theta) = 0 \tag{1}$$

where g is gravity, L is the length of the pendulum and θ is the angle with respect to the vertical in radians.

PINNs redefines the equation by substitute its solution $f(t)$ with a neural network $N_p(t)$, where N is the neural network and p is its trainable parameters. The new equation is depicted as the Eq. (2).

$$N_p^{''}(t) + \frac{g}{L}\sin(\theta) = 0 \tag{2}$$

By creating a loss function to minimize the Eq. (2), PINNs utilizes the automatic differentiation capability in machine learning software to calculate the second order derivatives of $N_p(t)$ and optimize the loss function. As the result, the $N_p(t)$ is an approximation of the solution of Eq. (1).

Furthermore, PINNs can be effectively used to solve the forward problem as well as the inverse problem with minimum modifications to computational codes [18]. Additionally, a Petrov-Galerkin [10] version of PINNs have been employed to solve variational form of PDEs to reduce the training cost [14]. Likewise, modified versions of PINNs have been employed to solve fractional differential equations [23] and stochastic differential equations [34]. As a method to address lack of uncertainty quantification in PINNs, Zhang *et al.* [35] put forward the idea of using multiple deep neural networks to quantify the parametric uncertainty and dropouts to model the uncertainty stemming from the approximations resulting from the neural networks. As an effort to develop a theoretical basis of PINNs, Shin *et al.* [31] studied the convergence of the sequence of minimizers generated from PINNs corresponding to the sequence of neural networks to the solution of the given PDE. They found sequence of the minimizers strongly converges to the PDE solution in L2 space as well as each minimizer satisfies both initial and boundary conditions.

2.2 Universal Differential Equations

The Universal Differential Equations (UDE) method has some similarities with the PINN method: both rely on the scientific principles represented as differential equations to guide the computation and impose constraints. However, UDE is

more flexible to model the unknown functions and combine them with existing scientific knowledge. UDE relies on the numerical differential equation solvers to solve the problem while learning the unknown functions during the calculation.

The pendulum equation using UDE is depicted as the Eq. (3), which introduces a neural network N_p that represents the unknown function is the experiment, such as the friction and/or the external force. As the result, the UDE solution better fits with the observed experiment results as described in Sect. 3.

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\sin(\theta) + N_p(u) = 0 \tag{3}$$

The method designs a machine learning model representing unknown physical functions while computing the ODE numerically using the ODE solver. The benefit of using the UDE method is that the neural network does not learn the full dynamics, which may be extremely hard or even impossible due to the nonlinear dynamics in chaotic systems. The approximation caused by the neural network may much diverge the long-term prediction results if we simply use the data-driven statistics-based machine learning model. It is hard to believe that a neural network's universal approximation can be accurate enough for nonlinear dynamics prediction. The physical principles in dynamics need to be honored during the calculation. The UDE method applies the powerful universal approximation capacity in machine learning and respects physical constraints such as symmetry, invariance, and conservation.

2.3 Hamiltonian Neural Networks

In classical mechanics, Hamiltonian equations are widely adopted to model continuous time evolution of dynamic systems with physically conserved quantities such as energy and they can be effectively used to predict the phase space of dynamic system's using the current state of the generalized position and momentum. Additionally, Hamiltonian mechanics are smooth, time reversible and provide integral paths that conserve certain physical quantities such as energy. Greydanus *et al.* [11] introduced Hamiltonian neural networks (HNN) by incorporating Hamiltonian equations into the loss function of the neural network to learn the Hamiltonian of simple systems with noisy phase space data. Additionally, Toth *et al.* [33] used a generative model to infer the Hamiltonian from dynamic systems using high dimension data (pixel). Matteakis *et al.* [20] embedded physical constraints into the structure of the neural network using the Hamiltonian equations deviating from other studies using the HNN method.

HNN may also help reduce the expensive computational costs for solving scientific problems. The HNN method creates a neural network N_p that meets the following requirements:

$$\frac{dx_1}{dt} = \frac{\partial H}{\partial x_2} = \frac{\partial N_p}{\partial x_1}, \quad \frac{dx_2}{dt} = -\frac{\partial H}{\partial x_1} = -\frac{\partial N_p}{\partial x_2} \tag{4}$$

where (x_1, x_2) are two inputs of the HNN network, and denote the position and momentum.

2.4 Neural Ordinary Differential Equations (Neural ODE)

Since it was first observed by Weinan E [8], the relation between ResNet [13] and dynamical systems has been widely explored and utilized to increase the capability and stability of deep networks [4–6,17,19,32]. Connecting deep networks with ODE was largely inspired by the success of ResNet, whose network architecture is strikingly similar to the well-known Euler method for differential equations. With this observation, a natural idea is to generalize existing numerical methods to deep networks [19,36]. Neural ODE, proposed by Chen et al. [6], went one step further: it replaces deep networks such as ResNet with existing efficient ODE solvers.

