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 20 — Obs-to-Obs Learning on a 2D Toy Atmosphere

Goal of this lecture

- Build a simple 2D dynamical system $\phi(x, z, t)$ with transport + diffusion + source
- Generate two observation types
  - **Radiosondes (RS):** sparse vertical profiles at a few columns
  - **Satellite (SAT):** integrated vertical weighted observations for every column
- Train a neural network to predict next-step observations

$$(y_t^{sat},\ y_t^{rs}) \ \mapsto\ (y_{t+1}^{sat},\ y_{t+1}^{rs})$$

- Reconstruct the full field at $t + 1$ by querying RS predictions everywhere

Key message: learn a forecast step in observation space, and still recover a state-like field.

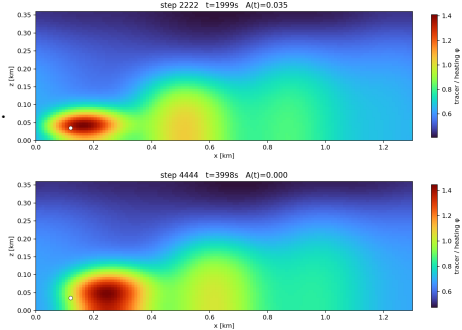
Toy Dynamics: Advection–Diffusion with Source and Damping

We simulate a tracer/heating field $\phi(x, z, t)$ on a 2D domain:

$$(x, z) \in [0, L_x] \times [0, L_z].$$

Dynamics (PDE)

$$\frac{\partial \phi}{\partial t} = -u\frac{\partial \phi}{\partial x} - w\frac{\partial \phi}{\partial z} + K\nabla^2\phi + A(t)\, S(x, z) - \lambda\, \phi - \lambda_{\text{top}}(z)\, \phi.$$



Key design choices

- Mean wind: (u, w) right + upward $\Rightarrow$ visible transport
- Source $S(x, z)$: localized bottom-left heating region
- Pulsed forcing $A(t) \Rightarrow$ visible trace stripes
- Damping λ + top sponge $\lambda_{\text{top}} \Rightarrow$ equilibrium

Time stepping: RK4.

Boundary Conditions: Wrap-Around Transport (Periodic in x)

We want a flow where structures move out to the right and re-enter from the left.

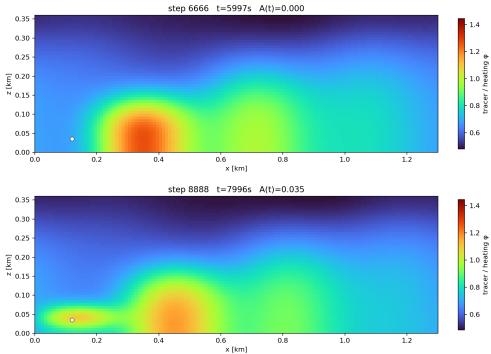
Boundary conditions

- Periodic in x: outflow wraps around
- Reflective/top treatment in z (plus sponge)

Effect

- A persistent tracer train forms
- The system reaches a statistical steady state
- Ideal for learning time transitions

This creates a clean, cyclic "atmosphere" for ML demos.



RK4 Implementation (Core Loop)

Right-hand side

$$\mathrm{RHS}(\phi, t) = -u\,\partial_x\phi - w\,\partial_z\phi + K\nabla^2\phi + A(t)S(x,z) - (\lambda + \lambda_{\mathrm{top}})\phi.$$

One RK4 step (schematic)

```
k1 = rhs(phi, t)
k2 = rhs(phi + 0.5*dt*k1, t + 0.5*dt)
k3 = rhs(phi + 0.5*dt*k2, t + 0.5*dt)
k4 = rhs(phi + dt*k3,     t + dt)
phi_next = phi + (dt/6)*(k1 + 2*k2 + 2*k3 + k4)
```

Snapshot logic

- Choose n_{vis} time indices between 0 and *nsteps*
- Save numbered PNGs: `1_dyn_XX.png`

This gives clean training snapshots + nice lecture figures.

