Python and AI/ML for Weather, Climate and Environmental Applications

Let us enjoy playing with Python and AI/ML!

Five-Day Schedule Overview

Time	Day 1	Day 2	Day 3	Day 4	Day 5
09:00–10:00	Opening by ECMWF DG, Start: Coding & Science in the Age of AI	Neural Network Architectures	Diffusion and Graph Networks	MLOps Foundations	Model Emulation, AIFS and AICON
10:00–11:00	**Lab:** Python Startup: Basics	**Lab:** Feed-forward and Graph NNs	**Lab:** Graph Learning with PyTorch	**Lab:** Containers and Reproducibility	**Lab:** Emulation Case Studies
11:00–12:00	Python, Jupyter and APIs	Large Language Models	Agents and Coding with LLMs	CI/CD for Machine Learning	AI-based Data Assimilation
12:00–12:45	**Lab:** Work environments, Python everywhere	**Lab:** Simple Transformer and LLM Use	**Lab:** Agent Frameworks	**Lab:** CI/CD Pipelines	**Lab:** Graph-based Assimilation
12:45–13:30			**Lunch Break**		
13:30–14:30	Visualising Fields and Observations	Retrieval-Augmented Generation (RAG)	DAWID System and Feature Detection	Anemoi: AI-based Weather Modelling	AI and Physics
14:30–15:30	**Lab:** GRIB, NetCDF and Obs Visualisation	**Lab:** RAG Pipeline	**Lab:** DAWID Exploration	**Lab:** Anemoi Training Pipeline	**Lab:** Physics-informed Neural Networks
15:30–16:15	Introduction to AI and Machine Learning	Multimodal Large Language Models	MLflow: Managing Experiments	**The AI Transformation**	Learning from Observations Only
16:15–17:00	**Lab:** Torch Tensors and First Neural Net	**Lab:** Radar, SAT and Multimodal Data	**Lab:** MLflow Hands-on	**Lab: How work style could change**	**Lab:** ORIGEN and Open Discussion
17:00–20:00				Joint Dinner	

Lecture 19: AI and Physics and Data

Core question
Given a dynamical system, what can machine learning do?

- Solve known equations (ODE/PDE)
- Discover unknown governing laws from observations
- Emulate complex dynamics as a surrogate model

Unifying viewpoint
All methods impose (explicitly or implicitly) a constraint:
state evolution must be consistent with $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$.

How can different ML approaches enforce this consistency?

Three routes in this lecture

1. **PINNs:** physics drives training
2. **SINDy:** sparse laws from data
3. **Neural RHS learning:** black-box emulation

Key trade-offs

- Accuracy vs. interpretability
- Data-efficiency vs. flexibility
- Stability / extrapolation vs. expressiveness

Message:
same goal (dynamics), different framework.

Lecture Roadmap

Part I — Physics-Informed Neural Networks

- learn solutions of known equations
- training uses ODE/PDE residuals + anchor conditions
- representation matters for extrapolation

Part II — Discovering equations from data

- **SINDy:** sparse regression on a function library
- **Neural SINDy:** smooth NN trajectory ⇒ stable derivatives

Part III — Learning the Force Term

- learn the unknown RHS / forcing from data:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}_\theta(\mathbf{x})$$

- hybrid modeling: known physics + learned closure
- stable rollout by integrating the learned system

Part IV — Causal Modeling with Neural Networks

- distinguish correlation vs. cause
- learn structural relations (SCMs) as NN modules
- intervene and test counterfactual predictions:

$$do(X = x) \quad \Rightarrow \quad p(Y \mid do(X = x))$$

A Minimal Physics-Informed Neural Network (PINN)

Problem setup

We consider a simple second-order ODE:

$$y''(x) + y(x) = 0$$

with boundary conditions

$$y(0) = 0, \qquad y'(0) = 1.$$

The unique solution is

$$y(x) = \sin(x).$$

This example is deliberately simple, but already captures all essential PINN ingredients.

PINN idea

- Approximate $y(x)$ by a neural network $y_\theta(x)$
- No training data $y(x)$ are used
- Training is driven by physics constraints

What is enforced

- Differential equation via automatic differentiation
- Boundary conditions via penalty terms

The network learns the solution by minimizing violations of physics.