One key difference between solving ODE and training deep networks is that their goals are different. The goal of training deep networks is to find functions that fit the data while numerical ODE is to approximate solutions of the ODE. The idea of the dynamical systems approach for deep networks is to tune the vector field so that its flow map can reproduce nonlinear functions needed to fit the data [8]. More specifically, consider

$$\frac{dz}{dt} = f(z, t, \theta), \quad z(0) = x \tag{5}$$

Let $z(t, x, p)$ be the solution to the initial value problem (5), let $T > 0$ be a fixed time and p be a set of parameters. The flow map

$$x \to z(T, x, \theta)$$

defines a function from input to output, which is generally nonlinear [8]. Here f could be a neural network to model vector fields.

Neural ODE uses existing solvers to solve the ODE (5) for a given set of parameter and input values. The step after solving the ODE is to adjust the values of p and repeat the process to find optimal values for p so that the flow map fits the data best. Just as regular optimization, this process requires to compute the gradient of a designed loss function with respect to p. A beautiful benefit coming out of Neural ODE approach is that the computation of gradient is easier and independent of the solver and can be carried out by the classical adjoint sensitivity method [26]. Another benefit of this method is that Neural ODE can naturally used for time dependent data as the pendulum data discussed in this paper.

A computational disadvantage is that ODE solver often requires a larger number of evaluations than in a standard deep network, which tends to get worse over the training [16].

3 Physical Experiments

3.1 Quadruple Spring Mass System

A quadruple-springs-mass system allowed to move in a 2-D frictionless surface (Fig. 1) can exhibit simple harmonic motion as well as non-linear dynamic motion. The motion depends on the initial conditions of the system and also on the physical properties of the springs (i.e. spring constants and unstretched lengths of the springs). For simplicity, here we only consider massless springs.

The time independent Hamiltonian of the quadruple-springs-mass system can be given using generalized position(x, y) and momentum(p_x, p_y) as;

$$H = \frac{p_x^2 + p_y^2}{2m} + \frac{1}{2}\sum_{i=1}^{n=4} k_i(l_i - a_i)^2 \tag{6}$$

where: a_i = un-stretched lengths of the springs,
l_i = stretched lengths of the springs,
k_i = springs constants,
m = mass of the particle in the middle

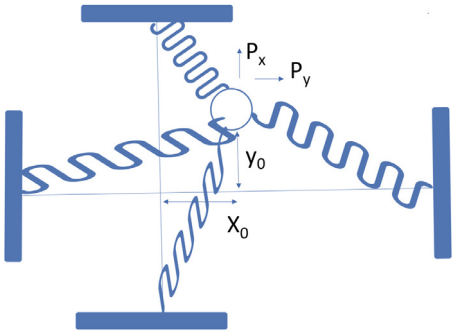


Fig. 1. Quadruple-springs-mass system

Where, $l_1 = \sqrt{(a_1 + x)^2 + y^2}$, $l_2 = \sqrt{(a_2 - x)^2 + y^2}$, $l_3 = \sqrt{(a_3 - y)^2 + x^2}$ and $l_4 = \sqrt{(a_4 + y)^2 + x^2}$. We simulate one instance of simple harmonic motion and another instance of non-linear dynamic motion of the quadruple-springs-mass system by solving the Hamiltonian equations (Eq. 4) while utilizing the Hamiltonian given in Eq. 6. Initial conditions for the simple harmonic motion and the non-linear dynamic motion are given in Table 1. The data generated for a period of 5π from the two experiments.

Table 1. Initial conditions for the simple harmonic motion and non-linear dynamic motion

Motion type	Unstretched length	Spring const	Init. pos	Init. moment	Mass
Parameters	a_1, a_2, a_3, a_4	k_1, k_2, k_3, k_4	x_0, y_0	P_{x0}, P_{y0}	m
SHM	$1, 1, 1, 1$	$1, 1, 1, 1$	$-0.2, -0.2$	0.1, 0.1	1.0
Nonlinear dynamics	$1, 2, 3, 4$	$4, 3, 2, 1$	$-0.2, 0.1$	$0.1, -0.2$	1.0

3.2 Pendulum

A pendulum is a classical physical phenomenon that has been studied to understand its dynamics for a long time. Figure 2 shows a simple gravity pendulum with angle (θ) and the length of slender rod L, the mass of pendulum bob m, and its angular velocity $\omega = \frac{d\theta}{dt}$.

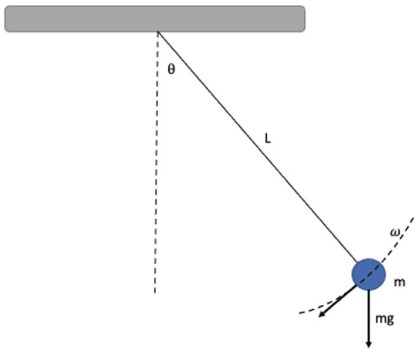


Fig. 2. Pendulum motion

The simple gravity pendulum [22] is a harmonic motion without any friction or external forces, which is governed by the simple second-order differential Eq. 1.

For Eq. (1), we may use numerical methods to solve the ODE by specifying a small-enough time step. The results are a sequence of the angles θ and the angular velocities $\frac{d\theta}{dt}$ for each time step during the pendulum simulation time span.

