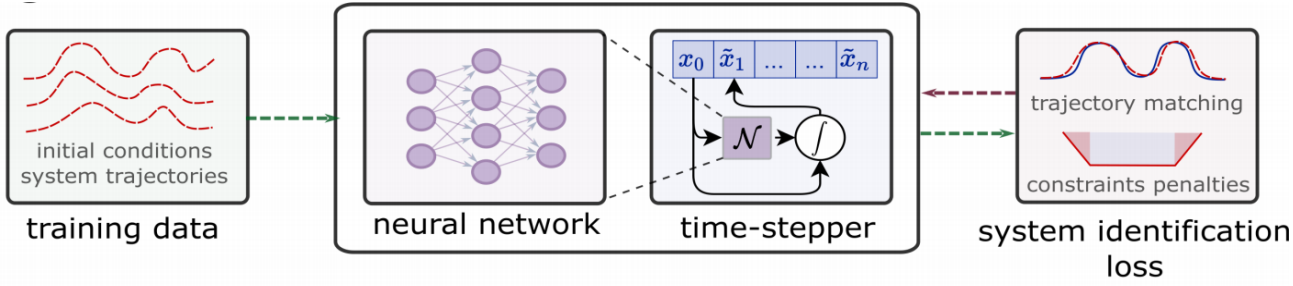


SciML Tutorial: Learning to Model (L2M)

Thomas Beckers
Assistant Professor
Department of Computer Science
Vanderbilt University

8th Annual Learning for Dynamics & Control Conference
University of Southern California, June 17

Problem formulation

unknown

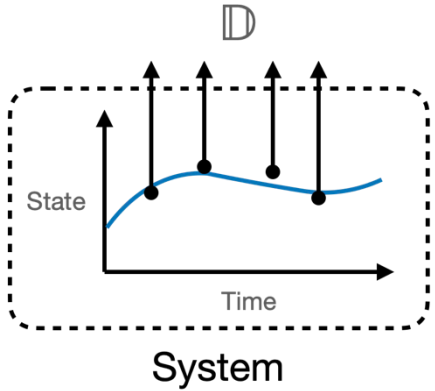
Physical system $\dot{x}(t) = \overbrace{f(x(t), u(t), t)}$

often Gaussian

Noisy measurements $\hat{x}_i = x(t_i) + \epsilon, \quad \epsilon \sim \mathcal{D}$

Dataset $\mathbb{D} = \{\{t_k, \hat{x}_k, u_k\}_{k=0}^{N}\}_{i=1}^{M}$

Observations # Trajectories



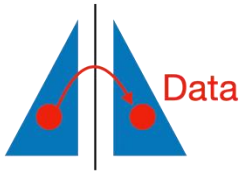
Goal

Given the dataset $\mathbb{D}$, we aim to find a dynamical model that

- represents a large class of systems -> expressive
- Integrates physical prior knowledge -> generalizes well
- allows uncertainty quantification -> reliable
- accelerates simulation -> fast solution

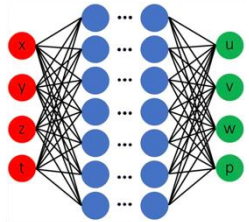
How to include physics?

Observational bias
Introducing data that embody underlying physics



Symmetry

Learning bias
Inference/learning algorithm selection, loss function, optimization constraints



$$Loss = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}$$

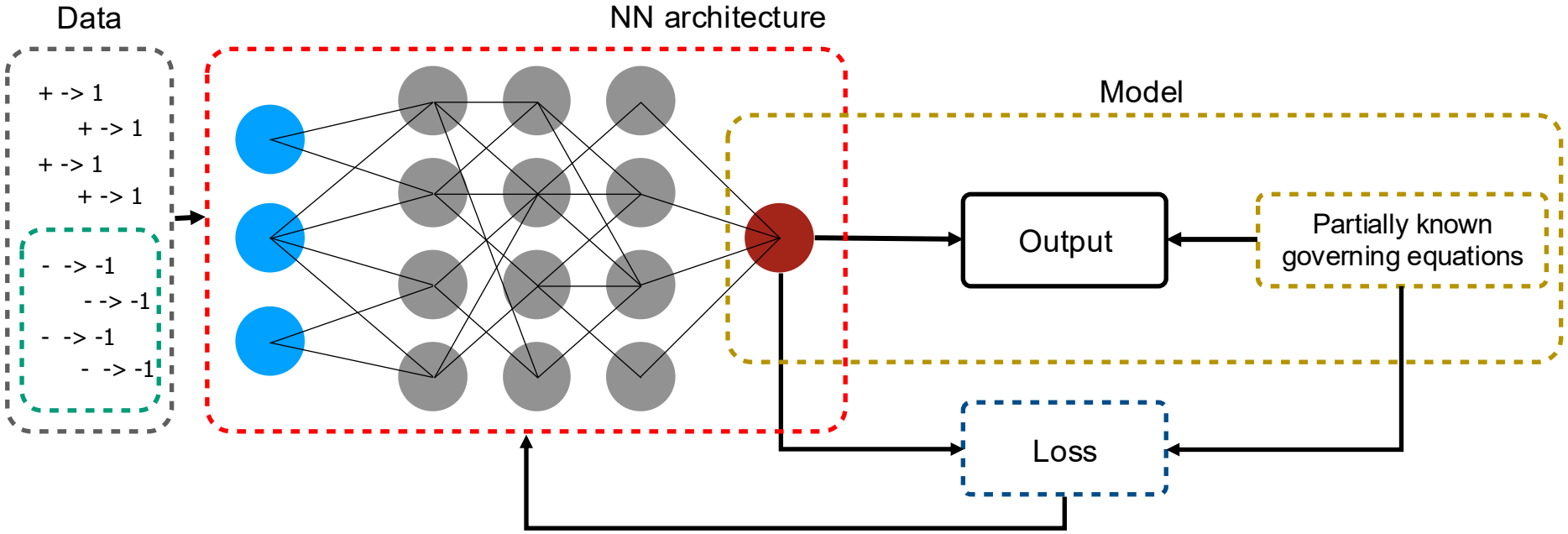
Inductive bias
Incorporate prior assumptions and physical constraints