PINN Loss: Physics Instead of Data

ODE residual

Using automatic differentiation, we compute

$$y'_\theta(x), \qquad y''_\theta(x).$$

The differential equation is enforced by minimizing

$$r(x) = y''_\theta(x) + y_\theta(x).$$

The corresponding loss term is

$$\mathcal{L}_{\text{ODE}} = \frac{1}{N}\sum_{i=1}^{N}\big(y''_\theta(x_i) + y_\theta(x_i)\big)^2.$$

Collocation points x_i are sampled in the domain.

Boundary conditions

Boundary (anchor) constraints enforce uniqueness:

$$y_\theta(0) = 0, \qquad y'_\theta(0) = 1.$$

This yields the boundary loss

$$\mathcal{L}_{\text{BC}} = \big(y_\theta(0)\big)^2 + \big(y'_\theta(0) - 1\big)^2.$$

Total loss

The parameter λ controls the strength of the boundary conditions.

$$\mathcal{L} = \mathcal{L}_{\text{ODE}} + \lambda\,\mathcal{L}_{\text{BC}}.$$

No data term appears anywhere in the loss.

PINN Result: Naive Training, Extended Evaluation

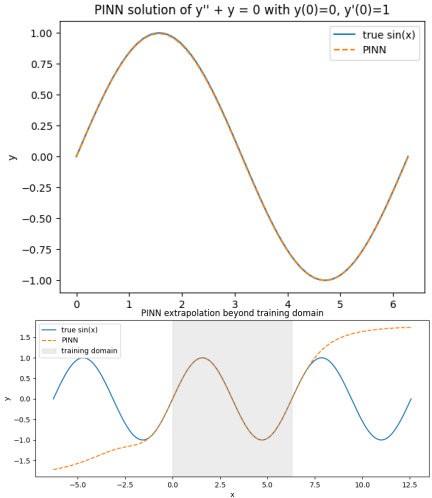
Training setup

- ODE residual enforced on $[0, 2\pi]$
- Boundary conditions at $x = 0$
- No data, no periodic constraints
- Standard MLP representation

Evaluation

- Solution evaluated on a larger domain
- Outside the region where physics was enforced

This tests extrapolation, not interpolation.



Improving Extrapolation by Wider Residual Sampling

Key modification

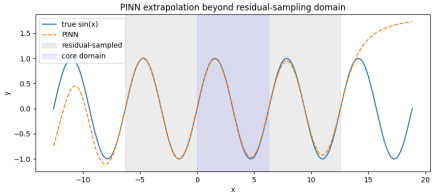
The PINN formulation is unchanged, but the ODE residual is enforced on a wider domain.

Training setup

- ODE residual sampled beyond $[0, 2\pi]$
- Boundary conditions still imposed at $x = 0$
- Same network architecture and loss terms

Effect

- Physics is enforced more globally
- Extrapolation becomes significantly more stable
- No change in representation or constraints



This already fixes many extrapolation problems for simple ODEs.

Improving Representation with Fourier Features

Motivation

Standard MLPs learn functions of x that are biased toward smooth, slowly varying behavior.

Oscillatory solutions, such as

$$y(x) = \sin(x),$$

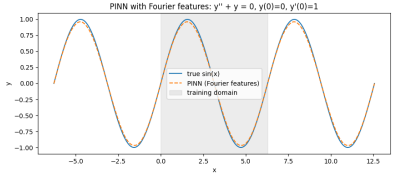
are therefore harder to represent and extrapolate.

Fourier feature idea

Instead of learning directly from x, we apply a fixed feature map:

$$x \mapsto \big(\sin(\omega_k x),\ \cos(\omega_k x)\big)_{k=1}^{m}.$$

The neural network then learns a function of these periodic features.



Effect on the PINN

- Periodicity is easy to represent
- Long-range extrapolation improves
- Fewer parameters are needed

Interpretation

- Linear models become Fourier series fits
- Nonlinear MLPs allow mode interactions

This changes the representation, not the physics.