The differential equation (1) is changed by adding the air resistance or friction component to simulate a damping harmonic pendulum, linear or nonlinear, depending on the scenario. Equation (7) shows linear air resistance/friction integrated into the motion to slow the pendulum down gradually.

$$\frac{d^2\theta}{dt^2} + \mu\frac{d\theta}{dt} + \frac{g}{L}\sin(\theta) = 0 \tag{7}$$

where: $\mu\frac{d\theta}{dt}$ = the linear air resistance/friction.

The air resistance/friction may also be nonlinear represented as a polynomial function such as: $\mu_2(\frac{d\theta}{dt})^2 + \mu_1\frac{d\theta}{dt}$, then the Eq. (7) is changed to Eq. (8).

$$\frac{d^2\theta}{dt^2} + \left(\mu_2\left(\frac{d\theta}{dt}\right)^2 + \mu_1\frac{d\theta}{dt}\right) + \frac{g}{L}\sin(\theta) = 0 \tag{8}$$

Besides the gravity and resistance, an external force may interfere with the pendulum motion, which creates a non-harmonic oscillator. The external force $f(\theta)$ can be a motor or wind that varies based on the pendulum's radiant. The differential equation (8) is expanded to become a non-homogeneous differential equation (9) including the external force in Eq. (10).

$$\frac{d^2\theta}{dt^2} + \left(\mu_2\left(\frac{d\theta}{dt}\right)^2 + \mu_1\frac{d\theta}{dt}\right) + \frac{g}{L}\sin(\theta) = f(\theta) \tag{9}$$

and

$$f(\theta) = \frac{6}{mL^2}\cos(\theta) \tag{10}$$

where m is the mass of pendulum bob and f is the external force driven by a wind or a motor. In Sect. 3.3, we assume that the external force $f(\theta)$ is independent of time. However, it could be time-dependent as $f(t, \theta)$, as the example detailed in Sect. 3.4, or even stochastic.

3.3 Simulated Pendulum

It is challenging or impossible to analytically solve the nonlinear dynamics equation since it is tough to simplify or divide-and-conquer the problem. Fortunately, we can solve the problem approximately using the numerical method using the finite difference method (FDM), the finite element method (FEM), or the finite volume method (FVM).

These differential equations described in Sect. 3.2 can be solved using the ODE numerical solvers implemented in SciPy or Julia. There are many numerical algorithms for these ODE solvers to choose to solve these equations that simulate the temporal behavior of the pendulum dynamics concerning the pendulum's angle (θ) and its angular velocity (ω) during a time frame.

The work uses the Julia [3] programming environment and its DifferentialEQuations.jl package [28] to solve these ODEs for pendulum simulations. Julia provides a high-level programming interface similar to Matlab/Python with salable performance on both CPUs and GPUs. It includes a rich set of computational packages such as differential equations of ODEs/PDEs, linear algebra,

optimizations, automatic differentiation, dynamical systems, and data science packages such as its machine learning package Flux and Boltzmann Machines.

The Julia code that defines the pendulum ODE and initial values is listed in Fig. 3. The ODE solver uses the Tsit5 algorithm - the Tsitouras 5/4 Runge-Kutta method with the free fourth-order interpolant, which is efficient and accurate in solving the pendulum ODE equation. The code defines a non-homogeneous differential equation with an external force and polynomial friction. The initial θ value is $\pi/2$ and the velocity ω is 0. The period is set from 0 to 10 s, with 0.1 s as the time step. It generates 101 samples of (θ, ω) after the calculation.

```
θ₀ = pi/2                                  # initial angular deflection [rad]
ω₀ = 0.0                                   # initial angular velocity [rad/s]
u₀ = [θ₀, ω₀]                              # initial state vector
tspan = (0.0,10.0)                         # time range
Δt = 0.1                                   # time interval
M = θ->2.0*cos(θ)                          # external torque [Nm]

function pendulum!(du,u,p,t)
  du[1] = u[2]                          # θ'(t) = ω(t)
  du[2] = poly_friction(du[1]) -(g/L)*sin(u[1]) + 3/(m*L^2)*p(u[1])
  # ω'(t) = friction(ω(t)) -g/(L)sin θ(t) + 3/(ml^2)M(θ(t))
end

prob = ODEProblem(pendulum!,u₀,tspan,M)
sol = solve(prob, Tsit5(), saveat=Δt)
```

Fig. 3. Pendulum motion ODE code

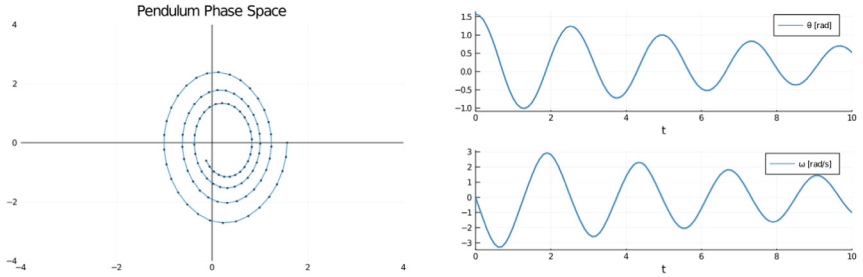
Figure 4(a) shows the phase space of the pendulum dynamics based on the θ and the ω for 10 s motion with nonlinear friction and external torque. Figure 4b illustrates the pendulum angles θ and the angular velocity ω temporal changes during the 10 s of pendulum motion simulation. Due to the external torque and friction, the motion is non-harmonic, leaning toward its right-hand side gradually.

