On Neural Differential Equation Report

Neural Differential Equations (NDEs) are suitable for tackling generative problems, dynamical systems, and time series (particularly in physics, finance, …) and are thus of interest to both modern machine learning and traditional mathematical modelling. NDEs offer high-capacity function approximation, strong priors on model space, the ability to handle irregular data, memory efficiency, and a wealth of available theory on both sides.

Topics include: *neural ordinary differential equations* (e.g. for hybrid neural/mechanistic modelling of physical systems); *neural controlled differential equations* (e.g. for learning functions of irregular time series); and *neural stochastic differential equations* (e.g. to produce generative models capable of representing complex stochastic dynamics, or sampling from complex high-dimensional distributions).

1. *What is a neural differential equation?*

A neural differential equation is a differential equation using a neural network to parameterise the vector field. The canonical example is a neural ordinary differential equation [1]:

$$y(0) = y_0 \qquad \frac{dy}{dt}(t) = f_\theta(t, y(t))$$

Here θ represents some vector of learnt parameters, $f_\theta: \mathbb{R} \times \mathbb{R}^{d_1 \times \ldots \times d_k} \to \mathbb{R}^{d_1 \times \ldots \times d_k}$ is any standard neural architecture, and $y: [0, T] \to \mathbb{R}^{d_1 \times \ldots \times d_k}$ is the solution. For many applications f_θ will just be a simple feedforward network.

The central idea now is to use a differential equation solver as part of a learnt differentiable computation graph (the sort of computation graph ubiquitous to deep learning).
As a simple example, suppose we observe some picture $y_0 \in \mathbb{R}^{3 \times 32 \times 32}$ (RGB and 32x32 pixels), and wish to classify it as a picture of a cat or as a picture of a dog.
We proceed by taking $y(0) = y_0$ as the initial condition of the neural ODE and evolve the ODE util some time T. An affine transformation $l_\theta: \mathbb{R}^{d_1 \times \ldots \times d_k} \to \mathbb{R}^2$ is then applied followed by a softmax, so that the output may be interpreted as a length-2 tuple (**P**(picture is of a cat), **P**(picture is of a dog)). This is summarized in Fig 1.

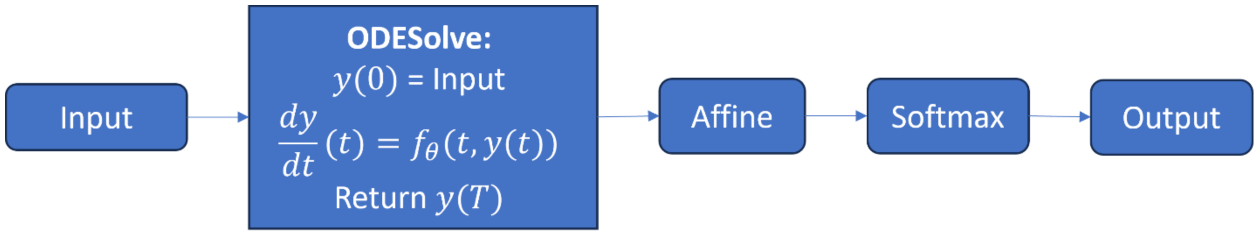


Figure 1 General pipeline for applying Neural ODE

In conventional mathematical notation, this computation may be denoted

$$softmax\left(l_\theta\left(y(0) + \int_0^T f_\theta(t, y(t))dt\right)\right).$$

The parameters of the model are θ. The computation graph may be backpropagated through and trained via stochastic gradient descent in the usual way.

1.1 Continuous-depth neural networks

Recall the formulation of a residual network [2]:

$$y_{j+1} = y_j + f_\theta(j, y_j) \qquad (1.1)$$

Where $f_\theta(j, \cdot)$ is the j-th residual block. (the parameters of all blocks are concatenated together into θ).

Given the neural ODE

$$\frac{dy}{dt}(t) = f_\theta(t, y(t))$$

Discretising this via the explicit Euler method at times t_j uniformly separated by Δt gives

$$\frac{y(t_{j+1}) - y(t_j)}{\Delta t} \approx \frac{dy}{dt}(t_j) = f_\theta\left(t_j, y(t_j)\right),$$

So that

$$y(t_{j+1}) = y(t_j) + \Delta t f_\theta\left(t_j, y(t_j)\right),$$

Absorbing the Δt into the f_θ, we recover the formulation of equation (1.1)

Having made this observation – that neural ODE are the continuous limit of residual networks. It transpires that the key features of a GRU [3] or an LSTM [4], over generic recurrent networks, are updates rules that look suspiciously like discretised differential equations.

1.2 An important distinction to PINN

There has been a line of work on obtaining numerical approximations to the solution y of an ODE

$$\frac{dy}{dt}(t) = f(t, y(t))$$

by representing the solution as some neural network $y = y_\theta$. Perhaps f is known, and the model y_θ is fitted by minimizing a loss function of the form

$$\min_\theta \frac{1}{N}\sum_{i=1}^{N}\left\|\frac{\mathrm{d}y_\theta}{\mathrm{d}t}(t_i) - f(t_i, y_\theta(t_i))\right\| \qquad (1.2)$$

for some points $t_i \in [0, T]$. As such each solution to the differential equation is obtained by solving an optimization problem. This is known as a physics-informed neural network (PINN). PINNs are effective when generalized to some PDEs, in particular nonlocal or high-dimensional PDEs, for which traditional solvers are computationally expensive. (Although in most regimes traditional solvers are still the more efficient choice.) [5] provide an overview. However, we emphasize that this is a distinct notion to neural differential equations. NDEs use

neural networks to specify differential equations. Equation (1.2) uses neural networks to obtain solutions to prespecified differential equations. This distinction is
a common point of confusion, especially as the PDE equivalent of (1.2) is sometimes referred to as a “neural partial differential equation”.
In other words, in Neural ODE, a neural network f_θ is used as surrogate model to learn the derivative corresponding to an ODE.