Energy / Mass Budget Monitoring (Sanity Checks)

We monitor injected heat content and losses:

$$C(t) = \int \phi \, dx \, dz, \qquad I(t) = \int A(t) S \, dx \, dz, \qquad L(t) = \int (\lambda + \lambda_{top}) \phi \, dx \, dz.$$

Cumulative budget (discrete)

$$\text{acc\_in} = \sum_n I(t_n) \Delta t, \qquad \text{acc\_loss} = \sum_n L(t_n) \Delta t.$$

- $\int \phi$ stabilizes near equilibrium
- injected vs lost energy becomes balanced
- confirms numerical stability and correct forcing/damping design

Always build monitoring into the toy model: it prevents silent nonsense.

Observations: Radiosondes (Sparse Vertical Profiles)

Radiosonde observation operator

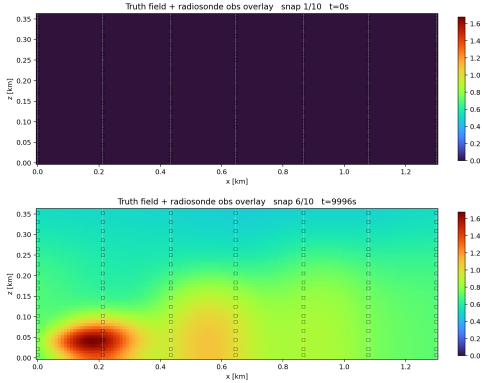
At station column x_s and selected heights z_k:

$$y_t^{rs}(s,k) = \phi(x_s, z_k, t) + \epsilon.$$

Geometry is discrete

- n_{rs} station columns: `rs_ix`
- n_{vert} vertical levels: `rs_iz`
- output array: `yrs[time, station, vert]`

RS gives a sparse but physically intuitive reference view.



Observations: Satellite (Vertical Weighting Integrals)

Satellite observation operator

For each channel c (Gaussian vertical weights $w_c(z)$):

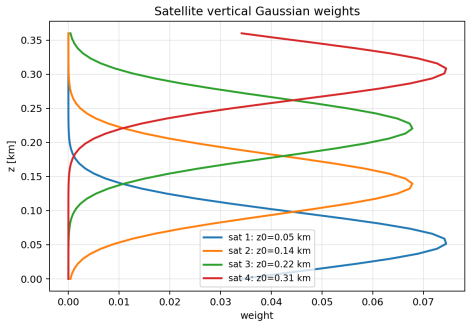
$$y_t^{sat}(c, x_i) = \sum_{j=1}^{n_z} w_c(z_j)\, \phi(x_i, z_j, t)\, \Delta z.$$

Properties

- available for all columns x_i
- integrated information $\Rightarrow$ ill-posed inverse problem
- multiple channels $\Rightarrow$ multi-layer sensitivity

Array

- `ysat[time, channel, x]`



Gaussian vertical weighting functions used for the SAT channels.

Saved Dataset: Truth + Two Observation Types

We store everything into `dyn_truth_obs.npz`:

- **Truth fields**

$$xtrue \in \mathbb{R}^{T_{snap} \times n_z \times n_x}$$

- **Radiosondes**

$$yrs \in \mathbb{R}^{T_{snap} \times n_{rs} \times n_{vert}}$$

- **Satellite**

$$ysat \in \mathbb{R}^{T_{snap} \times n_{sat} \times n_x}$$

- Snapshot times: `t_snap`
- Geometry: `rs_ix`, `rs_iz`
- SAT vertical weights: `sat_w`

This allows a clean separation: Notebook 1 = generate dataset, Notebook 2 = learn transitions.

Learning Task: Next-Step Obs Forecasting

We train an ML model that predicts the next observation state:

$$(y_t^{sat},\ y_t^{rs}) \ \mapsto\ (\hat{y}_{t+1}^{sat},\ \hat{y}_{t+1}^{rs}(\cdot)).$$

Inputs at time t

- Satellite curtain: $y_t^{sat}(c, x)$ for all columns
- RS set of points: $\{(x_m, z_m, y_m)\}_{m=1}^{N_{in}}$

Outputs at time $t+1$

- Predicted satellite curtain $\hat{y}_{t+1}^{sat}(c, x)$
- Predicted RS values at query points $\hat{y}_{t+1}^{rs}(x_q, z_q)$

Important: query points can be anywhere $\Rightarrow$ generalization to new RS placements.

