Using Machine Learning to Predict Chaotic System: A Case Study on Lorenz System

Sandy (Yuelin) Hou

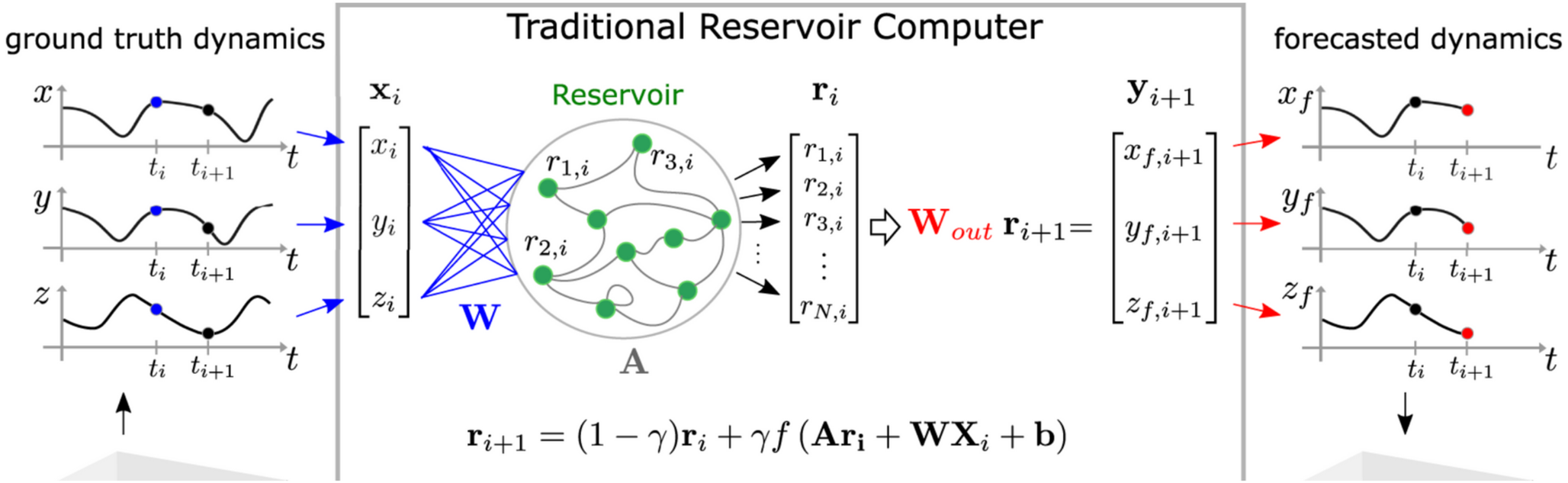
Math 303 ODE and Dynamic Systems

Outline

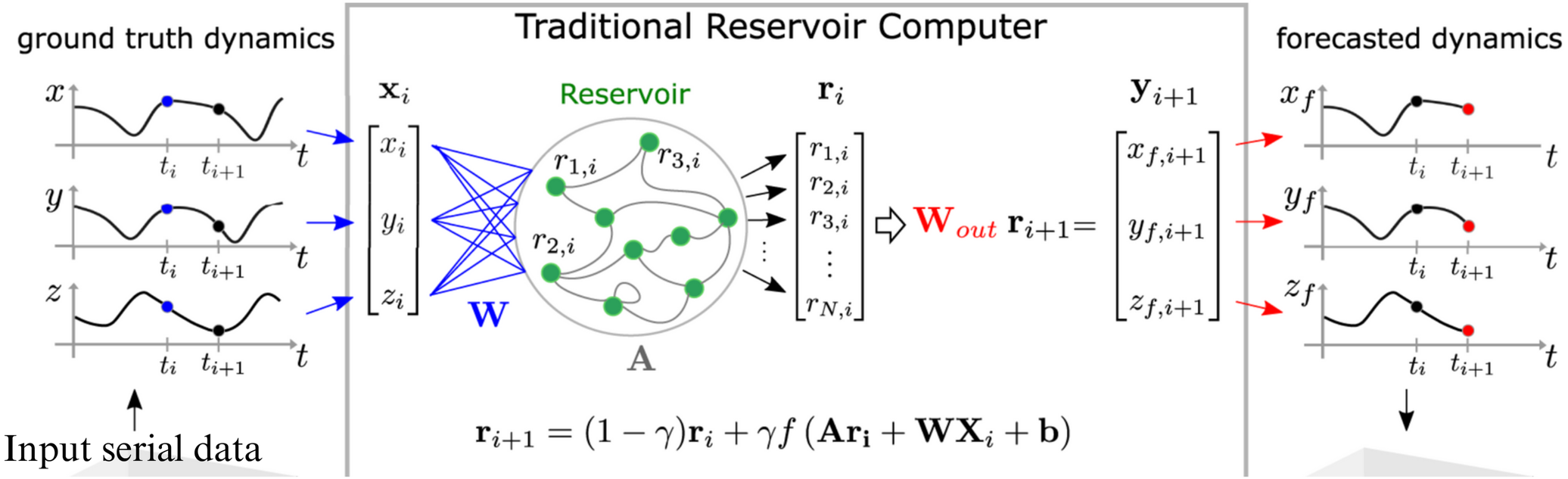
Reservoir Computing

Example: Lorenz System

Reservoir Computing