3.4 Simulation of Wind Forced Pendulum

In this simulation a quick air flow is used to start the oscillations of pendulum. The goal of the experiment is to infer the time profile of the air flow pulse from the simulated measurements of the angular position of the pendulum (Fig. 5).

We assume that the drag force that acts on the pendulum is proportional to the relative velocity of the pendulum with respect to the air, according to Stoke's law:

$$\vec{F}_d = b(\vec{u} - \vec{v})$$



(a) Pendulum Phase Space in 10 Seconds (b) Pendulum Motion Simulation in 10 Seconds

Fig. 4. Pendulum motion simulation shown in the phase space

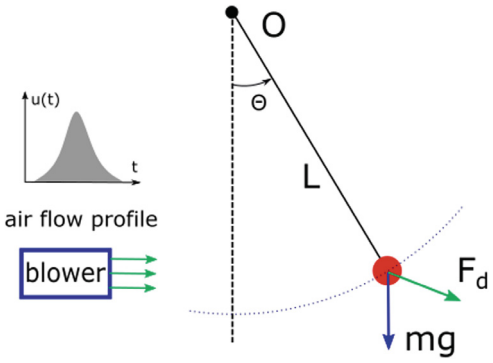


Fig. 5. A pendulum forced to oscillate by a quick air blow of wind

where the drag coefficient is $b = 6\pi\eta r$ for a spherical object of radius r, and η is the viscosity. The air flow is assumed to be uniform, and oriented along the x-axis $\boldsymbol{u} = u(t)\hat{x}$. Under these assumptions, the differential equation for the pendulum is

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\sin\theta - \frac{b}{m}\frac{d\theta}{dt} + \frac{b}{mL}u(t)\cos\theta \tag{11}$$

with initial conditions: $\theta(0) = d\theta/dt(0) = 0$. The solution $\theta(t)$ depends on the airflow profile $u(t)$ and the drag coefficient b as external parameters, while gravity g and length L of the pendulum are assumed to be known.

Given a set of measurements of time and angular position ($\{t_k, \theta_k\}$, $k = 1, 2, \ldots, N$), the unknown airflow profile, as well as the drag coefficient, can be inferred by minimizing the loss function

$$L(u(t), b) = \frac{1}{2}\sum_{k=1}^{N}\left(\theta(t_k) - \theta_k\right)^2$$

This can be obtained by using a Conjugate Gradient Descent method where better candidates for b and $u(t)$ are calculated at each iteration as

$$b \to b' = b - \eta \frac{\partial L}{\partial b}, \qquad u(t_k) \to u'(t_k) = u(t_k) - \eta \frac{\partial L}{\partial u(t_k)}$$

where η is a small learning rate chosen appropriately and the airflow profile is discretized at the same temporal points t_k, for convenience. The partial derivatives of L with respect to b and $u(t_k)$ are obtained in turn as

$$\frac{\partial L}{\partial b} = \sum_{k=1}^{N} (\theta(t_k) - \theta_k) \frac{\partial \theta}{\partial b}(t_k)$$

and

$$\frac{\partial L}{\partial u(t_k)} = \sum_{k=1}^{N} (\theta(t_k) - \theta_k) \frac{\partial \theta}{\partial u(t_k)}(t_k)$$

The sensitivities of the ODE solution $\partial\theta/\partial b$ and $\partial\theta/\partial u(t_k)$ can be calculated in several ways [25]. For our example, we used forward differentiation package ForwardDiff [30], that employs dual numbers [12] during the iterative calculation of the solution of Eq. (11). Each time step during the iterative calculation is calculated according to Heun's modification of Euler's method [9]. At the start of integration all parameters are set as dual numbers with zero dual part, except the parameter for which the sensitivity is required, which is set with 1. The solution obtained at the grid points t_k will in turn be dual numbers that represent the solution, as the main part, and the sensitivity of the solution with respect to the chosen parameter, as its dual part. The advantage of this approach is that all calculations are done in place with modest memory requirements.

Figure 6 show the results of our experiment. We simulated a 1.0 kg pendulum of length 1.0 m that has a drag coefficient of $b = 0.25$ kg/s. The pendulum is forced to oscillate by a short gaussian blow pulse of amplitude 4.0 m/s, centered around $t = 2.8$, with standard deviation of 0.2 s. Starting from random guesses for b and $u(t)$, the procedure converges toward the anticipated values. At every time step, only positive values of $u(t)$ are allowed. The convergence is slow, but it can be accelerated by using more refined strategies, like ADAM or RMSProp [15].