Result: Fourier-Feature PINN for a Modulated Oscillation

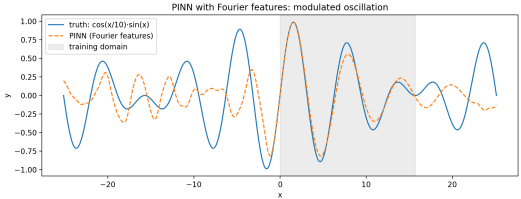
What changed compared to previous examples

- Governing ODE and anchor conditions unchanged
- Same PINN loss formulation
- Only the input representation is modified

The network uses a Fourier feature embedding.

Observed behavior

- Accurate solution on the training domain $[0, 5\pi]$
- Stable extrapolation far beyond the training region
- Correct phase and amplitude over many oscillations



Gray shading indicates the training domain. The solution is evaluated well beyond the region where the ODE residual was enforced.

Representation choice alone can control extrapolation quality.

SINDy: Sparse Identification of Nonlinear Dynamics

Problem setting

We observe a dynamical system

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)), \qquad \mathbf{x}(t) \in \mathbb{R}^n,$$

from time series data $\mathbf{x}(t_i)$.

Key assumption (sparsity)

The vector field can be written as a sparse combination of candidate functions:

$$\dot{\mathbf{x}}(t) \;\approx\; \Theta(\mathbf{x}(t))\,\Xi,$$

where

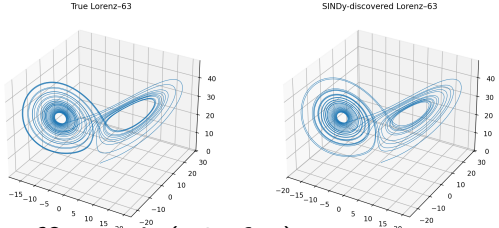
- $\Theta(\mathbf{x})$ is a library of functions (e.g. $1, x, y, z, xy, xz, yz, \dots$),
- Ξ is a sparse coefficient matrix.

Identification

SINDy solves a sequence of

$$\min_{\Xi} \; \|\Theta(\mathbf{x})\Xi - \dot{\mathbf{x}}\|_2^2$$

with thresholding to eliminate small coefficients.



Lorenz–63 example (noise-free).

Left: true trajectory. Right: trajectory from *SINDy*.

SINDy Sparse Regression of Nonlinear Dynamics: Lorenz–63 System

True governing equations

The Lorenz–63 system is defined by

$$\dot{x} = \sigma(y - x), \qquad \sigma = 10,$$
$$\dot{y} = x(\rho - z) - y, \qquad \rho = 28,$$
$$\dot{z} = xy - \beta z, \qquad \beta = \tfrac{8}{3}.$$

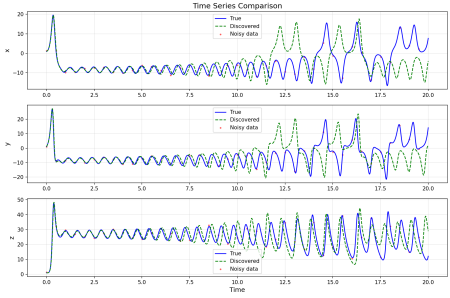
SINDy function library

SINDy assumes the dynamics can be written as a sparse linear combination of candidate functions:

```
Theta(x,y,z) = [  1,  x, y, z,
                  x*y, x*z, y*z ]
```

Only a few of these terms are retained in each equation after sparse regression.

Goal: recover the correct active terms and coefficients directly from time series data.



Time series of the Lorenz–63 state variables used as input for SINDy (blue). Numerical derivatives are estimated and matched against the candidate library during sparse regression. Discovered dynamics evolution in Green.

Neural SINDy: Using Neural Networks as Smooth Surrogates

Neural SINDy augments classical SINDy by introducing a neural network as a smooth surrogate for the observed trajectory:

$$\mathbf{x}(t) \longrightarrow \mathbf{x}_\theta(t).$$

The neural network is trained on noisy observations $\mathbf{x}_{\text{obs}}(t_i)$, but constrained to produce a *smooth time-continuous representation*.

