Reconstructing Chaotic Dynamics with Delay Embedding and Gaussian Process Regression: Applications to Ecological Biomass Data

Joey Park

Thomas Jefferson High School for Science and Technology,
Fairfax, VA, USA

1

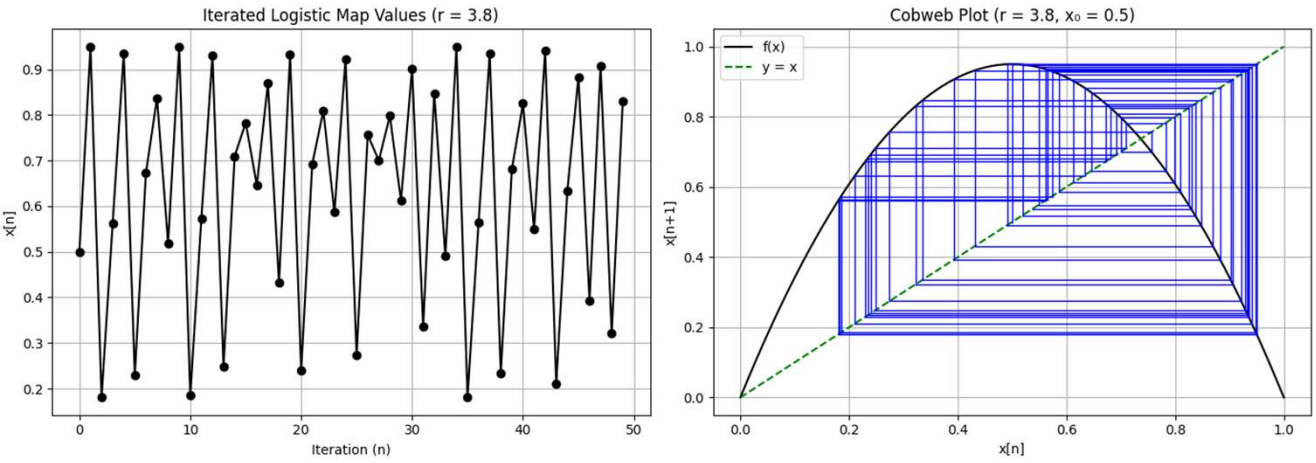
2-1. Explore Logistic Map

- The logistic map is a simple quadratic recurrence relation: $x_{n+1} = rx_n(1 - x_n)$
- Despite its simplicity, it exhibits a wide range of dynamics:
  - Stable fixed points (low r)
  - Period doubling → route to chaos
  - Chaotic behavior (high r)
- Bifurcation diagrams visualize transitions from order to chaos.
- Cobweb plots illustrate the iterative process.
- Serves as a model system for studying complexity and chaotic attractors.
- Foundation for applying tools like delay embedding and Gaussian Process Regression (GPR).

2

Figure 1

- **r = 3.8**
- X[0] = 0.5
- Steps = 50



3

2-2. Delay Embedding + GPR Reconstruction (r = 3.8)

Experimental Setup

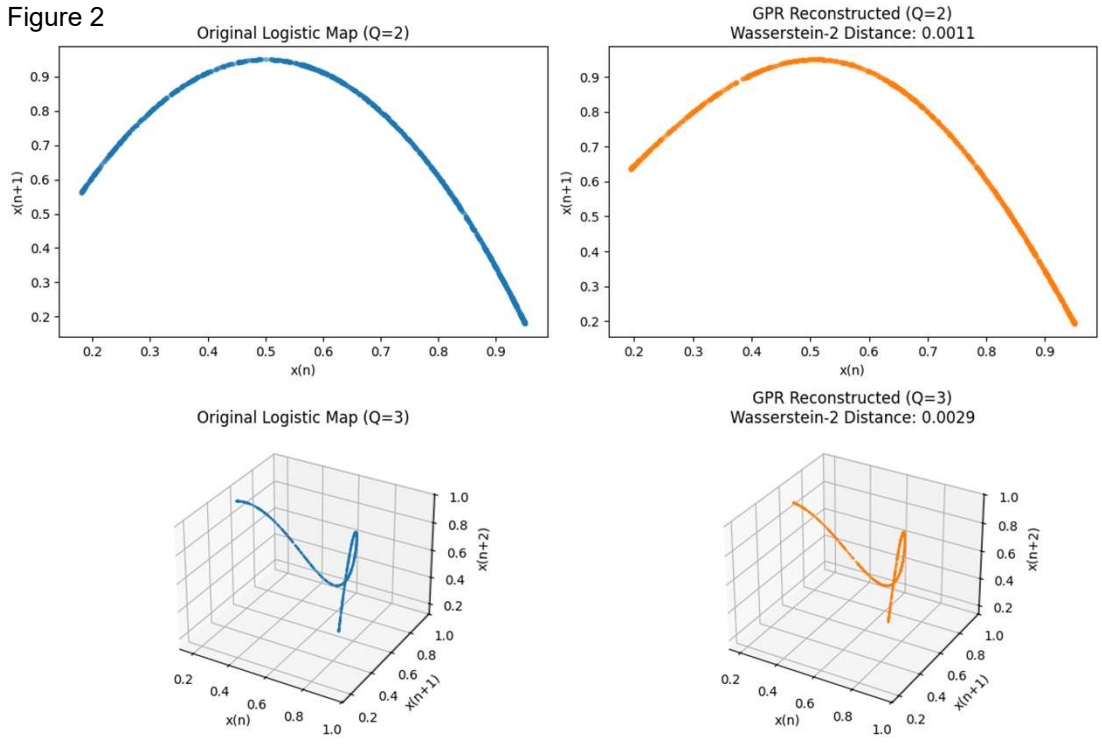
- Parameters: $r = 3.8,\ x_0 = 0.4,\ n = 1000$
- Training size = 10 points only
- Delay embedding with $\tau = 1$
- Embedding dimensions: $Q = 2{,}3$
- GPR regularization: $\alpha = 10^{-4}$
- Goal: Test whether GPR can reconstruct logistic map dynamics from minimal training data.