Conservation laws

Discrepancy bias
Including “terms” from partial knowledge of the model

$$\dot{x} = Ax + Bu + f(x, u)$$

Example



Observation bias: Introducing data that embodies underlying physics

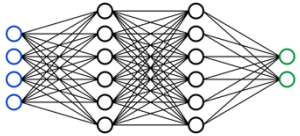
Inductive bias: Prior assumptions and physical constraints

Learning bias: Physics based loss function, optimization algorithm

Discrepancy bias: Introducing data that embodies underlying physics

Combining the best of both worlds

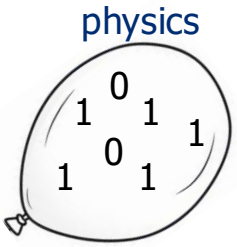
Learning-based



Scientific principles / physics

$$\dot{x} = f(t, x, u) = \dots$$

$$0 = f(\frac{\partial x}{\partial t}, \frac{\partial x}{\partial z}, \dots) = \dots$$



Structure to constrain/inform the model output to be physically consistent

Data...101011011

"Best" Learning technique?

- Neural Networks
- GNNs, RNNs, LSTMs
- Gaussian Processes
- Operator Learning

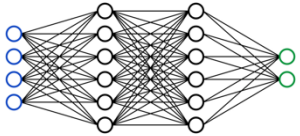
"Best" Structure?

- ODE / PDE
- Parametric
- Hamiltonian Mechanics
- Lagrangian Mechanics
- Port-Hamiltonian Systems

Choice depends on the use-case!

Combining the best of both worlds

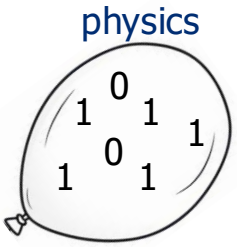
Learning-based



Scientific principles / physics

$$\dot{x} = f(t, x, u) = \dots$$

$$0 = f(\frac{\partial x}{\partial t}, \frac{\partial x}{\partial z}, \dots) = \dots$$



Structure to constrain/inform the model output to be physically consistent

Data...101011011

"Best" Learning technique?

- Neural Networks
- GNNs, RNNs, LSTMs
- Gaussian Processes
- Operator Learning

"Best" Structure?

- ODE / PDE
- Parametric
- Hamiltonian Mechanics
- Lagrangian Mechanics
- Port-Hamiltonian Systems

Descent amount of data + no physical prior knowledge

Neural ODE

Continuous-Time Ordinary Differential Equation Modeling

Given observed state-control trajectories

$$\mathbb{D} = \left\{ \{t_k,\ \hat{x}_k,\ u_k\}_{k=0}^{N},\ x_0 \right\}_{i=1}^{M},$$

sampled at temporal points t_k, and the initial conditions x_0. The objective is to learn parameters θ of a continuous-time dynamical model:

$$\frac{dx}{dt} = f_\theta(x(t),\ u(t),\ t), \qquad x(t_0) = x_0,$$

s.t. the simulated states $x(t_k)$ match the observed states $\hat{x}_k$ on the time domain $t \in [t_0, t_K]$.

Idea

- A neural network learns the flow field
- Output is obtained by integrating the learned ODE