Neural trajectory fitting

The network parameters θ are obtained by minimizing

$$\min_\theta \sum_i \|\mathbf{x}_\theta(t_i) - \mathbf{x}_{\text{obs}}(t_i)\|^2 + \alpha \int \left\| \tfrac{d}{dt}\mathbf{x}_\theta(t) \right\|^2 dt.$$

A regularization term penalizes rapid temporal variations and suppresses noise amplification.

Derivative estimation via autograd

Once trained, time derivatives are computed analytically:

$$\dot{\mathbf{x}}_\theta(t) = \frac{d}{dt}\mathbf{x}_\theta(t),$$

using automatic differentiation.

Sparse discovery step

The smoothed trajectory and its derivatives are then passed to the *unchanged* SINDy pipeline:

$$\dot{\mathbf{x}}_\theta(t) \approx \Theta(\mathbf{x}_\theta(t))\,\Xi, \qquad \Xi \text{ sparse.}$$

Here:

Neural networks are *not* used to represent the dynamics, but only to stabilize derivative estimation prior to sparse regression.

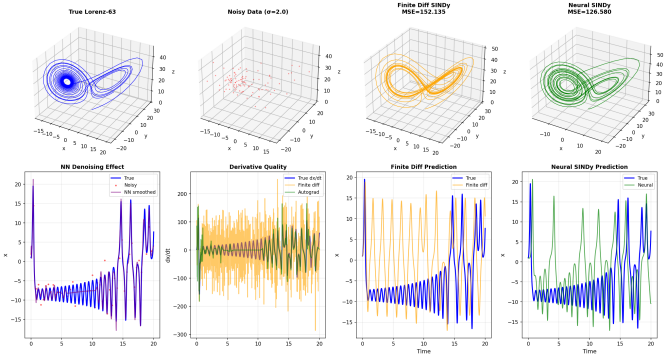
Neural SINDy: Results on Noisy Lorenz–63 Data

Experimental setup

- Lorenz–63 system
- Strong additive noise on observations
- Identical SINDy library and sparsity settings

Comparison

- Classical SINDy: finite-difference derivatives
- Neural SINDy: NN-smoothed trajectory + autograd



Comparison of trajectories, derivatives, and predictions for classical vs. Neural SINDy on noisy data.

Outcome

- Neural SINDy recovers an improved equation structure

Alternative: Learning the Full RHS with Neural Networks

Black-box RHS learning

Instead of discovering equations, one may directly learn the vector field

$$\dot{\mathbf{x}} = \mathbf{f}_\theta(\mathbf{x}),$$

where $\mathbf{f}_\theta$ is a neural network.

The network is trained to match observed time derivatives or trajectories.

- Very flexible function class
- Can approximate complex, unknown dynamics
- Naturally handles noise with regularization

Neural SINDy: a different goal

Neural SINDy uses neural networks only to stabilize intermediate steps (denoising and differentiation), but still identifies

$$\dot{\mathbf{x}} \approx \Theta(\mathbf{x})\,\Xi,$$

with a sparse, explicit structure.

Key distinction:

- Neural RHS learning $\rightarrow$ predictive black box
- Neural SINDy $\rightarrow$ *interpretable equations*

This distinction is critical in scientific modeling, where the goal is understanding, not only prediction.

Neural RHS Learning: Results on Lorenz–63

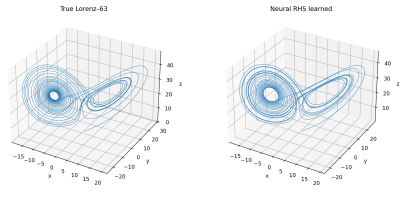
Experimental setup

- Lorenz–63 system
- Neural network trained on $(\mathbf{x}, \dot{\mathbf{x}})$ pairs
- No sparsity or physics constraints

Observations

- Learned vector field $\mathbf{f}_\theta$ reproduces the chaotic attractor
- Short- to medium-term trajectories remain accurate, **climatology ok!**
- Long-term divergence is unavoidable due to chaos

Interpretation:
The neural network successfully emulates the dynamics, but the governing equations remain hidden.