3.5 Physical Experimental Pendulum

Besides the simulation, we also recorded a one-minute video for the pendulum experiment shown in Fig. 7(a). In the experiment, we measured the mass of the pendulum bob and the length of the pendulum. The angle θ and angular velocity ω were calculated based on the image processing algorithms. The friction is unknown in the experiment.

To process the experiment video, we first extract the frames out of the video that is recorded with the frame rate of 60 per seconds. We then apply the Blob detection algorithm from the Scikit-image image processing package,

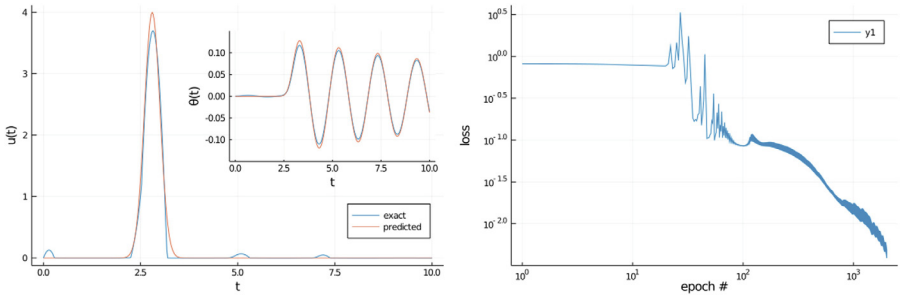
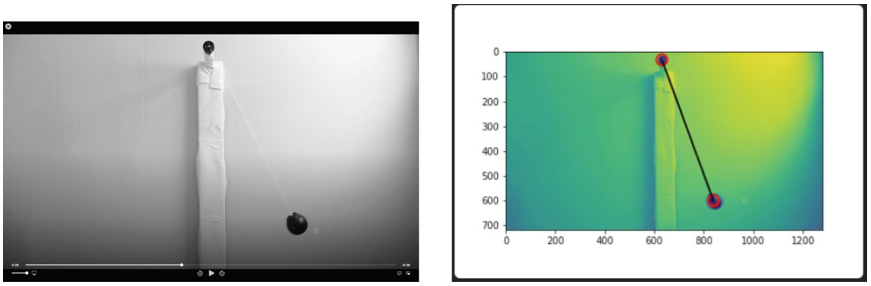


Fig. 6. Left panel: comparison between the exact and calculated airflow profiles, and angle vs. time (inset). Right panel: convergence of the loss function.



(a) Experimental Pendulum Video

(b) Labeled Experimental Pendulum Video

Fig. 7. Pendulum experiment recorded in video

which detects the coordinates of the pendulum bob and center. The Difference of Hessian (DoH) algorithm in Blob detection gives us the best performance and less false positives. Figure 7(b) shows the detected coordinates of pendulum center and bob. These coordinates detected are used to compute the angle θ and angular velocity ω based on the geometry and the prior state. The results are a collection of 3600 pendulum angles and angular velocity states in one minute with 1/60 s for each time step.

4 Learning the Nonlinear Dynamics with Scientific Machine Learning

In Sect. 3, the paper shows the simulation results pendulum nonlinear dynamics with assumptions of known functions of the friction and external torque. Can SciML augment the scientific machine learning by using the collected data set?

For real-world experiment, we may not know some of these functions, but we can collect the motion data (θ and ω) based on experiments Sect. 3.5. The question is if the SciML model can learn the unknown nonlinear dynamics hidden in these systems?

4.1 What Do These SciML Models Learn?

In the pendulum study, we knew that the simple harmonic pendulum's motion is governed by the Eq. (1). The initial conditions include the pendulum angle, the angular velocity, the length, the mass, and the constant gravity. All of them can be measured to determine the motion. In reality, what we do not know at the beginning is the friction and external force in the Eq. (9) and (10). The SciML model only models the friction and external force functions using a neural network and learns the two functions through the recorded data. The ODE solver calculates the harmonic motion.

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\sin(\theta) = N_p(\frac{d\theta}{dt}, m, L) \tag{12}$$

The Eq. (9) and (10) is revised as the new Eq. (12), in which the $N_p(\frac{d\theta}{dt}, m, L)$ is a neural network with four inputs and one output that learns the friction and external torque. The N_p is a four-layer fully connected neural network with 4-64-64-1 neurons in each layer, and it uses the hyperbolic tangent *tanh* as its activation function.

The software package used in the paper is one of the SciML packages named DiffEqFlux implemented in Julia software stack. The neural network is trained by using a small data set from the pendulum simulation with a time span of [0, 10] and a time step 0.1, which gives us 101 samples. The training starts with the Adam optimizer for the first 100 iterations and then switch to the BFGS optimizer after the 100th iteration. Figure 8 shows the loss values of using these two optimizers. In this experiment, the BFGS optimizer learns the function faster than Adam optimizer.