Example:

A damped nonlinear pendulum governed by an ODE:

$$\frac{d}{dt}\begin{bmatrix}\theta\\ \omega\end{bmatrix} = \begin{bmatrix}\omega\\ -(\mu\omega+\frac{g}{L}sin\theta)\end{bmatrix}, \begin{bmatrix}\theta(0)\\ \omega(0)\end{bmatrix} = \begin{bmatrix}\theta_0\\ \omega_0\end{bmatrix} \quad (2)$$

The trajectory of equation (2)

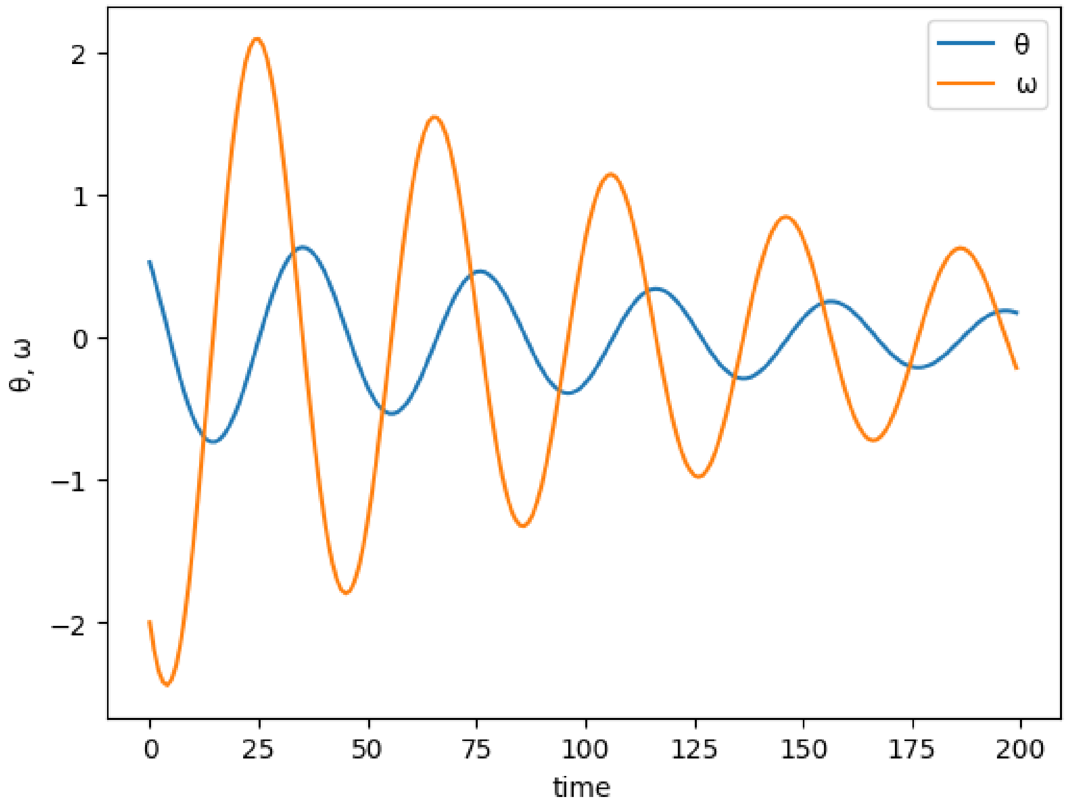


Figure 2 Generated trajectories using governing equation

Given time t $\in [0, 10]$, $\Delta t = 0.05$, the trajectory contains 200 data points (θ_t, ω_t)
Initial condition: (θ_0, ω_0) = (0.5235988, -2.), (μ, g, L) = (0.3, 9.81, 1.0),
Numerical solver: Euler.

With $f(t, \theta, \omega) = \begin{bmatrix}\omega\\ -(\mu\omega+\frac{g}{L}sin\theta)\end{bmatrix}$

At trajectory (n+1)-th: $y_{n+1} = y_n + \mathrm{f}(t_n, \theta_n, \omega_n)$

Now, we use a MLP model f_θ as surrogate model of $\mathrm{f}(\mathrm{t}, \theta, \omega)$ to learn the derivative corresponding to equation (2); $f_\theta(\mathrm{t}, \theta, \omega) \sim \mathrm{f}(\mathrm{t}, \theta, \omega)$
Hence, at trajectory (n+1)-th: $y_{n+1} = y_n + f_\theta(t_n, \theta_n, \omega_n)$

The results are shown as below:

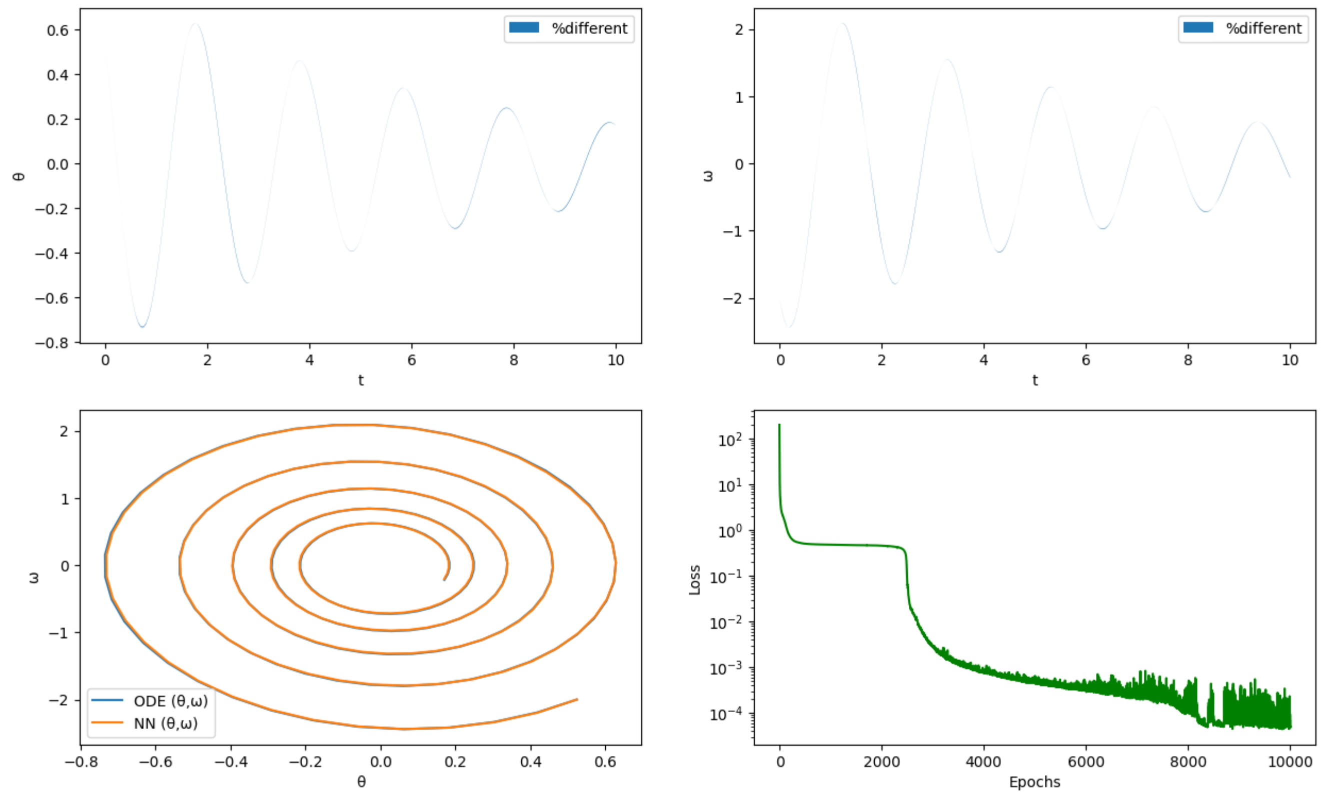


Figure 3 Difference between numerical solver's trajectories vs neural network'prediction (two figures on the first row). A plot of θ vs ω generated from both models, and the training loss on the second row.