Neural RHS Learning in State Space

Experimental setup

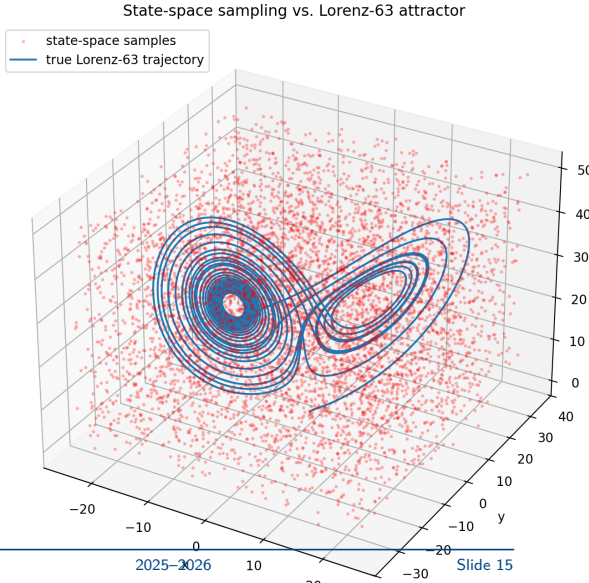
- Lorenz–63 vector field sampled in state space
- Random training points covering a 3D domain
- Neural network trained on $(\mathbf{x}, \dot{\mathbf{x}})$ pairs

Key difference to trajectory-based learning

- Training data no longer restricted to the attractor
- Vector field is constrained in a full region of state space
- Dynamics are learned as a global mapping

Outcome:

The learned neural vector field reproduces the Lorenz dynamics consistently from unseen initial conditions, not only along the original trajectory.



Learning Dynamical Systems: Goals and Methods

Three fundamentally different goals

- **Solve known equations**
  - Given: governing ODE/PDE
  - Task: compute the solution
  - Method: **PINNs**
- **Discover equations from data**
  - Given: time series observations
  - Task: identify governing laws
  - Method: **SINDy / Neural SINDy**
- **Emulate dynamics**
  - Given: state–derivative pairs
  - Task: reproduce system behavior
  - Method: **Neural RHS learning**

What is learned in each case?

- **PINNs**
  - learn the *solution* $x(t)$
  - equations are assumed known
- **SINDy / Neural SINDy**
  - learn *explicit equations*
  - sparse, interpretable models
- **Neural RHS learning**
  - learn a *vector field*
  - accurate dynamics, but opaque

Key trade-off:

Interpretability $\leftrightarrow$ Flexibility

Physics-Constrained Neural Emulators: 1D Periodic Advection

We consider 1D linear advection on a periodic domain:

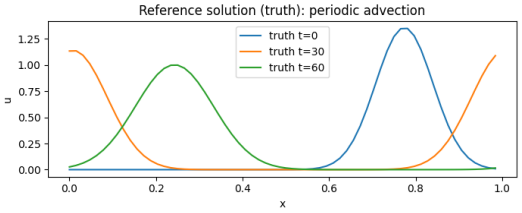
$$\partial_t u + c\,\partial_x u = 0, \qquad x \in [0,1], \quad u(0) = u(1).$$

Key physical properties:

- Pure translation on a ring
- Exact mass conservation
- Smooth initial conditions remain smooth

Learning task

- Learn a one-step map $u^n \mapsto u^{n+1}$
- Training data from a conservative upwind solver
- Compare different neural inductive biases



Reference solution (truth).
Gaussian bump advected periodically. Snapshots at $t = 0, 30, 60$ clearly show wrap-around and mass preservation.

Physics is simple — but violations are immediately visible.

CNN Emulator without Mass Conservation

Neural model

- Local 1D CNN with circular padding
- Learns one-step map $u^n \mapsto u^{n+1}$
- Trained by minimizing one-step MSE

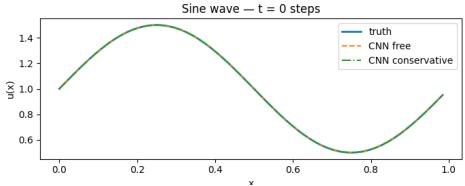
What is not enforced

- No mass conservation constraint
- No global invariant control

Observed behavior

- Low training loss
- Smooth short-term evolution
- Gradual drift in total mass

Locality alone is not enough to guarantee physical correctness.



CNN without conservation. Small amplitude and mass errors accumulate despite visually plausible transport.