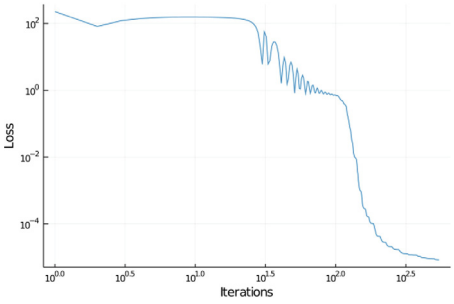
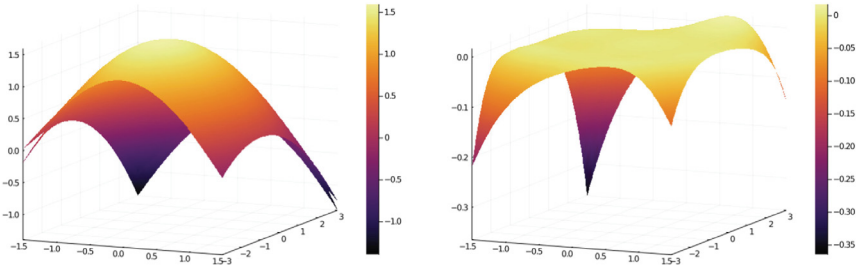


Fig. 8. Loss values during the UDE training

Once the loss value becomes small enough ($<10^{-4}$), the neural network approximates the friction and torque functions well. It is interesting to understand what the neural network learned and compares it with the ground-truth friction and torque functions. Figure 9a shows the overlay graph of the neural network and the ground-truth friction and torque functions. The figure shows that the neural network and ground-truth functions are close enough at most input spaces. Some minor differences are showing in the boundary areas of the input space. Figure 9b illustrates the differences between these two functions.



(a) The overlay plot of Neural Network and ground-truth

(b) Differences between Neural Network and ground-truth

Fig. 9. A comparison of the trained neural network and the ground-truth function

For the experiment pendulum in Sect. 3.5, the friction is unknown. The SciML UDE models the friction as part of the differential equation of pendulum dynamics. After training to fit with the collected coordinates and angular velocity from the video, the SciML UDE discovers the friction exerted in the pendulum shown in Fig. 10. Note that the SciML UDE may produce multiple solutions of the friction that fits the observed data well. We believe that the friction should has the property of odd symmetry where $f(x) = -f(-x)$. We selected Fig. 10 as the correct solution.

4.2 Can SciML Predict the Future?

In this experiment, the neural network training data was generated using a time interval of 0.1 s, which is sampled with a low frequency and generates sparse data points. Can the neural network be trained using the small data set and generalize well to predict the high-frequency data? It is a question about the capacity of the neural network's interpolation.

The paper answers the question by testing the SciML solution for the same period [0, 10] but reduce the time interval by ten times to 0.01. The result shows

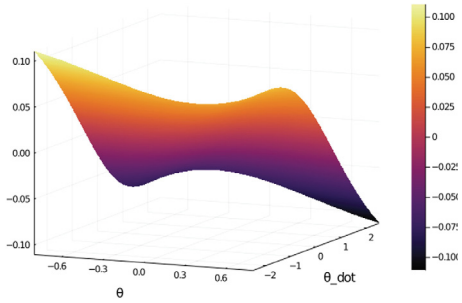
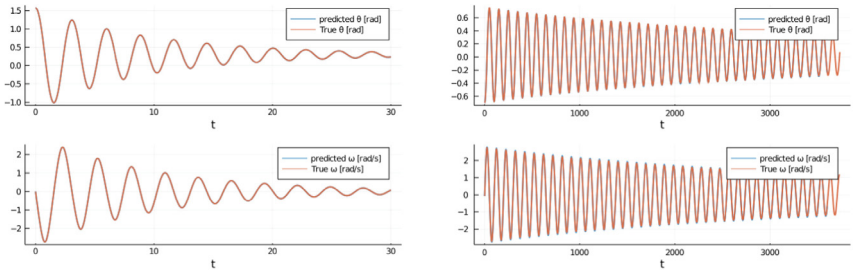


Fig. 10. The friction learned by sciml in the experiment pendulum

that the SciML solution with the trained neural network can predict the high-frequency data with a time step of 0.01. The interpolation and generalization capacity is well preserved in the SciML solution.



(a) a (b) b

Fig. 11. (a) Prediction of the simulated pendulum motion for 30 s (b) Prediction of the experiment pendulum motion for 60 s

One weakness of machine learning is that it does not perform well on extrapolation. In particular, if the problem is not periodic, it is very challenging or even impossible to predict the future without gaining the full data covering the entire problem space. By using the SciML methodology, the extrapolation is feasible since the neural network is embedded in the ODE models as a time-independent function. The ODE solver calculates all time-related dynamics. The combination of ODE and neural network works well on extrapolation in predicting the nonlinear pendulum dynamics.

Figure 11a shows the results of the simulated pendulum motion prediction for 30 s with a time interval of 0.01. The neural network was trained using 10 s of data with a time interval of 0.1. The figure shows a complete overlay of the prediction and the ground-truth data, which indicates that the prediction matches the real data very well. Note that the prediction is performed to generate much higher-

frequency data than the training data, which also proves that both interpolation and extrapolation work well using SciML.