Normalization: The Key Practical Ingredient

We normalize observations before training:

SAT: per-channel statistics

$$y_n^{sat} = \frac{y^{sat} - \mu_{sat}}{\sigma_{sat}}, \qquad \mu_{sat} = \mathbb{E}_{t,x}[y^{sat}], \qquad \sigma_{sat} = \mathrm{std}_{t,x}[y^{sat}].$$

RS / truth: global

$$\phi_n = \frac{\phi - \mu_\phi}{\sigma_\phi}, \qquad y_n^{rs} = \frac{y^{rs} - \mu_\phi}{\sigma_\phi}.$$

Without normalization the training becomes unstable and "learns the wrong scale".

Flexible RS Input: Two Dataset Modes (A/B Switch)

We implement a dataset switch (important for interpretability):

Mode A (synthetic RS from truth)

- sample RS input points randomly from $xtrue[t]$
- add realistic noise σ_{rs}
- excellent generalization for arbitrary RS geometry

Mode B (use only stored RS observations)

- RS input points are exactly `yrs[t,:,:]` at fixed `rs_ix, rs_iz`
- strict statement: **"input uses only observations at time t"**
- weaker geometric diversity, but realistic RS network

Reality: We have only fixed radiosondes, but we have airplanes !

```
Saved:
  data_dyn_obs/dyn_truth_obs.npz

Content:
  xtrue   shape=(10,50,130)
  yrs     shape=(10,7,18)
  ysat    shape=(10,4,130)

  t_snap  shape=(10,)
  rs_ix   shape=(7,)
  rs_iz   shape=(18,)
  sat_w   shape=(4,50)

  x       shape=(130,)
  z       shape=(50,)
```

Architecture: CNN on SAT + Set Encoder for RS + Query Head

Why a CNN?

SAT is a curtain $y^{sat}(c, x)$ along x.
Spatial patterns advect $\Rightarrow$ translation-like structure.

Components

- **SAT encoder:** 1D-CNN
  - extracts local features along x
- **RS encoder:** DeepSets / pooling
  - handles variable-size RS point sets
- **Query decoder:** predicts $y^{rs}_{t+1}(x_q, z_q)$

This enforces coupling: RS inference is driven by SAT structure.

Query-style output

$$\hat{y}^{rs}_{t+1}(x_q, z_q) = g_\theta\big(\mathrm{CNN}(y^{sat}_t),\ \mathrm{Set}(y^{rs}_t),\ x_q, z_q\big).$$

Practical win

- RS can be placed anywhere
- same model predicts:
  - sparse profiles
  - full fields (query everywhere)

DeepSets / pooling: why order does not matter

RS inputs form a *set* $\mathcal{S} = \{(x_m, z_m, y_m)\}_{m=1}^{N}$. A set has no order, so the encoding must satisfy $F(\mathcal{S}) = F(\pi(\mathcal{S}))$ for any permutation π.
We enforce this with a permutation-invariant encoder:

$$e_m = \psi_\theta(x_m, z_m, y_m), \qquad E = \mathrm{pool}(e_1, \ldots, e_N)$$

where pooling is sum/mean/max (commutative $\Rightarrow$ order-invariant). Locations matter via (x_m, z_m) inside ψ_θ.

Training Loss: Joint SAT + RS Query Targets

We train on snapshot transitions $t \to t+1$ using a joint loss:

SAT loss (next curtain)

$$\mathcal{L}_{sat} = \|\hat{y}^{sat}_{t+1} - y^{sat}_{t+1}\|_2^2.$$

RS query loss (next profile at query points)

$$\mathcal{L}_{rs} = \frac{1}{N_q} \sum_{q=1}^{N_q} \left(\hat{y}^{rs}_{t+1}(x_q, z_q) - y^{rs}_{t+1}(x_q, z_q)\right)^2.$$