CNN Emulator with Exact Mass Conservation

Conservative update

The CNN predicts a residual update

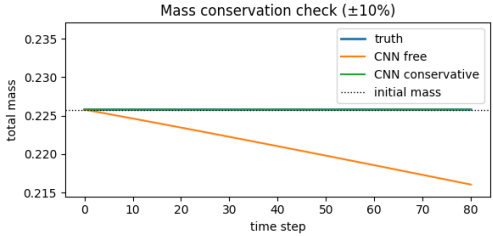
$$u^{n+1} = u^n + \Delta u,$$

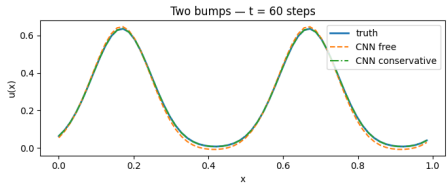
with the constraint $\sum_i \Delta u_i = 0$.

- **Mass conservation enforced by construction**
- Same architecture and data as unconstrained CNN
- No penalty tuning required

Effect

- Slightly higher one-step loss
- Exact preservation of the global invariant
- Significantly improved long-term behavior





CNN with mass conservation. The global invariant is preserved exactly.

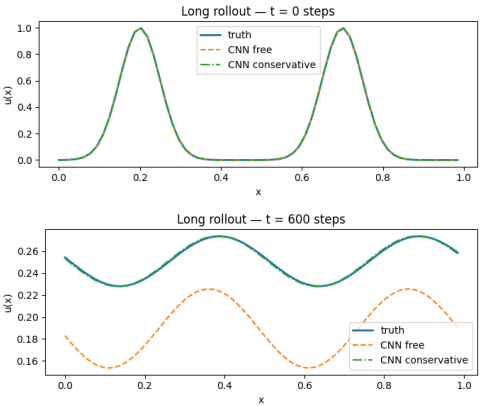
Long-Time Rollout: Accuracy vs Physical Validity

Long-time test

- Rollout over hundreds of time steps
- Same initial condition
- Compare truth, CNN-free, CNN-conservative

Key observation

- CNN-free: errors accumulate steadily
- CNN-conservative: structure remains coherent
- Physics constraints matter most far beyond training horizon



Short-term accuracy is not a proxy for long-term validity.

Only the conservative model maintains physically consistent transport.

Generalization: Advection of Unseen Shapes

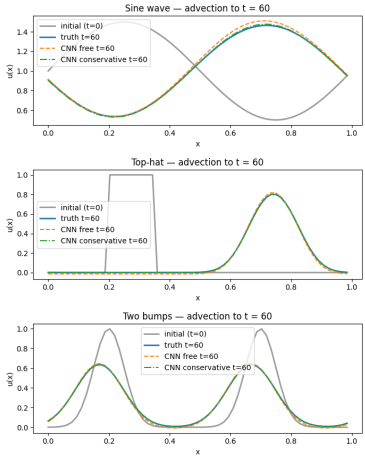
What was trained

- CNN emulator trained only on Gaussian initial conditions
- Local architecture with periodic padding
- Conservative variant enforces exact mass preservation

We test: Sine waves, Top-hat functions (discontinuous), Multiple separated bumps.

What we observe

- Correct translation on the periodic domain
- Shapes are transported without spurious creation or loss of mass
- Superpositions (multiple bumps) are handled consistently



Examples of unseen initial conditions advected by the CNN emulator.

Correlation vs Causality in Dynamical Systems

The trap

- Many variables in geosciences are strongly correlated
- But correlation alone cannot identify directionality
- Hidden drivers (confounders) can create spurious links

Causal question

$$P(T \mid P) \quad \neq \quad P\big(T \mid do(P)\big)$$

Why time series are special

- Dynamics introduce time ordering
- Causal effects typically appear with lags
- Strong autocorrelation can mask cross-effects

Key idea

- Use lagged dependencies to separate:
  - self-dynamics (memory)
  - cross-variable influences

In Earth-system applications this matters because

- interventions (what-if) require causal structure
- attribution needs confounding control
- robust extrapolation benefits from causal mechanisms