There is a strong agreement between neural ODE model vs numerical solver’s trajectories, however, there is still a gap between these models.

2. What is a neural stochastic differential equation?
   2.1 Stochastic differential equation (SDE)
   They are natural extension of ordinary differential equations (ODEs) for modeling systems that evolve in continuous time subject to uncertainty.
   The dynamics of an SDE consist of a deterministic term and a stochastic term:

$$\mathrm{dy(t)} = \mu(\mathrm{t}, \mathrm{y(t)})\mathrm{dt} + \sigma(\mathrm{t}, \mathrm{y(t)}) \circ \mathrm{dw(t)} \qquad (3)$$

   Where

$$\mu: [0, \mathrm{T}] \times \mathbb{R}^{d_y} \to \mathbb{R}^{d_y},$$
$$\sigma: [0, \mathrm{T}] \times \mathbb{R}^{d_y} \to \mathbb{R}^{d_y \times d_w},$$

are suitably regular functions, w: $[0, \mathrm{T}] \longrightarrow \mathbb{R}^{d_w}$ is a d_w- dimensional Brownian motion, and y: $[0, \mathrm{T}] \longrightarrow \mathbb{R}^{d_y}$ is the resulting d_w- dimensional continuous stochastic process.

2.2 Neural SDE

Consider the autonomous one-dimensional Itô SDE

$$\mathrm{dy(t)} = \mu\big(t, y(t)\big)dt + \sigma(t, y(t))^{\circ}dw(t)$$

with $\mathrm{y(t)}, \mu\big(\mathrm{y(t)}\big), \sigma\big(\mathrm{y(t)}\big), \mathrm{w(t)} \in \mathbb{R}$. Then its numerical Euler-Maruyama discretization is

$$y_{j+1} = y_j + \mu\big(y_j\big)\Delta t + \sigma(y_j)\Delta w_j$$

Where Δt is some fixed time step and $\Delta w_j \sim \mathrm{N}(0, \Delta t)$

Each sample y from an SDE (3) is a continuous-time path y: $[0, \mathrm{T}] \longrightarrow \mathbb{R}^{d_y}$

Given a numerical solver: g(x), and a surrogate neural network model f(x).

f(x) can not perfectly generate a trajectory that matches g(x) output.

We have g(x, t) = f(x, t) + res(x, t); where res(x) is the residual between g and f.

The idea is to model res(x, t) that can predict the next different state.

i. Given only initial conditions

1.1 Deterministic Approach

Use a neural ODE model to model res(x, t) – cannot evaluate uncertainty

1.2 Neural Stochastic Differential Equation Approach

Q1: Why Neural SDE?

Neural SDE can be viewed as using randomness as a drop-in augmentation for Neural ODE. As the stochastic generalization of ODE, Neural SDEs [6] have been proposed by regarding intrinsic stochasticity in data representation. The objective is that we can control the impact of perturbations on the output by adding a stochastic term to neural networks.

Integrating white noise from a Wiener process is a way to incorporate randomness or uncertainty into a system's dynamics. White noise is a type of random signal that has equal intensity at different frequencies, and it is often represented by a Wiener process (Brownian motion). Integrating white noise in this context involves creating a mathematical model where the system's states are influenced by random fluctuations.

Given a stochastic differential equation:

$$dXt = \mu(t, Xt)dt + \sigma(t, Xt)dWt \qquad (1)$$

Where b and σ are the drift and diffusion functions, respectively. W_t is the Wiener process.

- $\mu(t,\, y_t) \; - drift\ term$:

+ Represents the deterministic or predictable part of the system's dynamics.

+ It describes the average or expected rate of change of the process at a given point in the state space.

+ Influences the long-term or average behavior of the system.

- $\sigma(t,\ y_t)\ - diffusion\ term$:

  + Represents the stochastic or random part of the system's dynamic.

  + It accounts for random fluctuations or noise in the system that cannot be predicted.

  + The diffusion term is multiplied by the Wiener process (dW_t) , introducing random increments into the process.

Q2: How to model res(x, t)?

- Given a well-trained Neural ODE model, but there still exists errors between Neural ODE vs numerical model as shown below:

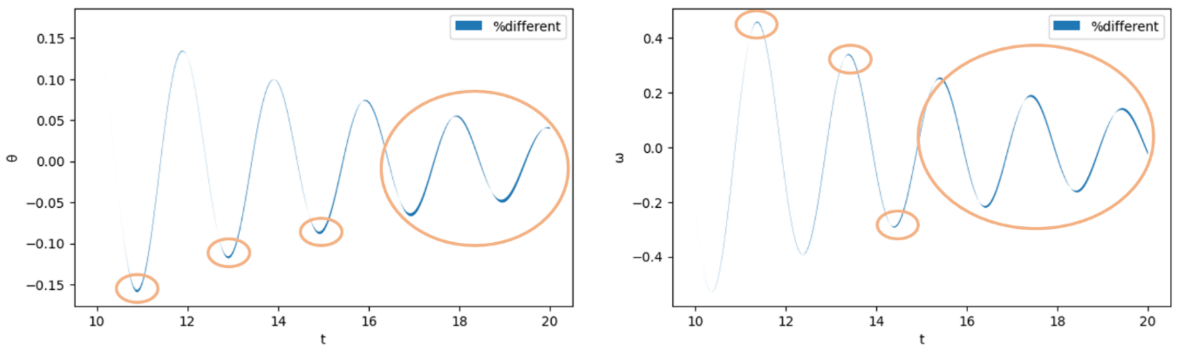


Figure 4 Residual between Neural ODE model vs numerical model

- Assume that Neural ODE (F_θ) can capture physical underlining features, the error is by unknown features. Then, the idea is to use a neural network (G_θ) to approximate the unknown features.
- With the semantic of diffusion term in Neural SDE, we can model $\sigma(t,\ y_t) =\ G_\theta$, and dW_t to introduce stochastic component to evaluate uncertainty in the prediction of a Neural SDE model (S_θ): $\mathbf{dX(t)} = \boldsymbol{F_\theta(t, X_t)dt +\ G_\theta(t, X_t)^\circ dW_t}$
- To train (S_θ): freeze $F_\theta(t, X_t)$ and only train $G_\theta(t, X_t)$.
- Loss function: $A = min_\theta||X_t - data(t)||^2$, where data(t) is generated trajectories from numerical solver.
- **Extension**: (1) given a dynamic system governed by a diffeq $\frac{dy(t)}{dt} = f(t, y(t))$, we can integrate physic constraint in the loss function by:

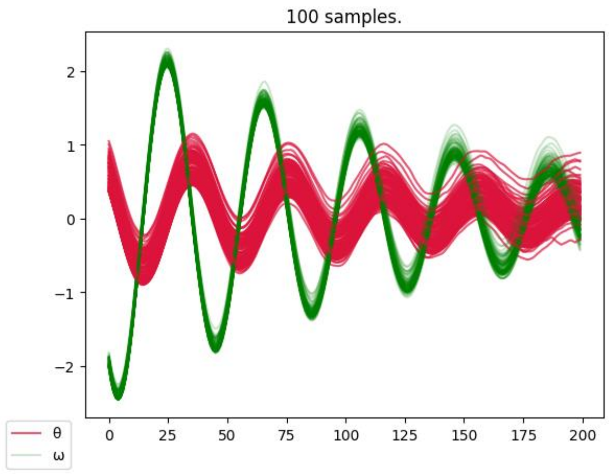
$$min_{\theta,\varphi}(||y(t) - data(t)||^2 + ||\frac{dy(t)}{dt} - f(t, y(t)), ||^2)$$

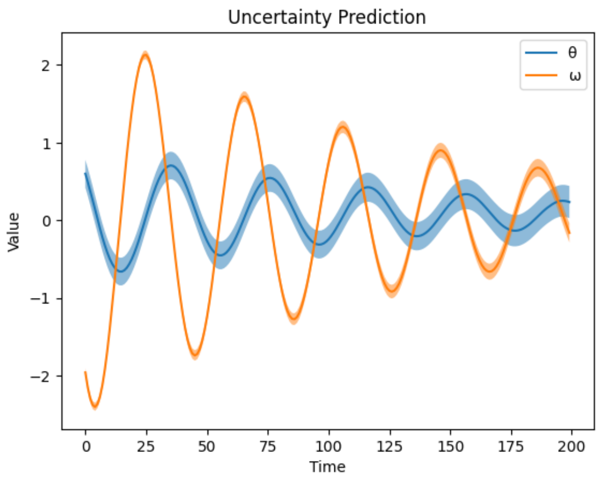
(2) Consider stochastic control;

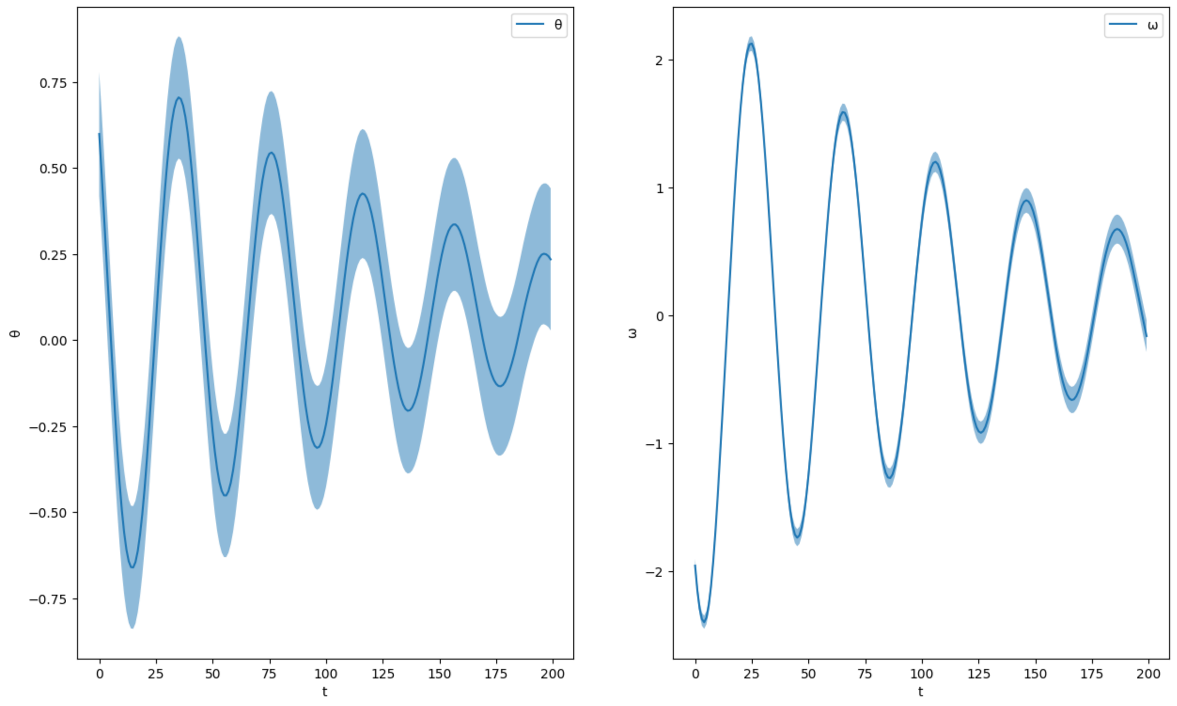
$\mathbf{dX(t)} = \boldsymbol{F_\theta(t, X_t, \pi(t))dt +\ G_\theta(t, X_t, \pi(t))^\circ dW_t,\ \ \pi}$ is the control process

(3) Fractional Brownian Motion (fBm) instead of standard Brownian Motion, to consider long-range dependency in trajectories.

RESULTS:



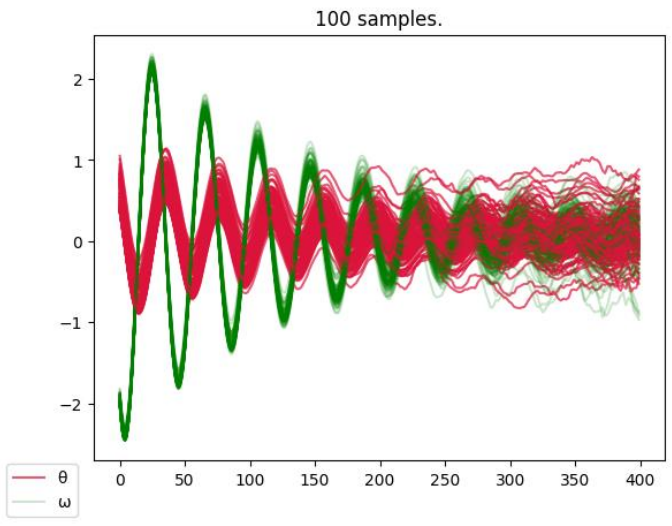




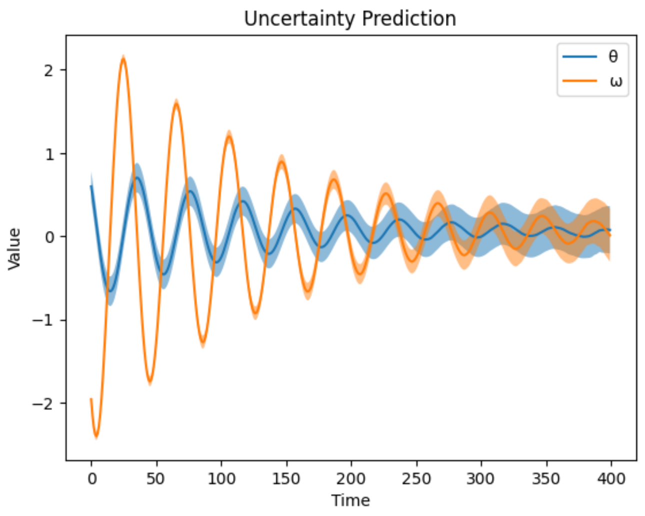
(1c)