$$\frac{dx}{dt} = f_\theta(x(t),\ u(t),\ t), \qquad x(t_0) = x_0,$$

Neural network parametrized by θ

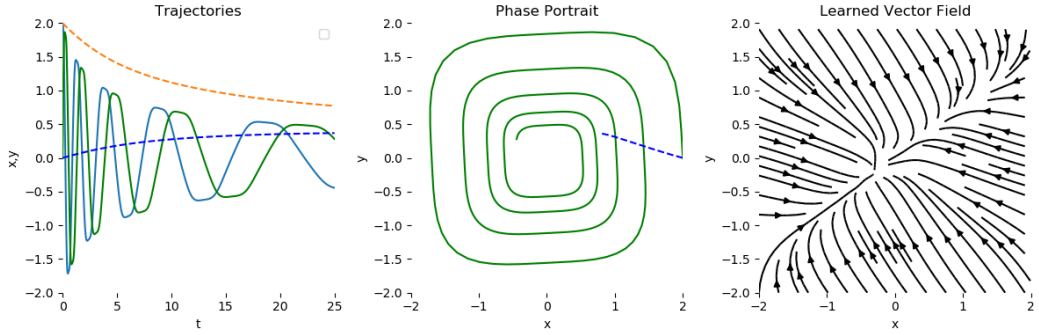
Neural ODE Loss

NODE Loss Function

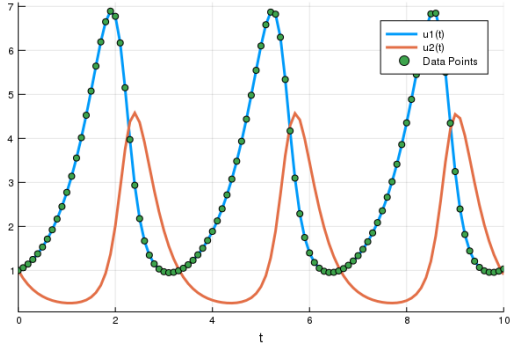
Given an NODE layer $\mathrm{ODESolve}(f_\theta(\cdot),\, t,\, x_0)$ that computes $x^i(t)$ from initial conditions x_0^i under inputs $u^i(t)$, the NODE loss measures trajectory mismatch and (optionally) model regularization:

$$\mathcal{L}(\theta) = \underbrace{\frac{1}{MN}\sum_{i=1}^{M}\sum_{k=0}^{N}\left\|x^i(t_k) - \hat{x}_k^i\right\|_2^2}_{\text{trajectory data fit}} + \underbrace{\lambda\mathcal{R}(\theta)}_{\text{model regularization}} .$$

https://colab.research.google.com/github/SciML-L4DC2026/SciML-L4DC2026/blob/main/notebooks/L2M/Example_1_NODE/Example_1_NODE.ipynb



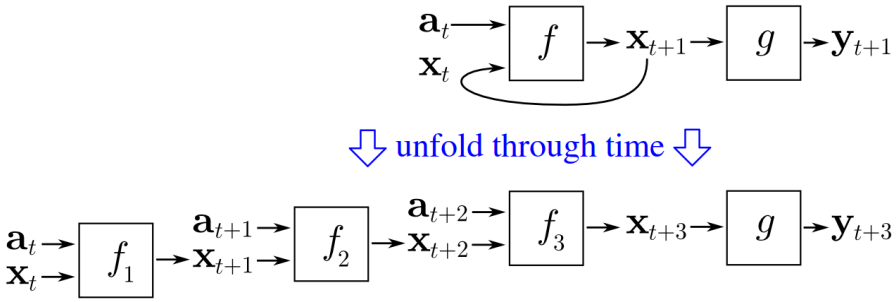
R. T. Q. Chen, et al., *Neural Ordinary Differential Equations*, 2019



docs.sciml.ai/DiffEqDocs/stable/

Neural ODE Training

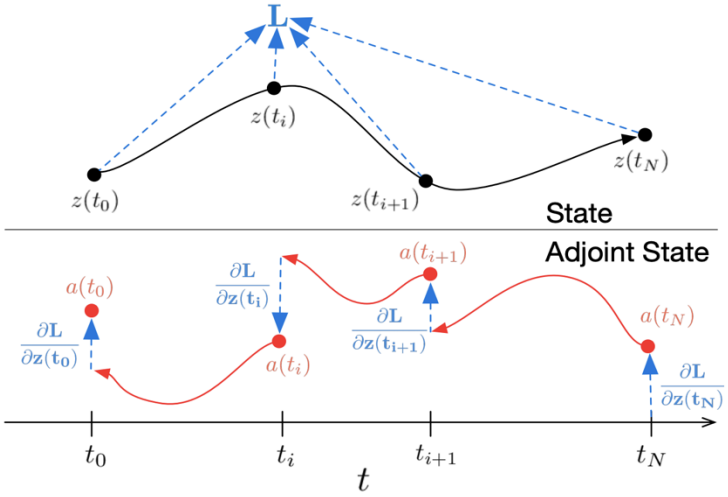
Backpropagation through time (BPTT)



Pros: exact for discretized systems, numerically more stable, wide support in modern AD libraries
Cons: high memory cost, requires storing all intermediate variables

https://en.wikipedia.org/wiki/Backpropagation_through_time

Adjoint sensitivity method

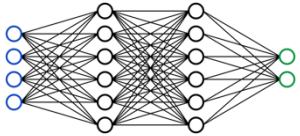


Pros: memory efficiency, no need to store intermediate variables, strong theory
Cons: error accumulation leading to numerical instability

R. T. Q. Chen, et al., *Neural Ordinary Differential Equations*, 2019

Combining the best of both worlds

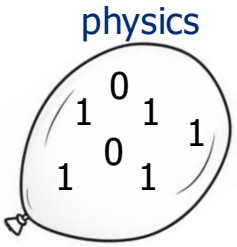
Learning-based



Scientific principles / physics

$$\dot{x} = f(t, x, u) = \dots$$

$$0 = f(\frac{\partial x}{\partial t}, \frac{\partial x}{\partial z}, \dots) = \dots$$



Structure to constrain/inform the model output to be physically consistent

Data...101011011

"Best" Learning technique?

- Neural Networks
- GNNs, RNNs, LSTMs
- Gaussian Processes
- Operator Learning

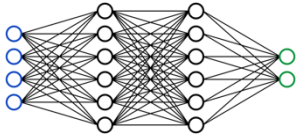
"Best" Structure?