Take-home message:

similar correlations do not imply same physics

Two Physical Processes Behind the Same Correlation

Scenario 1: direct causal coupling

- Pressure evolves under synoptic forcing
- Temperature responds via adiabatic processes

$$\frac{dP}{dt} = -\gamma_P(P - P_{eq}) + \beta_P F(t)$$
$$\frac{dT}{dt} = -\gamma_T(T - T_{eq}) + \alpha(P - P_{eq})$$

True causal link: $P \to T$

Scenario 2: common external forcing

- Air-mass advection drives both variables
- No direct physical coupling between P and T

$$\frac{dP}{dt} = -\gamma_P(P - P_{eq}) + \beta_P F(t)$$
$$\frac{dT}{dt} = -\gamma_T(T - T_{eq}) + \beta_T F(t)$$

No direct causal link: $P \not\to T$

Both scenarios can show similar P–T correlation — but they are physically different.

Baseline: Classical Statistical Causal Analysis

Naive statistical analysis

- correlation: $\mathrm{corr}(P, T)$
- regression: $T \sim P$

Problem

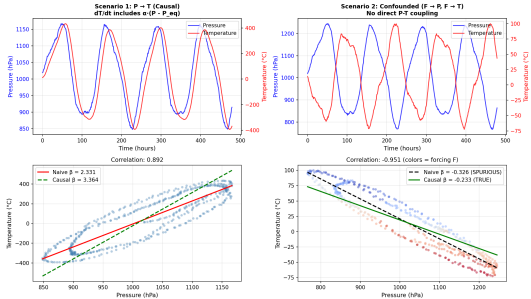
- in Scenario 2, regression suggests $P \to T$
- but correlation is spurious (confounded by F)

Improvement (if forcing known)

- control for confounder:

$$T \sim P + F$$

- can recover causal effect if model is correct



Simulation diagnostics. Time series + scatter: naive regression can be misleading when $F(t)$ induces common variability.

Limitation: Confounders must be known and included (strong prior assumptions).

Causal Discovery from Time Series: PCMCI (Classical Method)

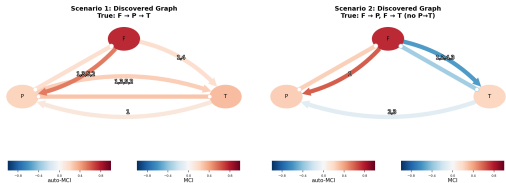
Goal

- learn the directed graph from multivariate time series
- distinguish:
  - direct causal links
  - common forcing and confounding

PCMCI in one line

- PC algorithm + conditional independence tests
- works with lagged dependencies
- reduces false links by conditioning on relevant parents

Important: PCMCI is statistical causal discovery, not neural.



Discovered graphs (PCMCI). Left: Scenario 1 matches the chain $F \to P \to T$. Right: Scenario 2 reveals no direct $P \to T$ link.

Message: same correlation $\neq$ same causal structure.

Key capability

- Scenario 1: recover $F \to P \to T$
- Scenario 2: recover $F \to P$ and $F \to T$ only

Neural Causal Discovery: Scaling Ideas to Complex Systems

Motivation

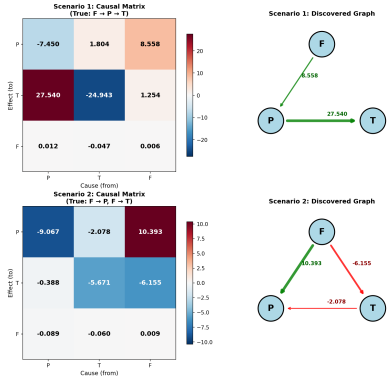
- Earth-system variables are high-dimensional
- dependencies are nonlinear and state-dependent
- autocorrelation dominates raw signals

Neural causal discovery idea

- first remove self-dependence (autocorrelation)
- learn cross-effects from innovations
- enforce sparsity $\Rightarrow$ interpretable directed links

Innovation form: $x(t) - a\,x(t-1)$

Key message: AI helps to extend classical causal ideas to nonlinear, high-dimensional dynamics.



Example result. After removing autocorrelation, sparse learning recovers directed links: Scenario 1: $F \to P \to T$, Scenario 2: $F \to P$, $F \to T$.