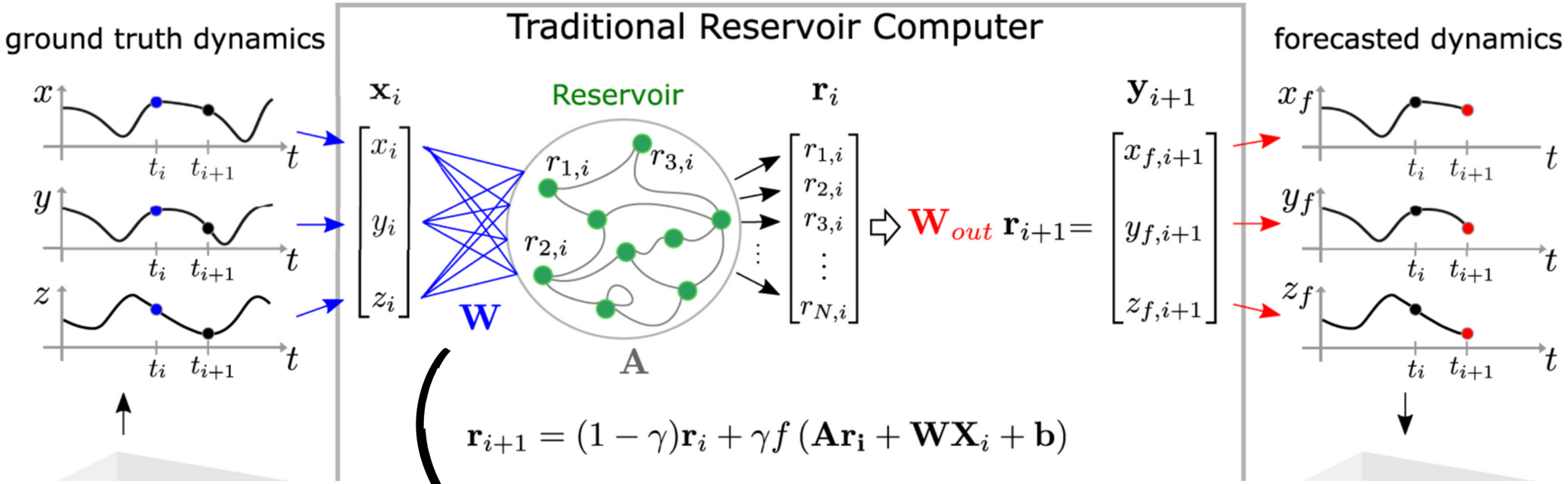
Reservoir Computing



Random, fixed input weights

Random, fixed recurrent connection

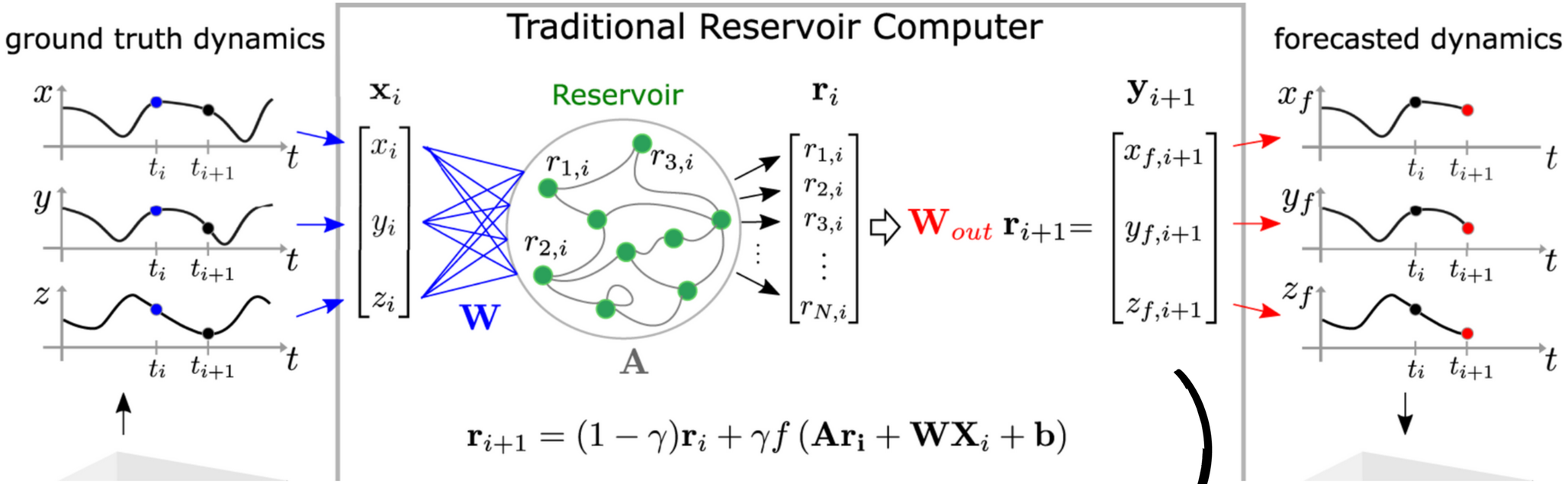
Reservoir Computing



$$r_i(t+\Delta t) = \tanh\left(\sum_{j=1}^{D_r} A_{ij} r_j(t) + \sum_{k=1}^{D} (W_{\text{in}})_{ik} u_k(t) + b_i\right)$$

$$\mathbf{r}(t+\Delta t) = \tanh\left(A\mathbf{r}(t) + W_{\text{in}}\mathbf{u}(t) + \mathbf{b}\right)$$

Reservoir Computing



$$\sum_{-T \le t \le 0} ||\mathbf{W}_{out}(\mathbf{r}(t), \mathbf{P}) - \mathbf{v}_d(t)||^2 + \beta||\mathbf{P}||^2,$$