Total

$$\mathcal{L} = \mathcal{L}_{sat} + \alpha\, \mathcal{L}_{rs}.$$

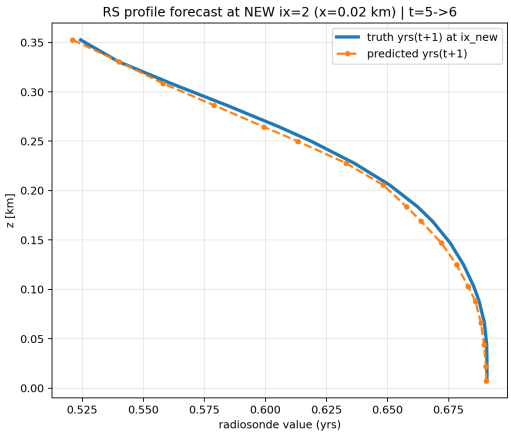
Note: RS targets for arbitrary (x_q, z_q) are taken from truth $xtrue[t+1]$.

Evaluation 1: Predict a New RS Profile at an Unseen Location

We test generalization:

- Input RS: taken from stored network `yrs[t,:,:]`
- Choose a new x column not used by RS
- Query all heights $\{z_q\}$ at that x
- Compare predicted profile $\hat{y}^{rs}_{t+1}$ vs truth $xtrue[t+1]$

This checks whether the model learned physics-like transport information from SAT.



Evaluation 2: Full-Field Reconstruction at $t+1$ (Query Everywhere)

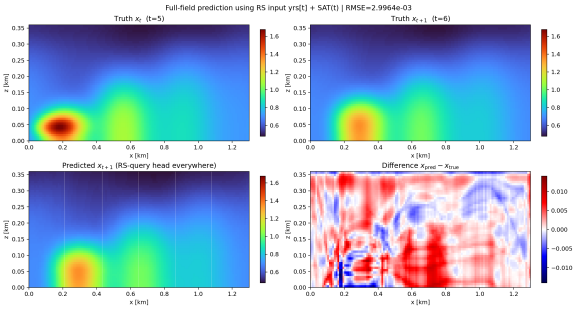
The RS query head can be evaluated on the whole grid:

$$x_{\mathrm{pred}}(x_i, z_j, t+1) := \hat{y}^{rs}_{t+1}(x_i, z_j).$$

We visualize

- $xtrue[t]$ (reference)
- $xtrue[t+1]$ (truth)
- $xpred[t+1]$ (prediction)
- difference $xpred - xtrue$

This turns obs-to-obs learning into a state-like field prediction.



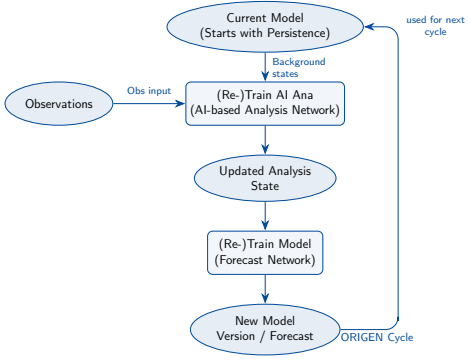
Observation-based Reconstruction & Inversion for Generative Emulation of Nonlinear Systems: ORIGEN

- A closed learning cycle linking observations, analysis reconstruction, and forecast models.

Conceptual loop

- Start from a current model (here: persistence / baseline)
- Use observations to reconstruct state: AI analysis / inversion
- Update the model/forecast emulator from reconstructed states
- Iterate: next cycle uses the new model

ORIGEN cycle schematic



ORIGEN on a Minimal Example: Simple Oscillator

Minimal testbed (2D circle)

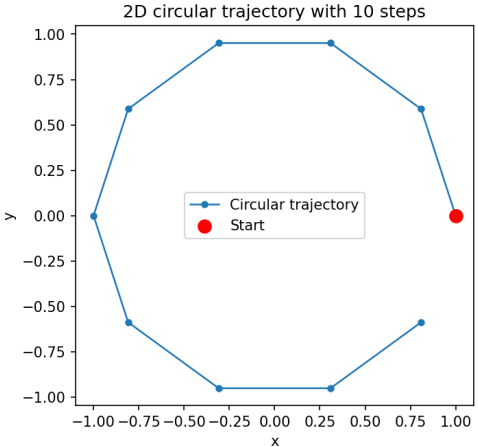
- Truth trajectory $x_k = (x_{1,k}, x_{2,k})$ on a circle

$$x_k = \begin{bmatrix} \cos\theta_k \\ \sin\theta_k \end{bmatrix}, \qquad \theta_k = \frac{2\pi k}{n}$$

- Observations are scalar and partial:

$$y_k = H_k x_k + \epsilon_k, \qquad H_k \in \{[1,0],[0,1]\}$$

Aim: Explain ORIGEN mechanics without complex dynamics.