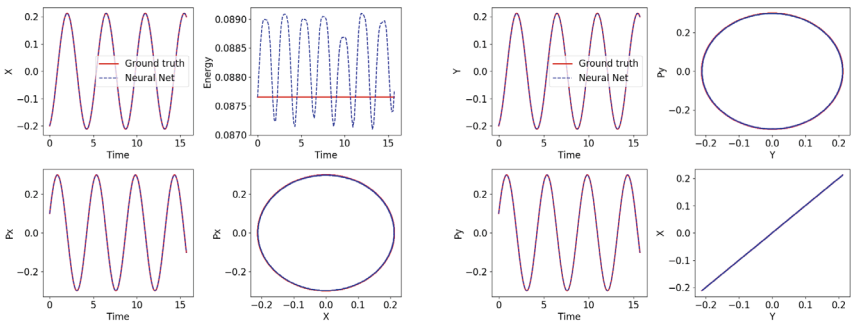
Figure 11b shows the results of the experiment pendulum motion prediction for one minute with a time interval of 1/60. The neural work trained to learn the friction and the results fit well with the observed data. Note that the figure actually shows both observed and predicted θ and ω, which overlap too well to distinguish them.

4.3 Can HNN Solve Complex Dynamic Problems?

The both simple harmonic motion and the non-linear dynamic motion of the quadruple-springs-mass system (Fig. 1) are also simulated using HNN method described by Mattheakis *et al.* [21]. We used four parametric solutions in the form of $\hat{z}(t) = z(0) + f(t)(N(t) - z(0)$ to solve for the four generalized coordinates [24]. We choose a parametric function $f(t) = 1 - exp(-t)$ and a symmetric activation function of *sin* following Mattheakis *et al.* [21]. We used six layer neural network with ten neurons for the HNN. We also used a learning rate of 0.0001.

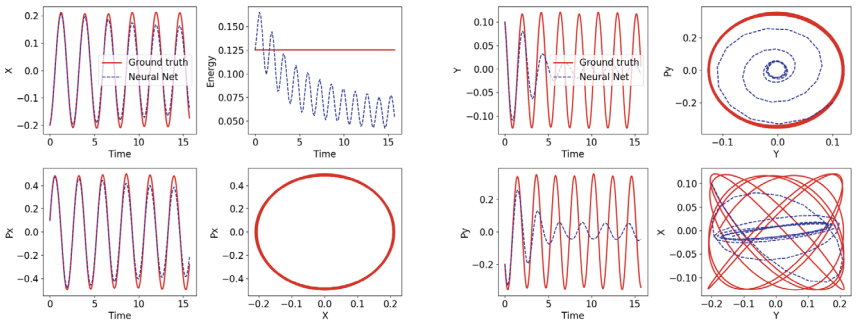
The HNN based simulation of the simple harmonic motion agrees well with the ground truth trajectories as shown in the Fig. 12(a) and 12(b). But, the energy does not seems to be conserved. However, the simulation of the non-linear dynamic system using HNN fails to produce the desired results (Fig. 13(a), 13(b)). The loss functions for the both systems are shown in the Fig. 14(a) and 14(b). The loss value of the simple harmonic motion is much lower than the non-linear dynamic system.

The HNN method requires to incorporate more physical constraints such as symmetry [20] to the neural network to improve the simulations of complex non-linear motions.



(a) Motion in the x-direction. (b) Motion in the y-direction.

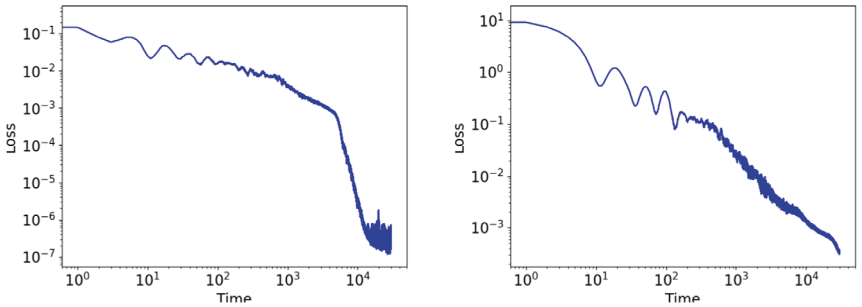
Fig. 12. A simulation of the simple harmonic motion of the quadruple-springs-mass system using the HNN method.



(a) Motion in the x-direction.

(b) Motion in the y-direction.

Fig. 13. A simulation of the non-linear dynamic motion of the quadruple-springs-mass system using HNN method.



(a) Loss function for the simple harmonic motion.

(b) Loss function for the non-linear dynamic system.

Fig. 14. The loss functions for the HNN method

5 Conclusion

SciML is a new method that combines the scientific principle-based solutions and the data-driven machine learning models, which creates a new set of tools for scientists to understand the scientific rules using existing theories and learning from data directly. The combination equips scientists with additional flexibility in modeling a scientific problem with partially known knowledge and limited data sets. The requirement of big data for machine learning may be relaxed to learn a scientific phenomenon since only the unknown portion is approximated with machine learning, while other portions are still governed by the existing deterministic scientific principles, which dramatically reduces the training complexity in machine learning.