- ODE / PDE
- Parametric
- Hamiltonian Mechanics
- Lagrangian Mechanics
- Port-Hamiltonian Systems

Choice depends on the use-case!

Combining the best of both worlds

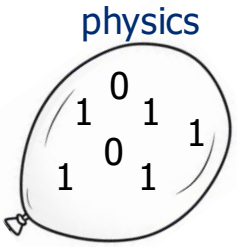
Learning-based



Scientific principles / physics

$$\dot{x} = f(t, x, u) = \dots$$

$$0 = f(\frac{\partial x}{\partial t}, \frac{\partial x}{\partial z}, \dots) = \dots$$



Structure to constrain/inform the model output to be physically consistent

Data...101011011

"Best" Learning technique?

- Neural Networks
- GNNs, RNNs, LSTMs
- Gaussian Processes
- Operator Learning

"Best" Structure?

- ODE / PDE
- Parametric
- Hamiltonian Mechanics
- Lagrangian Mechanics
- Port-Hamiltonian Systems

Little data + Energy-based prior

Gaussian Process

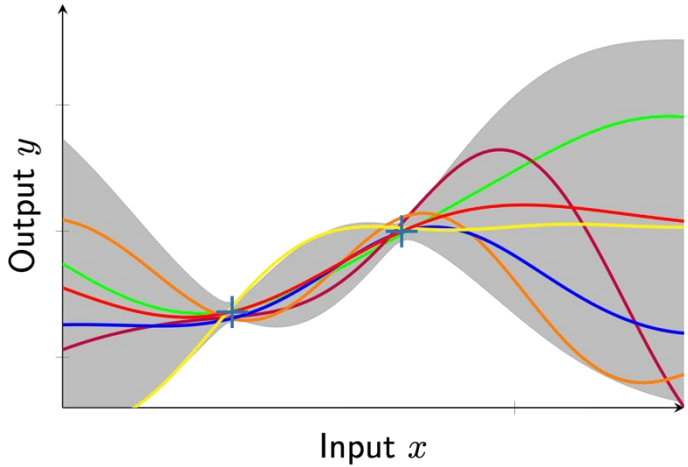
Prior: Gaussian distribution over function space

$$f(x) \sim GP(m(x), k(x, x'))$$

Mean function

Covariance

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}$$



C.E. Rasmussen. Gaussian Processes for Machine Learning. 2006

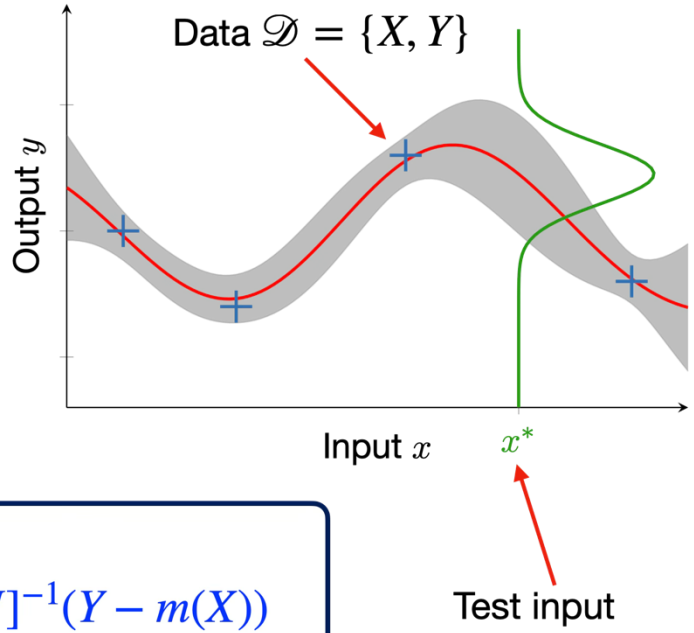
Gaussian Process

Prior: Gaussian distribution over function space

$$f(x) \sim GP(m(x), k(x, x'))$$

Mean function — Covariance

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}$$



Posterior

$$\mu(y^* \mid x^*, \mathscr{D}) = m(x^*) + k(x^*, X)[K(X, X) + \sigma_n^2 I]^{-1}(Y - m(X))$$

$$\Sigma(y^* \mid x^*, \mathscr{D}) = k(x^*, x^*) - k(x^*, X)[K(X, X) + \sigma_n^2 I]^{-1} k(x^*, X)^\top$$