Observations: Partial Measurements of x_1 or x_2

Time-dependent observation operator

- At each step k we observe only one component
- Selector:

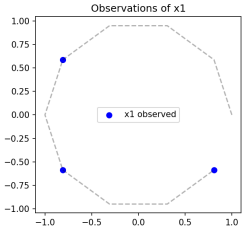
$$s_k \in \{1, 2\}$$

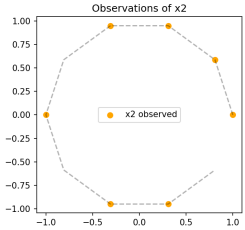
- Observation operator:

$$H_k = \begin{cases} [1, 0] & s_k = 1 \text{ (observe } x_1\text{)} \\ [0, 1] & s_k = 2 \text{ (observe } x_2\text{)} \end{cases}$$

- Observation equation:

$$y_k = H_k x_k + \epsilon_k, \qquad \epsilon_k \sim \mathcal{N}(0, R)$$





Blue: x_1 observed Orange: x_2 observed

Interpretation: This mimics heterogeneous sensors and missing data.

3D-Var Step: Background → Analysis (Single Observation)

We combine background + one scalar obs

- Background state: $x_k^b \in \mathbb{R}^2$
- Obs: $y_k \in \mathbb{R}$, operator $H_k \in \mathbb{R}^{1\times 2}$

Innovation

$$d_k = y_k - H_k x_k^b$$

Gain (scalar obs)

$$K_k = BH_k^\top \left(H_k B H_k^\top + R\right)^{-1}$$

Analysis update

$$x_k^a = x_k^b + K_k\, d_k$$

Effect: update only in observed direction, but shaped by B.

Initial choice of covariances

$$B = \begin{bmatrix} \sigma_b^2(x_1) & 0 \\ 0 & \sigma_b^2(x_2) \end{bmatrix}, \qquad R = \sigma_o^2$$

Cycling (persistence model)

$$x_{k+1}^b = M(x_k^a), \qquad M(x) = x$$

- This produces a full sequence:

$$x_0^b \to x_0^a \to x_1^b \to \dots$$

- We visualize results next: truth vs background vs analysis

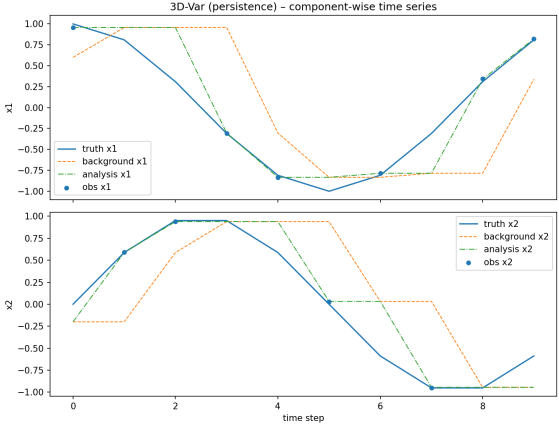
3D-Var Cycle Result: Time Series of x_1 and x_2

What we compare

- Truth x_k
- Background x_k^b
- Analysis x_k^a
- and the scalar observations y_k

Key behavior

- analyses are pulled towards the obs
- cycling propagates improvements forward
- unobserved component still benefits indirectly (via B and cycling)



Top: x_1 time series Bottom: x_2 time series

Idea: Information is collected iteratively.