References

1. Baker, N., et al.: Workshop report on basic research needs for scientific machine learning: core technologies for artificial intelligence, February 2019
2. Baydin, A.G., Pearlmutter, B.A., Radul, A.A., Siskind, J.M.: Automatic differentiation in machine learning: a survey. J. Mach. Learn. Res. **18**(1), 5595–5637 (2017)
3. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: a fresh approach to numerical computing. SIAM Rev. **59**(1), 65–98 (2017)
4. Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., Holtham, E.: Reversible architectures for arbitrarily deep residual neural networks. In: AAAI (2018)
5. Chang, B., Meng, L., Haber, E., Tung, F., Begert, D.: Multi-level residual networks from dynamical systems view. In: Conference on ICLR (2018)
6. Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural ordinary differential equations (2019)
7. Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 31, pp. 6571–6583. Curran Associates Inc. (2018)
8. Weinan, E.: A proposal on machine learning via dynamical systems. Commun. Math. Stat. **5**, 1–11 (2017). https://doi.org/10.1007/s40304-017-0103-z
9. Mayers, D.F., Süli, E.: An Introduction to Numerical Analysis. Cambridge University Press, Cambridge (2003)
10. Fries, T.-P., Matthies, H.G.: A review of Petrov-Galerkin stabilization approaches and an extension to meshfree methods (2004)
11. Greydanus, S., Dzamba, M., Yosinski, J.: Hamiltonian neural networks (2019)
12. Griewank, A., Walther, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, 2nd edn. SIAM, Philadelphia (2008)
13. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 630–645. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_38
14. Kharazmi, E., Zhang, Z., Karniadakis, G.E.: Variational physics-informed neural networks for solving partial differential equations (2019)
15. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
16. Kolter, Z., Duvenaud, D., Johnson, M.: Deep implicit layers - neural ODEs, deep equilibirum models, and beyond
17. Li, Z., Shi, Z.: Deep residual learning and PDEs on manifold. CoRR, abs/1708.05115 (2017)
18. Lu, L., Meng, X., Mao, Z., Karniadakis, G.E.: DeepXDE: a deep learning library for solving differential equations. CoRR, abs/1907.04502 (2019)
19. Lu, Y., Zhong, A., Li, Q., Dong, B.: Beyond finite layer neural networks: bridging deep architectures and numerical differential equations. In: Proceedings of the 35th International Conference on Machine Learning (2018)
20. Mattheakis, M., Protopapas, P., Sondak, D., Di Giovanni, M., Kaxiras, E.: Physical symmetries embedded in neural networks (2020)
21. Mattheakis, M., Sondak, D., Protopapas, P.: Hamiltonian neural networks for solving differential equations. Preparation
22. Mohazzabi, P., Shankar, S.P.: Damping of a simple pendulum due to drag on its string. J. Appl. Math. Phys. **05**(01), 122–130 (2017)

23. Pang, G., Lu, L., Karniadakis, G.E.: fPINNs: fractional physics-informed neural networks. SIAM J. Sci. Comput. **41**(4), A2603–A2626 (2019)
24. Paticchio, A., Scarlatti, T., Mattheakis, M., Protopapas, P., Brambilla, M.: Semi-supervised Neural Networks solve an inverse problem for modeling Covid-19 spread (2020)
25. Petzold, L., Li, S., Cao, Y., Serban, R.: Sensitivity analysis of differential-algebraic equations and partial differential equations. Comput. Chem. Eng. **30**(10), 1553–1559 (2006)
26. Pontryagin, L.S., Mishchenko, E.F., Boltyanskii, V.G., Gamkrelidze, R.V.: The Mathematical Theory of Optimal Processes. Wiley, New York (1962)
27. Rackauckas, C., et al.: Universal differential equations for scientific machine learning. arXiv preprint arXiv:2001.04385 (2020)
28. Rackauckas, C., Nie, Q.: Differentialequations.jl-a performant and feature-rich ecosystem for solving differential equations in Julia. J. Open Res. Softw. **5**(1) (2017)
29. 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. **378**, 686–707 (2019)
30. Revels, J., Lubin, M., Papamarkou, T.: Forward-mode automatic differentiation in Julia. arXiv:1607.07892 [cs.MS] (2016)
31. Shin, Y., Darbon, J., Karniadakis, G.E.: On the convergence and generalization of physics informed neural networks (2020)
32. Sonoda, S., Murata, N.: Double continuum limit of deep neural networks. In: ICML Workshop Principled Approaches to Deep Learning (2017)
33. Toth, P., Rezende, D.J., Jaegle, A., Racanière, S., Botev, A., Higgins, I.: Hamiltonian generative networks (2020)
34. Yang, L., Zhang, D., Karniadakis, G.E.: Physics-informed generative adversarial networks for stochastic differential equations. SIAM J. Sci. Comput. **42**, A292–A317 (2020)
35. Zhang, D., Lu, L., Guo, L., Karniadakis, G.E.: Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. J. Comput. Phys. **397**, 108850 (2019)
36. Zhang, X., Li, Z., Change Loy, C., Lin, D.: PolyNet: a pursuit of structural diversity in very deep networks. CoRR, abs/1611.05725 (2016)