C.E. Rasmussen. Gaussian Processes for Machine Learning. 2006

Gaussian Process

Posterior mean

Data points Covariance

$$\mu(y \mid x, \mathcal{D}) = m(x) + \sum_{i}^{N} w_i k(x, X_i)$$

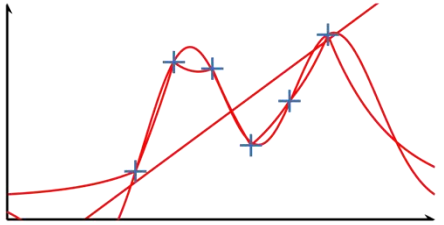
Prior mean function Weighting factor

Via mean function

$$m(x) = \ldots$$

Discrepancy bias

Via covariance function



Inductive bias

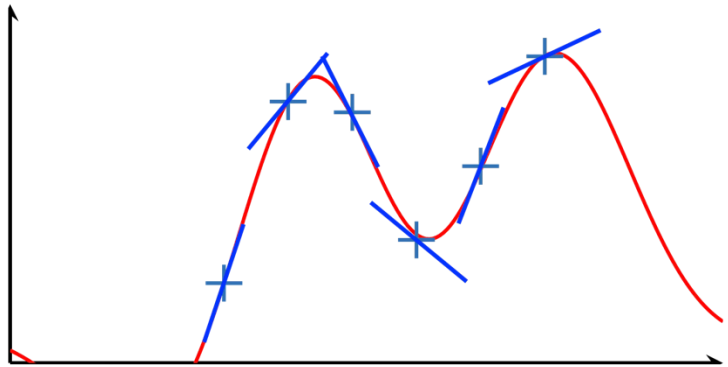
Linear operators

GPs are closed under linear operators

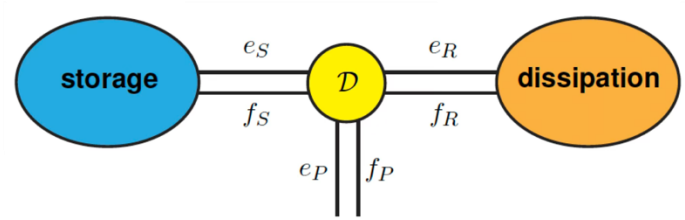
$$f(x) = \mathcal{L}_x g(x) \sim GP(\mathcal{L}_x \mu, \mathcal{L}_x k \mathcal{L}_x^\top)$$

Example: Differential operator $\mathcal{L}_x = \dfrac{\partial}{\partial x}$

$$\begin{bmatrix} y \\ Y' \end{bmatrix} = \mathcal{N}\left(0, \begin{bmatrix} k(x,x) & \frac{\partial}{\partial x} k(x,X) \\ \frac{\partial}{\partial x} k(x,X)^\top & \frac{\partial}{\partial x}^\top k(X,X) \frac{\partial}{\partial x} \end{bmatrix}\right)$$



Physics structure



[A. Van Der Schaft and D. Jeltsema]

Port-Hamiltonian system

Interconnection Dissipation Hamiltonian

$$\dot{x} = [J(x) - R(x)]\frac{\partial H}{\partial x} + G(x)u$$

$$y = G(x)^\top \frac{\partial H}{\partial x}$$

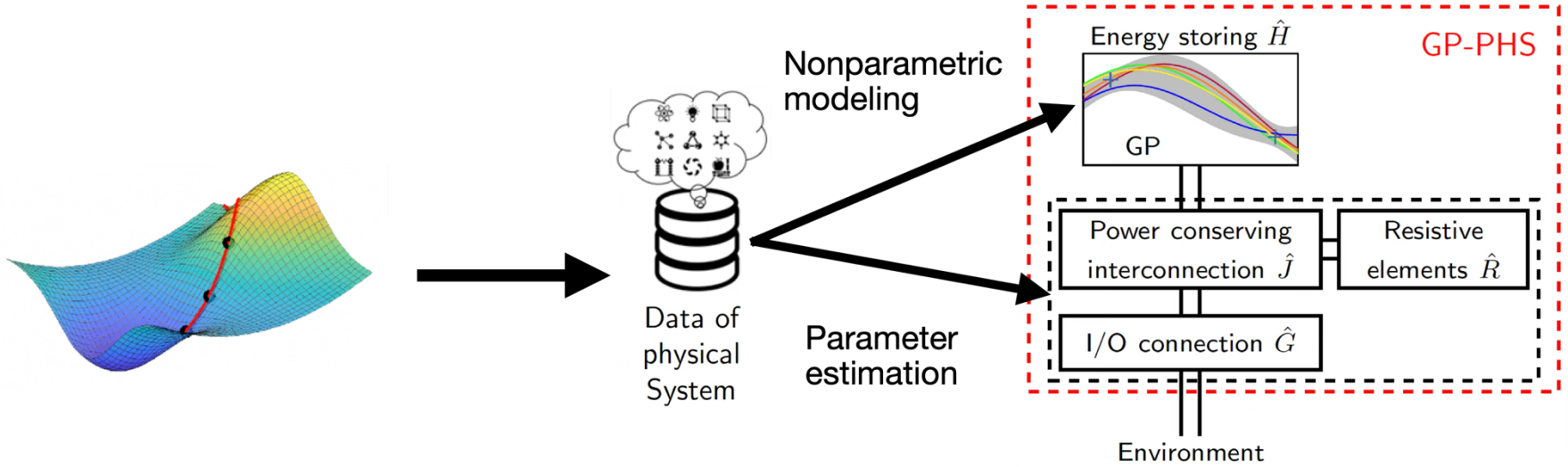
Input

Output

- Skew-symmetric J ensures energy conservation
- Input and output defines a passive systems
- Connection with another PHS preserves the structure
- Suitable for multi-domain applications via energy flows

A. Van Der Schaft and D. Jeltsema. "Port-Hamiltonian systems theory: An introductory overview." Foundations and Trends in Systems and Control. 2014