Figure 5 (1a) sampled prediction, (1b) mean and standard deviation of predicted trajectories, (1c) separated visualization

(ts = [0, 10], Δt = 0.05)

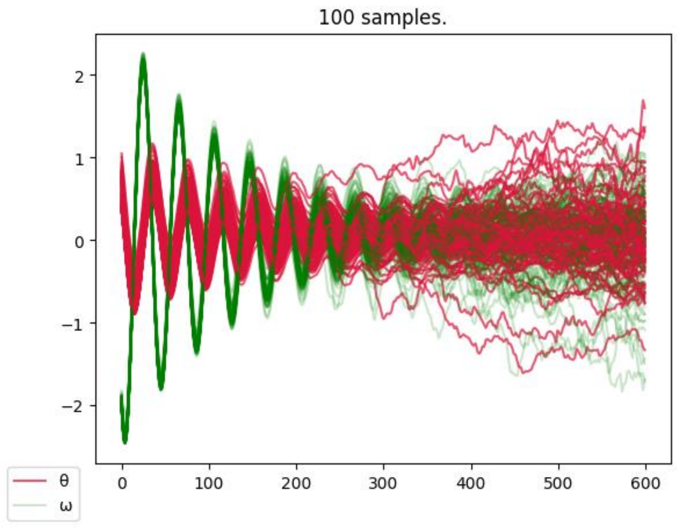


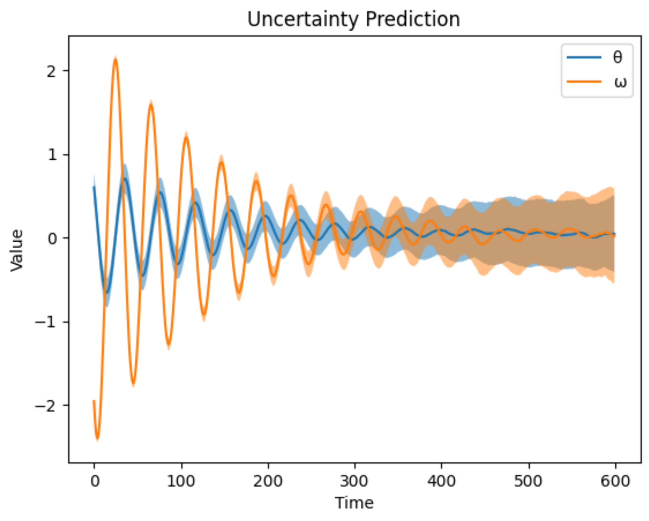
(2a)

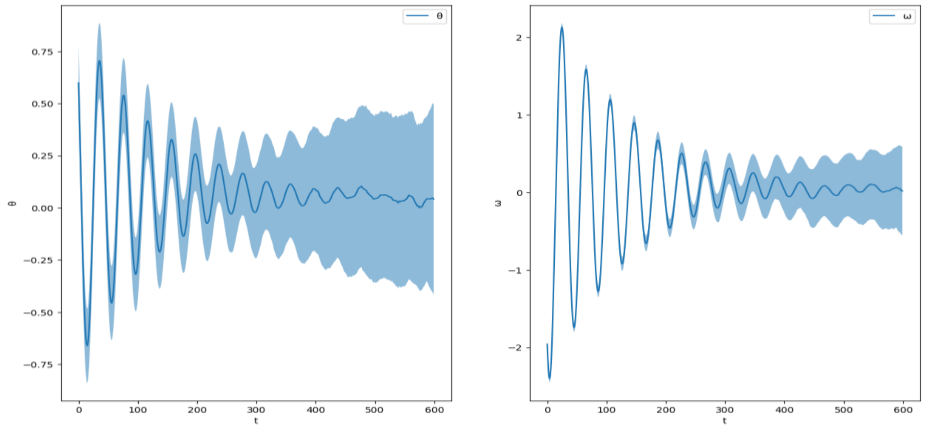


(2b)

Figure 6 (2a) sampled prediction, (2b) mean and standard deviation of predicted trajectories, (ts = [0, 20], Δt = 0.05)

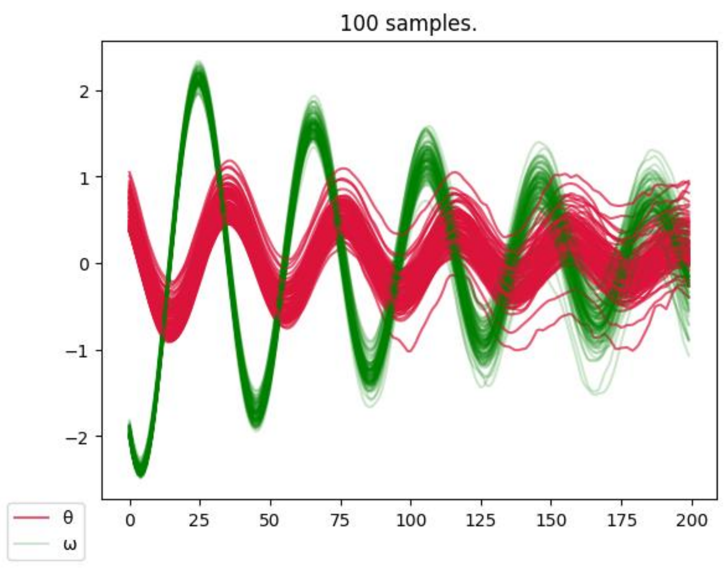




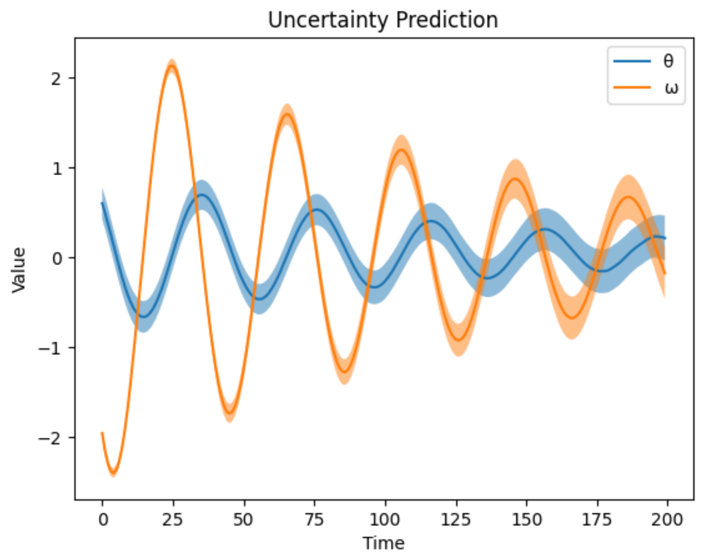


(3c)