$$\mathbf{v}(t) = W_{\text{out}}\mathbf{r}(t)$$

Outline

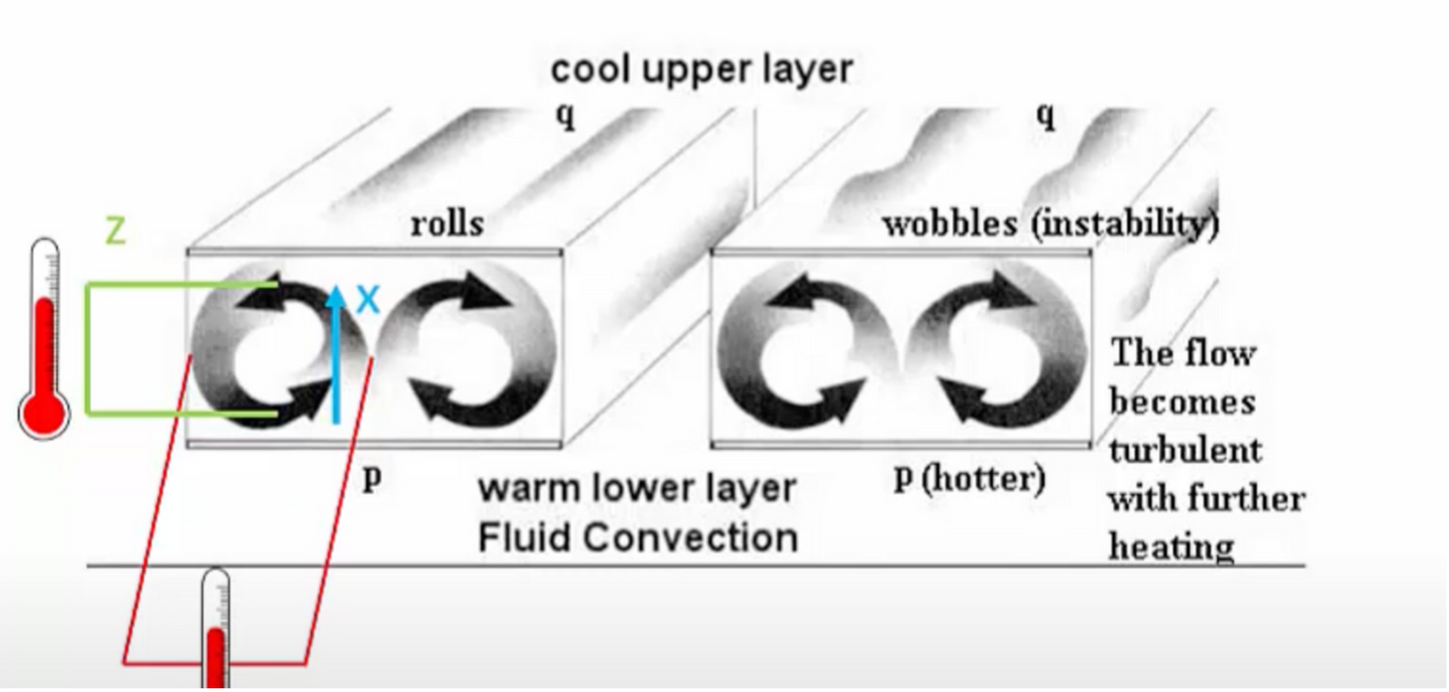
Reservoir Computing

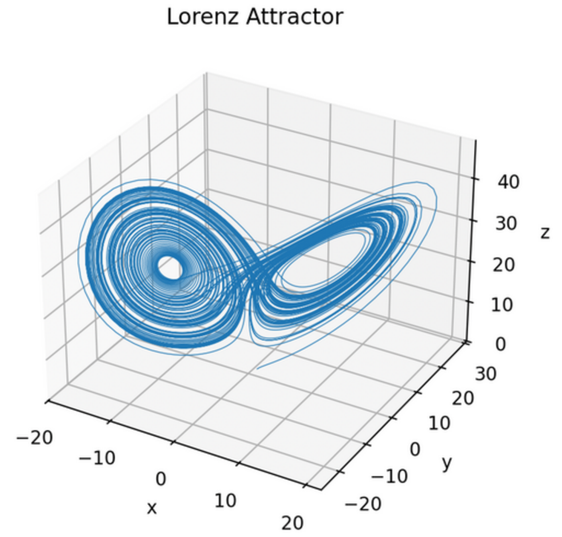
Example: Lorenz System

Lorenz System:

A set of three coupled, nonlinear ordinary differential equation

$$\dot{x} = 10(y - x),$$
$$\dot{y} = x(28 - z) - y,$$
$$\dot{z} = xy - 8z/3.$$





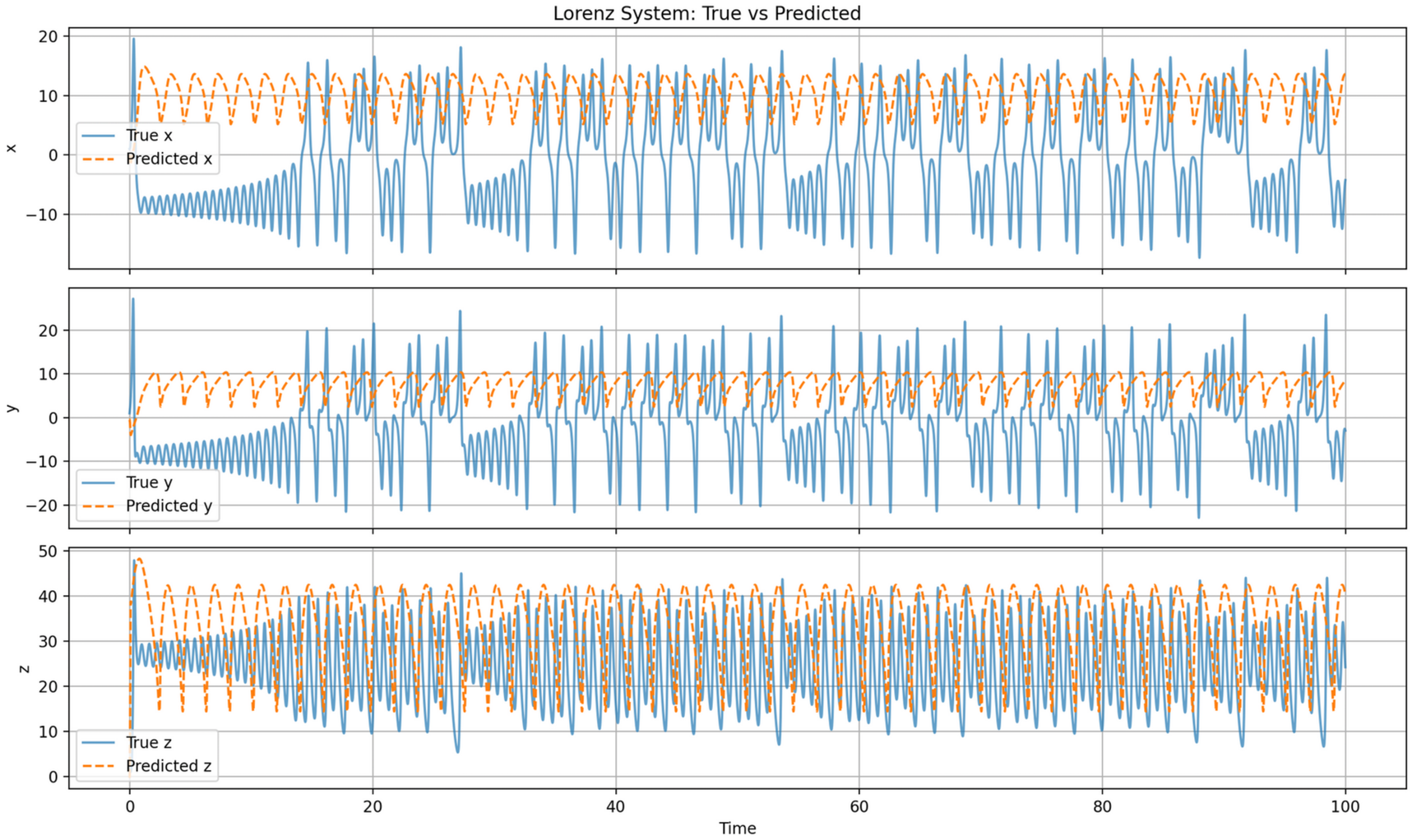
Goal: generate synthetic data integrating the lorenz system and use the data as an input to reservoir