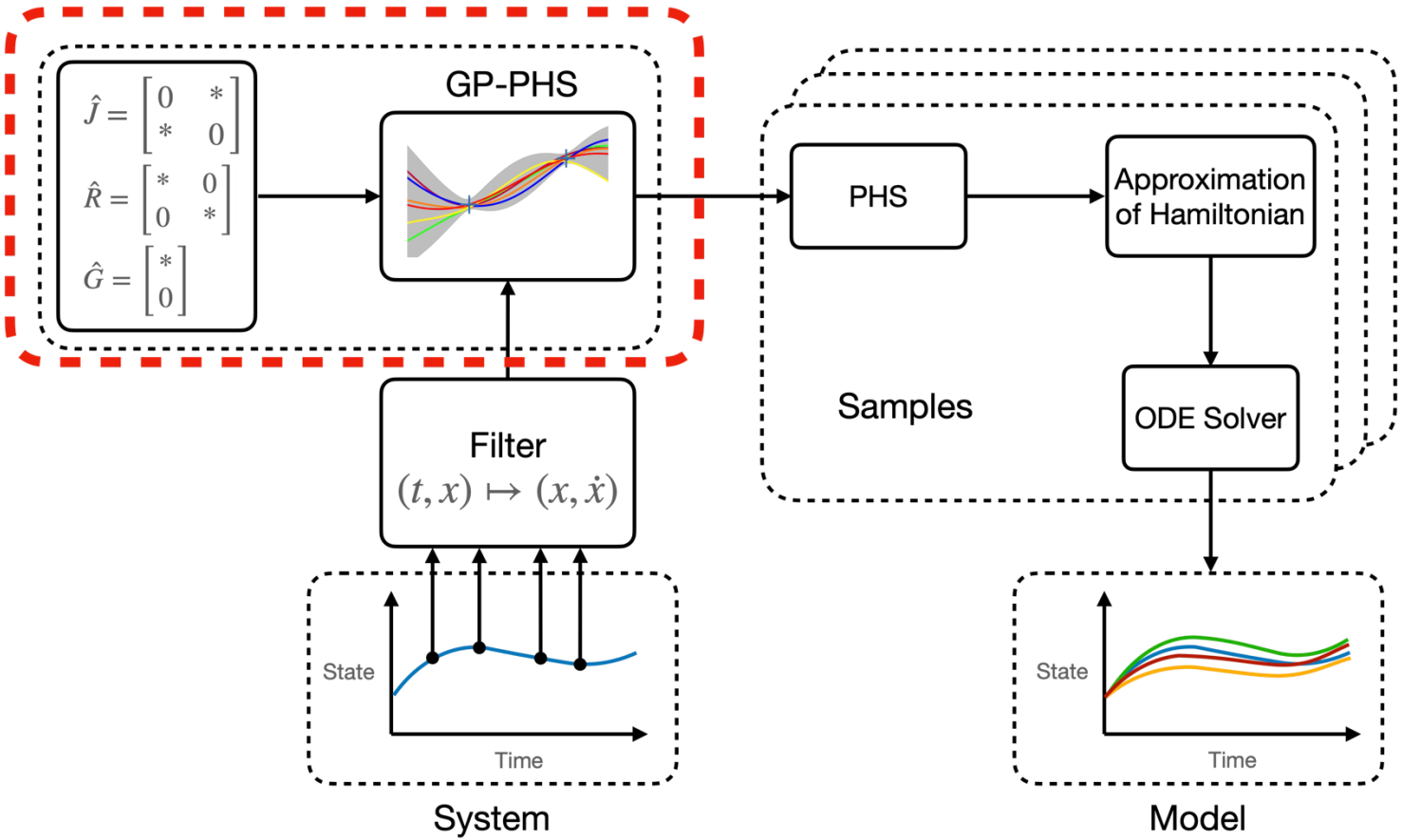
GP-PHS



- Physics-constrained Gaussian Process model
- GP-PHS includes all possible PHS under the GP prior on the Hamiltonian

T. Beckers, et al. “Gaussian Process Port-Hamiltonian Systems: Bayesian Learning with Physics Prior”. CDC, 2022

GP-PHS



T. Beckers, et al. “Gaussian Process Port-Hamiltonian Systems: Bayesian Learning with Physics Prior”. CDC, 2022

GP-PHS: Training

GP prior on Hamiltonian

$$\dot{x} = [J(x) - R(x)]\frac{\partial H}{\partial x} + G(x)u \quad \Longrightarrow \quad \dot{x} \sim GP(\hat{G}(x)u, k_{phs}(x, x'))$$

Linear operator

Port-Hamiltonian Kernel

$$k_{phs}(x, x') = \sigma_f^2 [\hat{J}(x) - \hat{R}(x)] \left[\frac{\partial}{\partial x \partial x'} \underbrace{\exp(-\|x - x'\|_\Lambda^2)}_{\text{Squared exp. kernel}} \right] \overbrace{[\hat{J}(x') - \hat{R}(x')]^\top}^{\text{Parametrized estimates}}$$