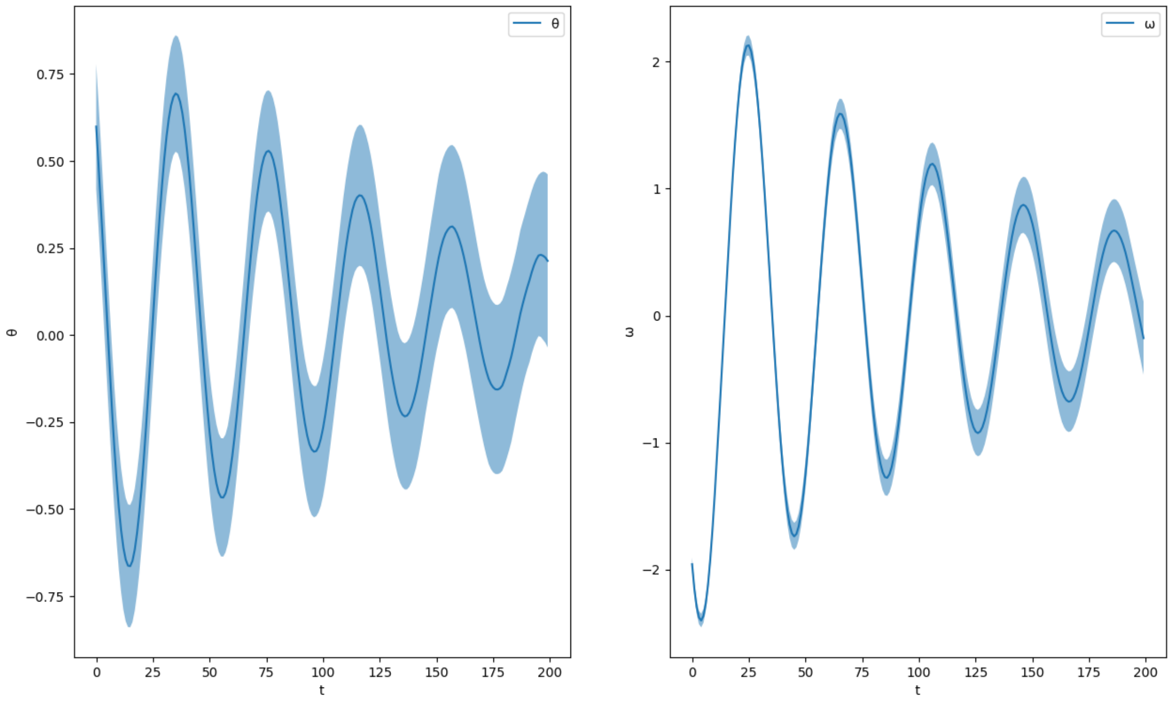
Figure 7 (3a) sampled prediction, (3b) mean and standard deviation of predicted trajectories, (3c) separated visualization



(4a)



(4b)



(4c)

Figure 8 (4a) sampled prediction, (4b) mean and standard deviation of predicted trajectories, (4c) separated visualization

(ts = [10, 20], Δt = 0.05)

Here are comparison results between neural SDE model, neural ODE model, and numerical solver.

Description of legends	
pred mean (θ, ω)	Neural SDE prediction mean with std in the same color
numerical (θ, ω)	Numerical solver's output
ode (θ, ω)	Neural ODE prediction
sde-mean vs num (θ, ω)	Difference between neural SDE prediction vs numerical solver
Ode vs num (θ, ω)	Difference between neural ODE prediction vs numerical solver
ode vs sde-mean (θ, ω)	Difference between neural SDE prediction vs neural ODE prediction

Figures are presented in the format: mean and standard deviation of predicted trajectories (2 figures above), difference comparison (2 figures below)

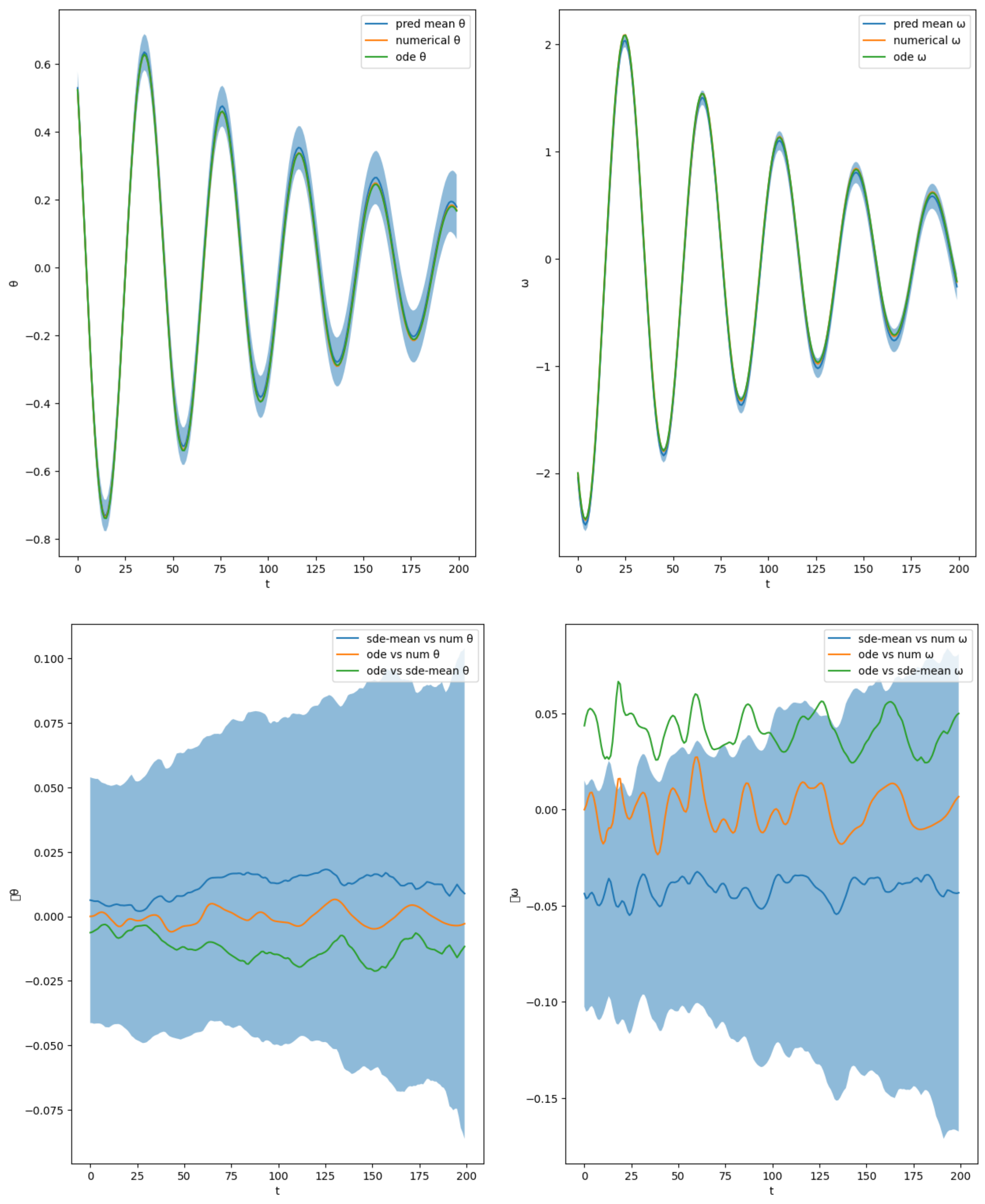


Figure 8 (ts = [0, 10], Δt = 0.05)