Prediction

```python
def create_reservoir(input_dim, reservoir_size, spectral_radius, input_scaling=0.1, sparsity=0.012):
    A = np.random.rand(reservoir_size, reservoir_size) - 0.5
    mask = np.random.rand(*A.shape) < sparsity
    A *= mask
    eigvals = np.abs(np.linalg.eigvals(A))
    A /= np.max(eigvals)
    A *= spectral_radius

    Win = (np.random.rand(reservoir_size, input_dim) - 0.5) * input_scaling
    return A, Win

def train_reservoir(A, Win, data, lambda_reg=1e-3, washout=500):
    reservoir_size = A.shape[0]
    r = np.zeros(reservoir_size)
    R_collect, U_collect = [], []

    for u in data:
        r_input = A @ r + Win @ u
        r_input = np.clip(r_input, -50, 50)
        r = np.tanh(r_input)
        R_collect.append(r.copy())
        U_collect.append(u.copy())

    R_collect = np.stack(R_collect)
    U_collect = np.stack(U_collect)

    R_train = R_collect[washout:]
    U_train = U_collect[washout:]
    Wout = np.linalg.solve(R_train.T @ R_train + lambda_reg*np.eye(reservoir_size), R_train.T @ U_train)
    return Wout, r
```

```python
def predict(A, Win, Wout, r0, steps):
    r_pred = r0.copy()
    predictions = []

    for _ in range(steps):
        u_pred = Wout.T @ r_pred
        r_input = A @ r_pred + Win @ u_pred
        r_input = np.clip(r_input, -50, 50)
        r_pred = np.tanh(r_input)
        predictions.append(u_pred)

    return np.stack(predictions)
```

Results: Trajectories



Results: Lyapunov Exponents

Lyapunov exponents are characteristic quantities of dynamical systems. For a continuous-time dynamical system, the maximal Lyapunov exponent is defined as follows.

Consider a trajectory x(t), t≥0 in phase space and a nearby trajectory x(t)+δ(t) where δ(t) is a vector with infinitesimal initial length. As the system evolves, track how δ(t) changes. The maximal Lyapunov exponent of the system is the number λ, if it exists, such that

$$|\delta(t)| \approx |\delta(0)| e^{\lambda t}.$$

Every dynamical system has a spectrum of Lyapunov exponents, one for each dimension of its phase space. Like the largest eigenvalue of a matrix, the largest Lyapunov exponent is responsible for the dominant behavior of a system.

Results: Lyapunov Exponents

Comparing the Lyapunov exponents of the Lorenz system

True Lorenz	r Result
$\lambda_1 \sim 0.9$	$\lambda_1 \sim 0.011$
$\lambda_2 \sim 0.0$	$\lambda_2 \sim -2.37$
$\lambda_3 \sim -14$	$\lambda_3 \sim -2.52$

```python
def compute_lyapunov_exponents(A, Win, Wout, r0, num_exponents=3, steps=5000, dt=1.0):
    Dr = A.shape[0]
    r = np.copy(r0)

    perturbations = np.random.randn(Dr, num_exponents)
    q, _ = np.linalg.qr(perturbations)
    perturbations = q

    lyapunov_sums = np.zeros(num_exponents)

    for step in range(steps):
        input_effect = Win @ (Wout.T @ r)
        r_input = A @ r + input_effect
        r_input = np.clip(r_input, -50, 50)
        r = np.tanh(r_input)

        diag = 1 - r**2
        J = np.diag(diag) @ (A + Win @ Wout.T)

        perturbations = J @ perturbations
        q, rmat = np.linalg.qr(perturbations)
        perturbations = q

        lyapunov_sums += np.log(np.abs(np.diag(rmat)) + 1e-10)

    return lyapunov_sums / (steps * dt)
```

- It tells whether the system is chaotic or not.
- It is a fingerprint of the system's dynamics:
  - True chaotic systems have one positive, one zero, and the rest negative exponents.

For Lorenz:

- Positive sensitive dependence (weather system analogy)
- Zero time shift symmetry
- Large negative collapse towards the butterfly-shaped attractor

Future Directions:

Problem: Randomly initialized reservoirs (random matrices) are unstable: some work well, others fail, with little way to predict success.

Move Beyond Random Reservoirs: Design structured reservoirs or replace random reservoirs entirely.

Formalize Universal Approximation Theory: RC + NVARs are now seen as universal approximators for dynamical systems (with bounded orbits). Find practical design rules for feature vectors and training strategies.

Q & A