Optimization problem

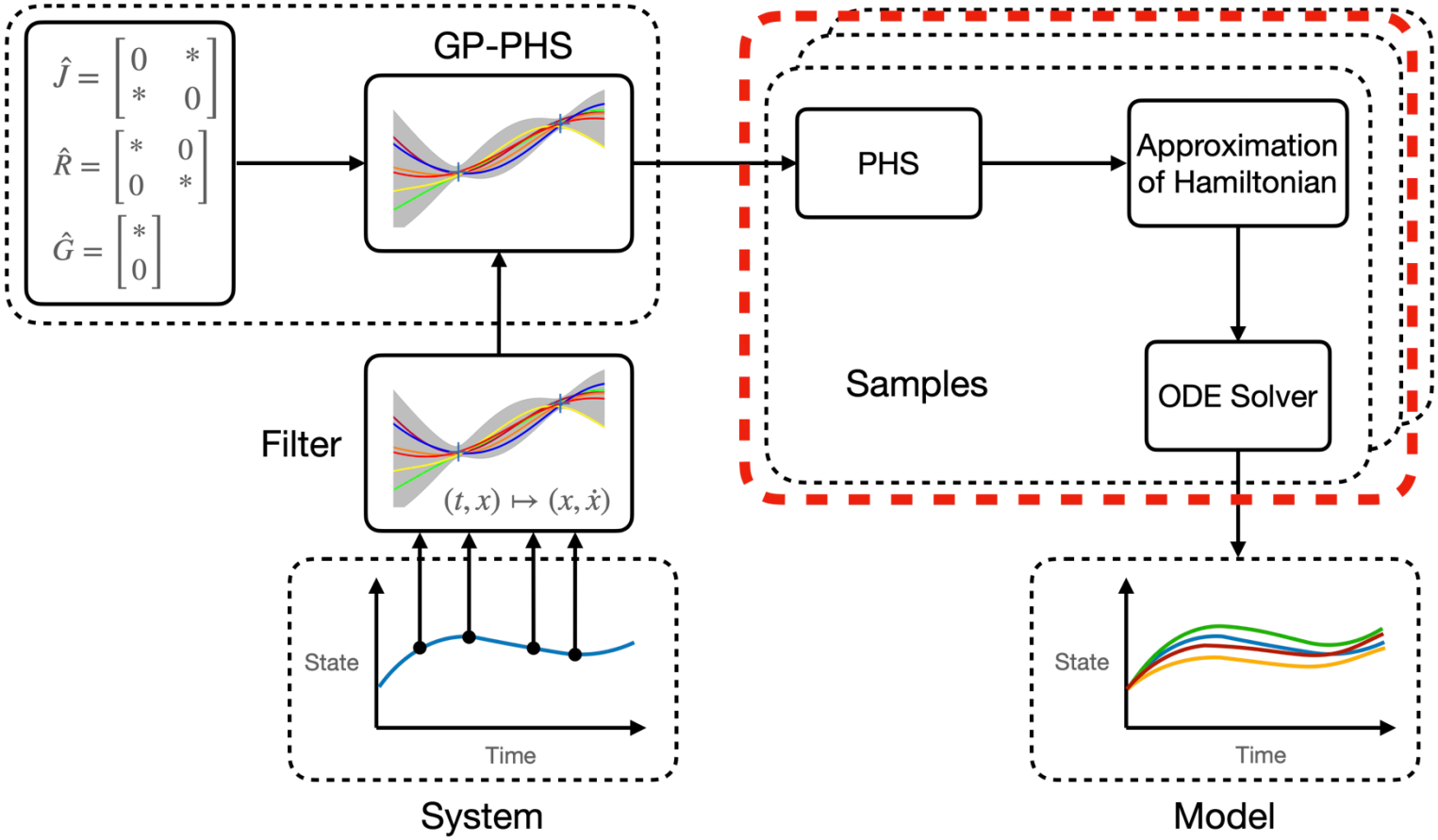
$$\min_\theta \; -\log p(\dot{X} \mid \theta, X) \sim \dot{X}^\top [K_{\text{phs}} + \Delta]^{-1} \dot{X} + \log |K_{\text{phs}} + \Delta|$$

$$\theta = \text{Kernel parameters} + \text{unknown PHS parameters}$$

$$\text{s.t.} \quad J = -J^\top, \quad R \succeq 0$$

T. Beckers, et al. “Gaussian Process Port-Hamiltonian Systems: Bayesian Learning with Physics Prior”. CDC, 2022

GP-PHS



T. Beckers, et al. “Gaussian Process Port-Hamiltonian Systems: Bayesian Learning with Physics Prior”. CDC, 2022

GP-PHS: Sampling

PHS

$$\begin{bmatrix} \dot{X} \\ \hat{f}(x^*) \end{bmatrix} = \mathcal{N}\left(0, \begin{bmatrix} K_{phs} & k_{phs}(X, x^*) \\ k_{phs}(X, x^*)^\top & k_{phs}(x^*, x^*) \end{bmatrix}\right)$$

$$\dot{x} = \hat{f}(x, \omega)$$

Not callable
(computational expensive)

Approximation of Hamiltonian

$$\begin{bmatrix} \dot{X} \\ \hat{H}(x^*) \end{bmatrix} = \mathcal{N}\left(0, \begin{bmatrix} K_{phs} & k_{\dot{x}H}(X, x^*) \\ k_{\dot{x}H}(X, x^*)^\top & k_{HH}(x^*, x^*) \end{bmatrix}\right)$$

$$\hat{H}(x^*, \omega) \xrightarrow{\text{Approx.}} \hat{H}^*(x^*)$$

ODE Solver

$$\dot{x} = [\hat{J}(x) - \hat{R}(x)]\frac{\partial \hat{H}^*}{\partial x} + \hat{G}(x)u \xrightarrow{\text{Solver}} x(t)$$