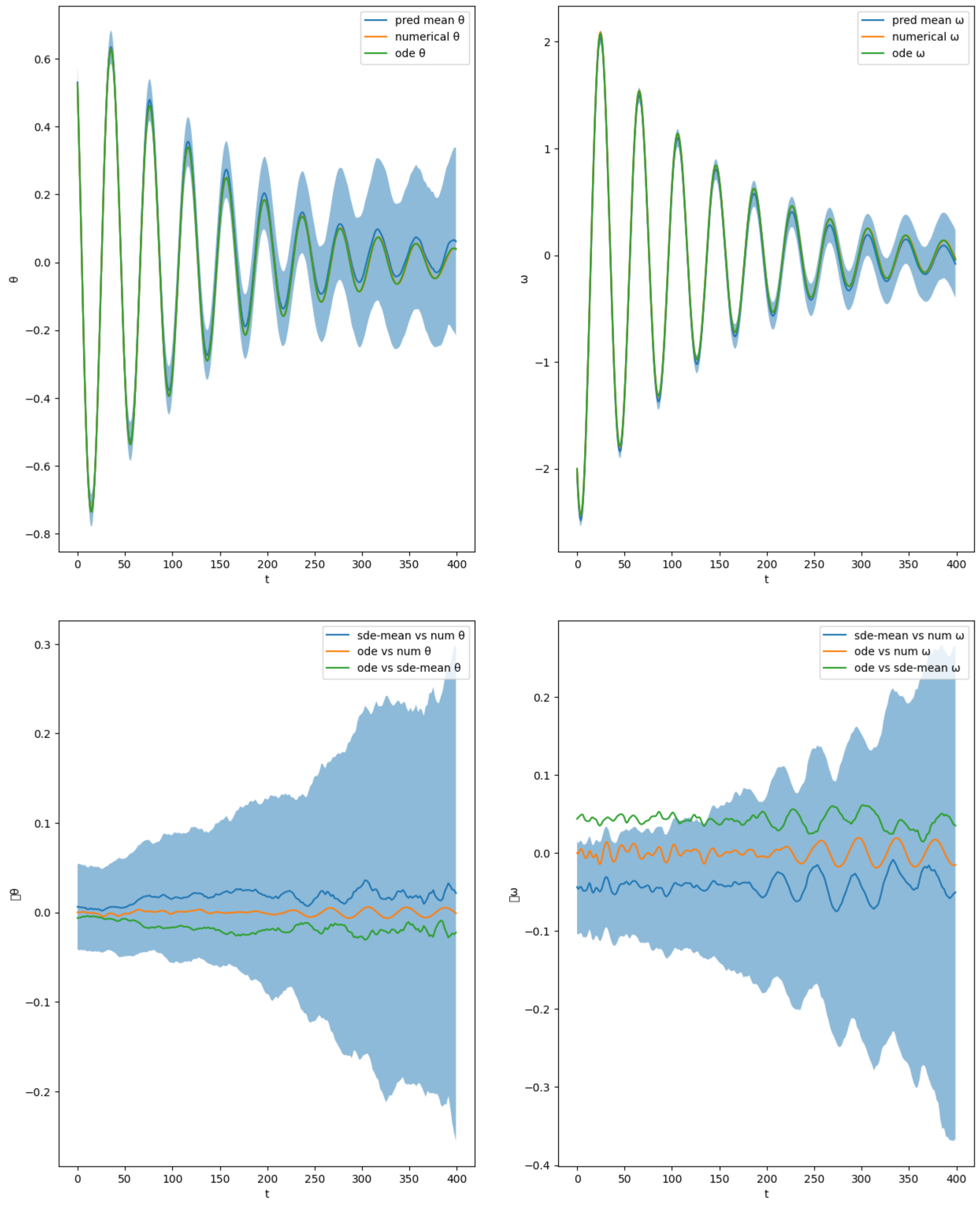


Figure 9 (ts = [0, 20], Δt = 0.05)