Sequential Rounds: With All Observations We Converge in Two Cycles

Idea: assimilate in two rounds

- Round 1: observe x_1 only

$$y_k^{(1)} = x_{1,k} + \epsilon_k$$

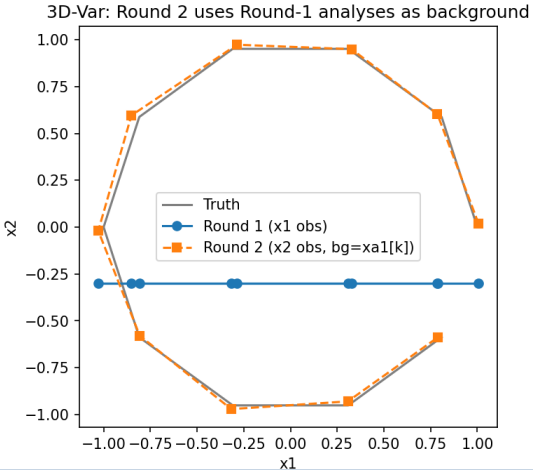
- Round 2: observe x_2 only, using

$$x_k^{b,(2)} := x_k^{a,(1)}$$

If obs error is small

- after Round 1: x_1 aligns with truth
- after Round 2: x_2 aligns with truth
- both components observed $\Rightarrow$ fast convergence

Message: With complete information, ORIGEN-style cycling converges quickly.



Iterative 3D-Var: Reconstruction Improves over Cycles (Demo)

What is iterated here?

- We reconstruct an entire trajectory

$$\{x_k^a\}_{k=0}^{n-1}$$

from noisy, partial observations

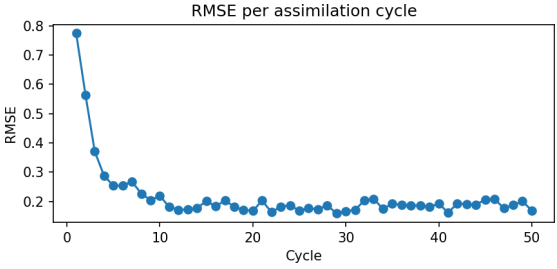
$$y_k = H_k x_k + \epsilon_k$$

- Each cycle produces a refined estimate of the full sequence:

$$\{x_k^{a,(c)}\} \Rightarrow \{x_k^{a,(c+1)}\}$$

Crucial point

- The "dynamics" is implicit in the reconstructed series
- We are not propagating with a separate model; we repeatedly improve the sequence itself



RMSE per assimilation cycle: iterative 3D-Var improves the reconstructed trajectory

ORIGEN: from the reconstructed $\{x_k^a\}$ we learn a forecast map

$$x^a(t) \mapsto x^a(t + \Delta t),$$

in the notebook we focus on trajectory reconstruction.

Reconstruction: 2D Trajectory over Iterative 3D-Var Cycles

Iterative 3D-Var reconstruction

- Each cycle reconstructs the full sequence:

$$\{x_k^a\}_{k=0}^{n-1}$$

- Observations remain noisy and partial:

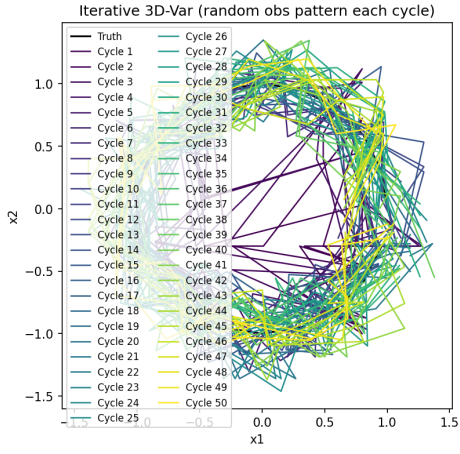
$$y_k = H_k x_k + \epsilon_k$$

- Repeated cycles reduce reconstruction error

What you see on the right

- Truth trajectory (black)
- Selected analysis cycles (colored)
- Later cycles are closer to truth

Message: 3D-Var keeps error coming in from the observation error



Reconstruction: Cycle Time Series with Noisy Observations

Final-cycle diagnostics

- Compare truth vs final reconstructed trajectory
- Show observations for the last random pattern:

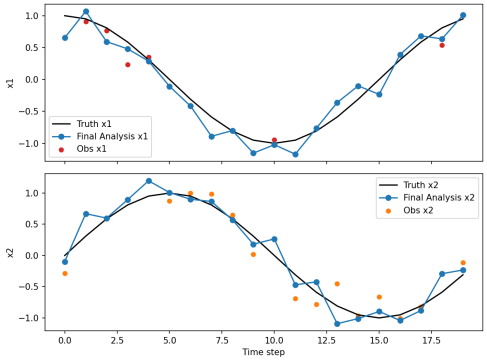
$$H_k \in \{[1,0],[0,1]\}$$

- Observations are noisy $\Rightarrow$ analysis does not overfit

3D-Var property

- analysis is a weighted compromise
- controlled by B and R

Message: 3D-VAR is not enough



Final cycle: time series for x_1 and x_2 with noisy partial observations

Adding a Covariance Update: Kalman Filter–Type Uncertainty Reduction

Upgrade: update B during cycling

- Until now: fixed background covariance B
- Now: after each analysis step we also update uncertainty

Kalman filter covariance update

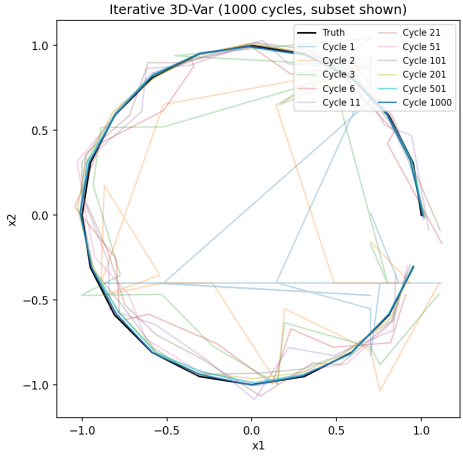
$$A_k = (I - K_k H_k)\, B_k \qquad \Rightarrow \qquad B_k \leftarrow A_k$$

Effect:

- uncertainty shrinks in observed directions
- gain adapts across cycles
- analysis trajectory converges strongly to truth

Message:

With adaptive B, iterative DA converges.



Full Convergence: Final Trajectory with Covariance Update

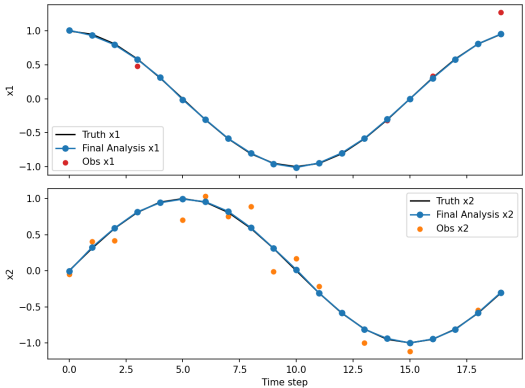
Final-cycle reconstruction

- With B update, repeated cycling yields near-perfect reconstruction
- Even with noisy and partial observations

Why it converges

- analysis reduces both error and uncertainty
- smaller uncertainty $\Rightarrow$ more consistent updates
- loop stabilizes around the truth trajectory

Message: ORIGEN principle: reconstruction + uncertainty adaptation $\Rightarrow$ convergence.



Final cycle: time series x_1, x_2 converged

Lorenz-63: ORIGEN Reconstruction & Learned Forecast Model

ORIGEN loop (concept)

1. Assimilate random observed subsets (x, y, z, xy, xz, yz)
2. Update background $\mathbf{x}^b \leftarrow \mathbf{x}^a$
3. Update covariance $B \leftarrow P^a$ (Kalman-style)

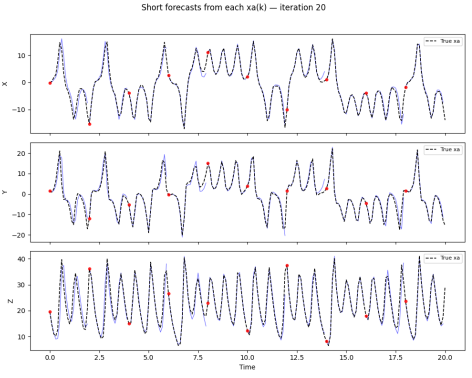
Model learning

- Train NN to learn one-step map

$$\mathbf{x}^a(k) \;\mapsto\; \mathbf{x}^a(k+1)$$

- **Fine-Tuning with Rollout**

Rollout: many subsequent short forecasts



Forecast rollouts (blue) starting from analysis states (red dots), compared to the reconstructed reference trajectory (black dashed).