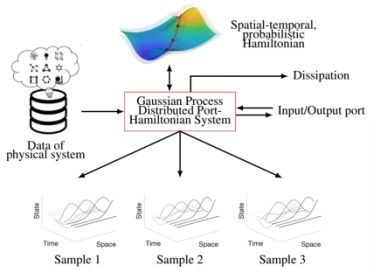
Approximation of H: Callable and structure preserving

T. Beckers, et al. “Gaussian Process Port-Hamiltonian Systems: Bayesian Learning with Physics Prior”. CDC, 2022

Use-cases and extensions

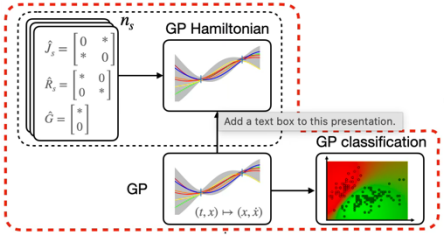
PDEs

Energy-based learning of PDE dynamics



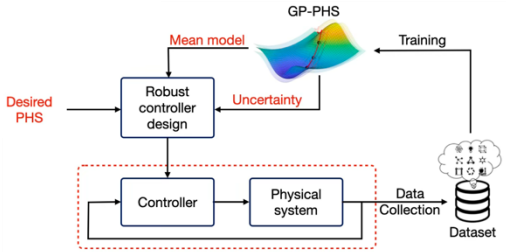
Switching systems

Learning the switching surface



Control

Passivity-based control approaches



...and more

Full Bayesian: Learning from input-output data

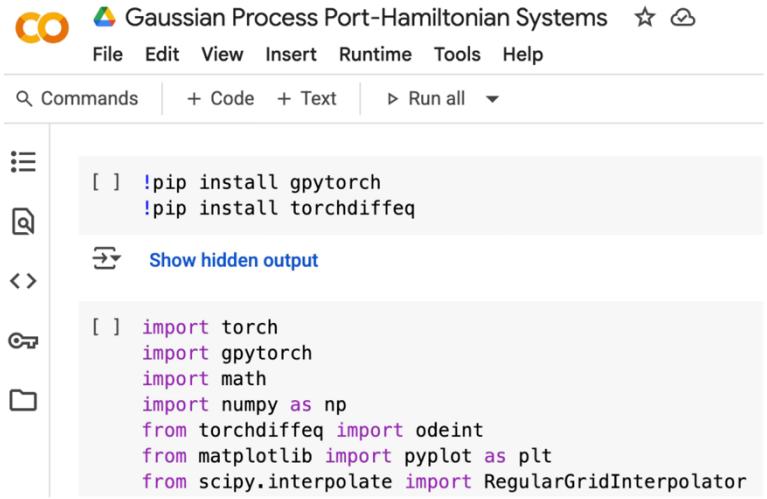
Ewering, Jan-Hendrik, et al. "Learning Dynamics from Input-Output Data with Hamiltonian Gaussian Processes." L4DC, 2026

Finite-element framework

Courteville, Florian, et al. "PFEM-GP-dpHs: a finite element framework for combining Gaussian processes and infinite-dimensional port-Hamiltonian systems." L4DC, 2026

K. Tan, et al. "Physics-Constrained Learning of PDE Systems with Uncertainty Quantified Port-Hamiltonian Models". L4DC, 2024
T. Beckers, et al. "Learning Switching Port-Hamiltonian Systems with Uncertainty Quantification". IFAC WC, 2023
T. Beckers. "Data-driven Bayesian Control of Port-Hamiltonian Systems". CDC, 2023

GP-PHS: Implementation



Implementation based on Torch and GPyTorch

- Use of Torch features such as GPU learning
- Tools from GPyTorch (speed up computation, etc.)
- PHS-Kernel implemented based on

RBFKernelGradGrad

class gpytorch.kernels.RBFKernelGradGrad(*ard_num_dims=None, batch_shape=None, active_dims=None, lengthscale_prior=None, lengthscale_constraint=None, eps=1e-06, **kwargs*) [source]

Computes a covariance matrix of the RBF kernel that models the covariance between the values and first and second (non-mixed) partial derivatives for inputs $\mathbf{x_1}$ and $\mathbf{x_2}$.

See `gpytorch.kernels.Kernel` for descriptions of the lengthscale options.

https://colab.research.google.com/github/SciML-L4DC2026/SciML-L4DC2026/blob/main/notebooks/L2M/Example_2_GPPHS/GP_PHS.ipynb

$$\begin{bmatrix} k(x,x) & \frac{\partial}{\partial x}k(x,X) \\ \frac{\partial}{\partial x}k(x,X)^\top & \frac{\partial}{\partial x}^\top k(X,X)\frac{\partial}{\partial x} \end{bmatrix}$$

Learning to Control (L2M) Summary

- **L2M via data + structure/physics**
  - Combining data and physics to improve model efficiency and generalization
  - Embedding prior knowledge through model structure, constraints, and inductive biases
  - Quantifying uncertainty to assess model confidence and guide decision-making

- **Challenges**
  - Balancing model expressiveness and physical interpretability
  - Efficient training and uncertainty-aware prediction
  - Learning from sparse, heterogeneous, noisy, and incomplete observations

- **Code**
  - github.com/pnnl/neuromancer
  - www.tbeckers.com