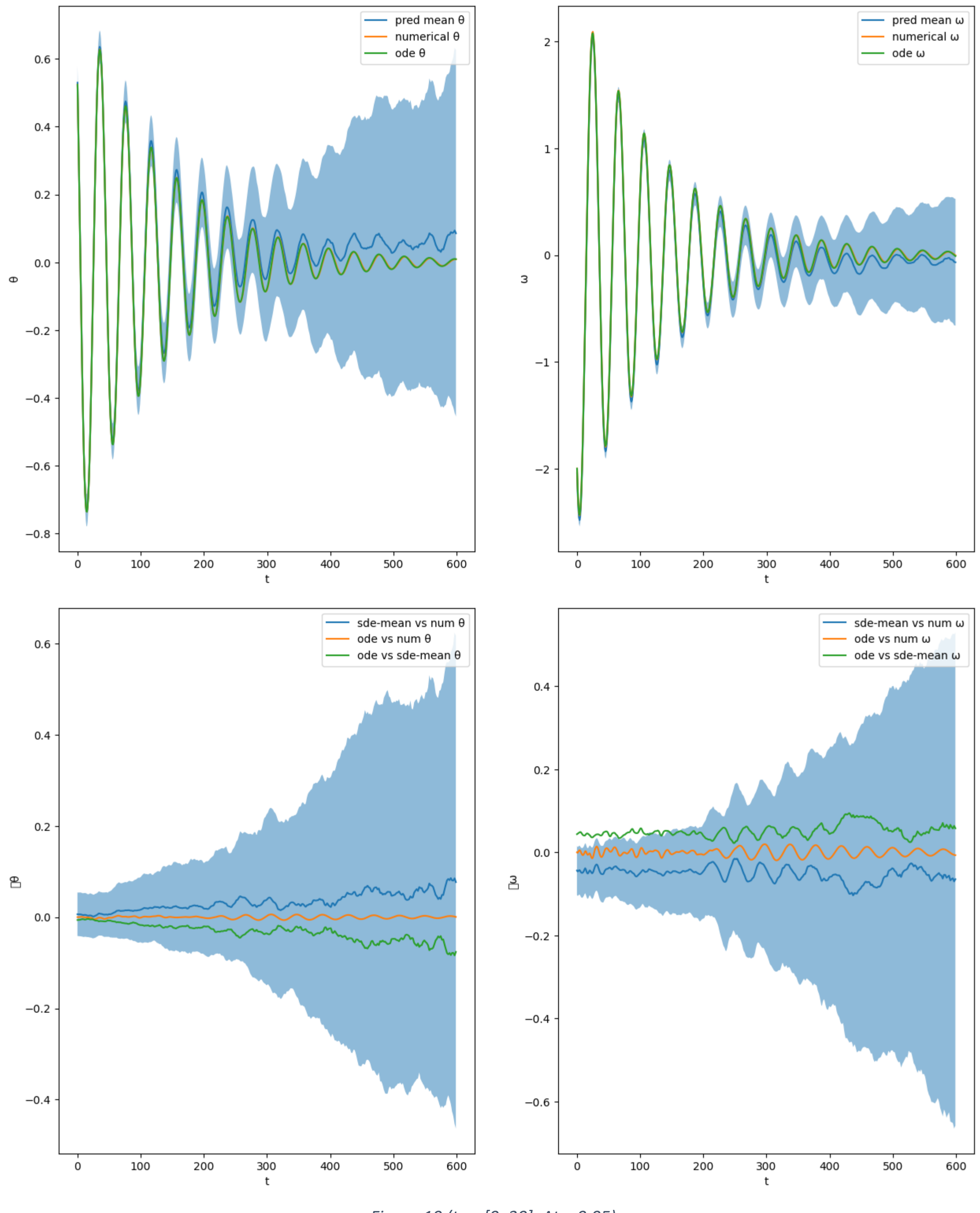


Figure 10 (ts = [0, 30], Δt = 0.05)

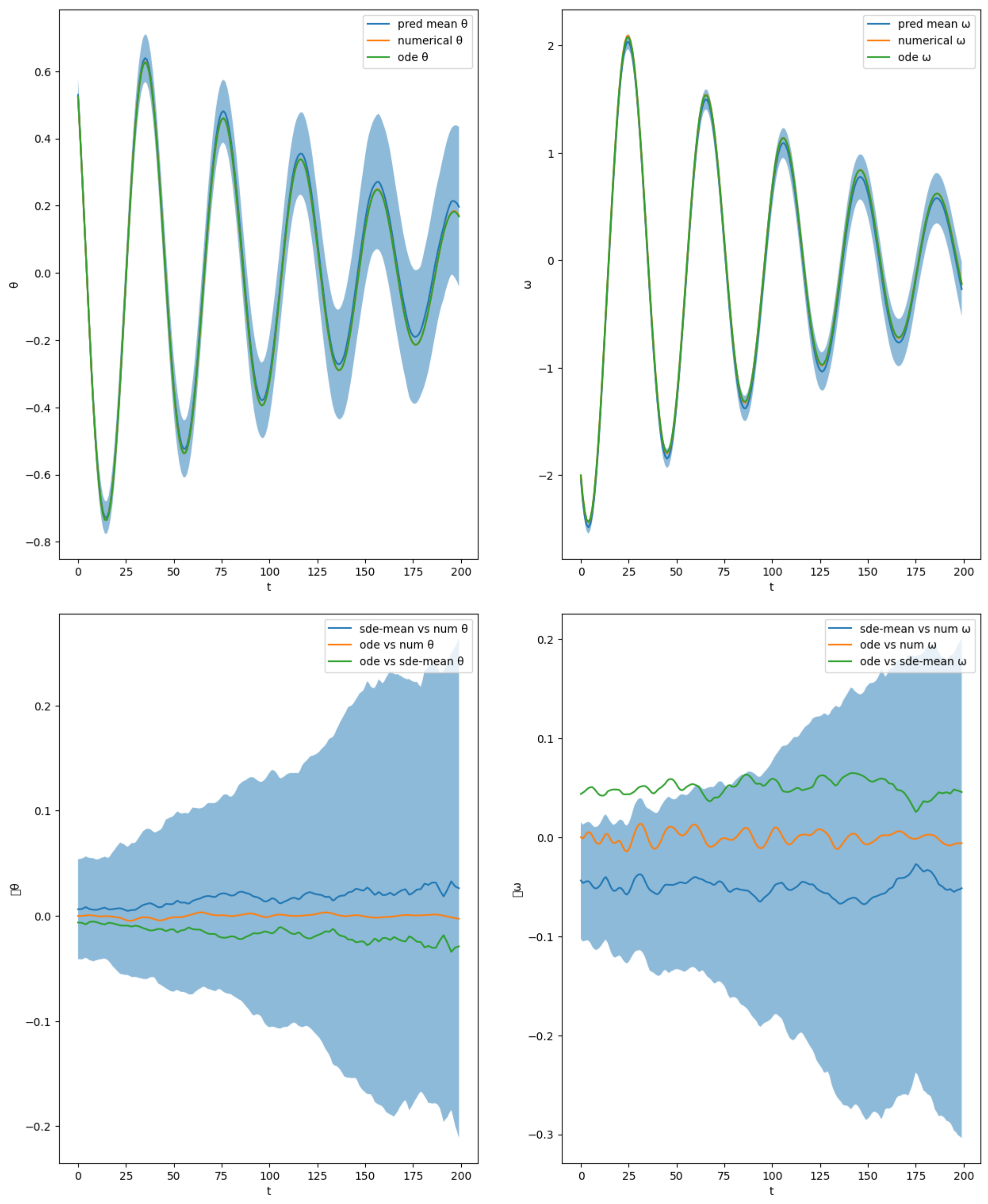


Figure 11 (ts = [10, 20], Δt = 0.05)

Discussion: In a real dynamic system, when the mean and the standard deviation of neural network prediction exceed a tolerance that indicates that the neural network model isn't confident enough about its prediction.

References:

[1] Chen, T. Q.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018. Neural ordinary differential equations. In Advances in neural information processing systems, 6571–6583.

[2] K. He et al. "Deep Residual Learning for Image Recognition". In: arXiv:1512.03385 (2015).

[3] K. Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: Empirical Methods in Natural Language Processing (2014).

[4] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: Neural Computation 9.8 (1997), pp. 1735-1780.

[5] K. Zubov et al. \NeuralPDE: Automating Physics-Informed Neural Networks (PINNs) with "Error approximations". In: arXiv:2107.09443 (2021).

[6] Liu Xuanqing, Si Si, Cao Qin, Kumar Sanjiv, and Hsieh Cho-Jui. 2019. Neural SDE: Stabilizing neural ode networks with stochastic noise. *Retrieved from https://arXiv:1906.02355*.

[7] Patrick Kidger, James Foster, Xuechen Li, and Terry Lyons. Efficient and accurate gradients for neural sdes. arXiv preprint arXiv:2105.13493,2021.