Takeaway

- Even with only **10 samples**, GPR effectively captures chaotic dynamics.
- Wasserstein distance provides a quantitative measure of closeness between original and reconstructed attractors.
- Higher embedding dimension (Q=3) retains geometric structure but requires more data to reduce error.
- Demonstrates potential of GPR model to approximate complex dynamics from sparse data.

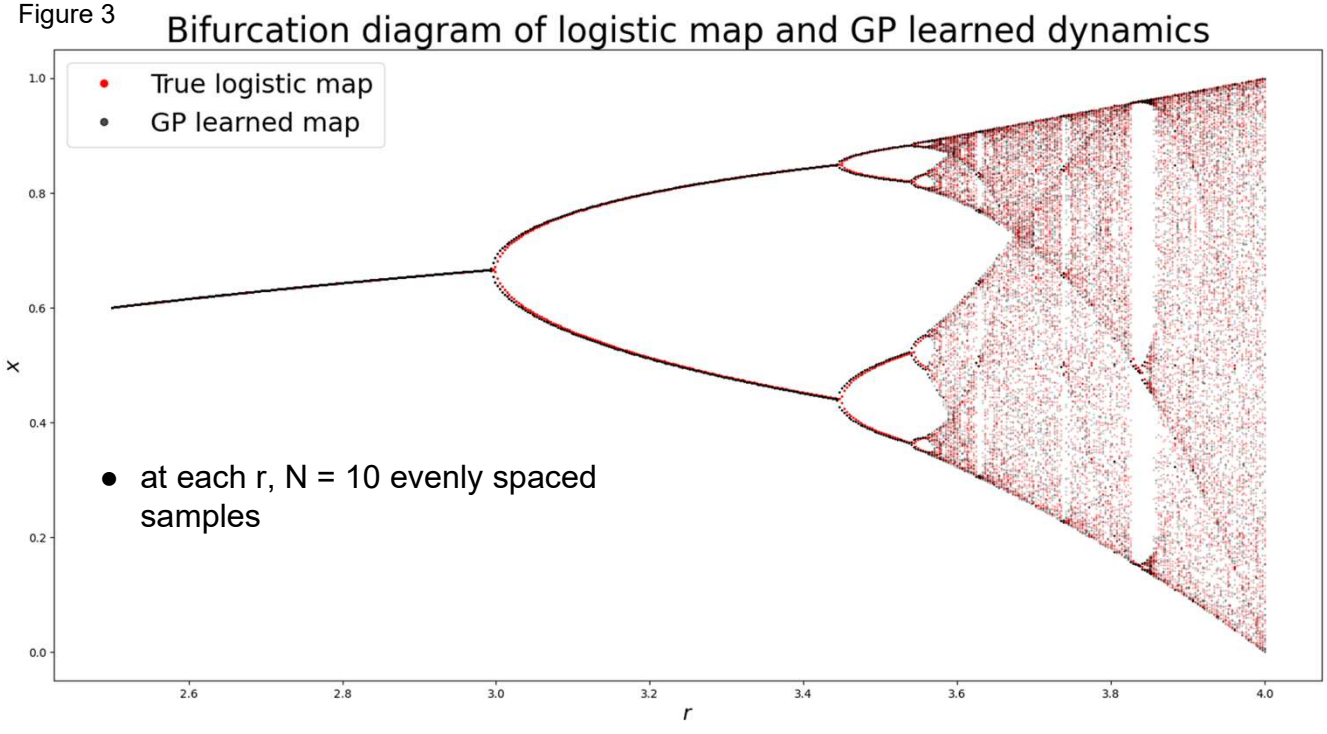
4

Figure 2



5

Figure 3



6

2-3. GPR on Logistic Map with Limited Data & Noise

Setup

- Applied **Gaussian Process Regression (GPR)** with only **5 data points**.
- Tested under different **noise levels** (std = 0.0, 0.02, 0.05, 0.1).
- GPR hyperparameter **α (alpha)** controls noise variance assumption:
  - α = 0 → prediction passes exactly through training points (risk of overfitting).
  - α > 0 → allows smoothing and better generalization.
- Used **α = (noise std)²** to balance data fit vs smoothness.

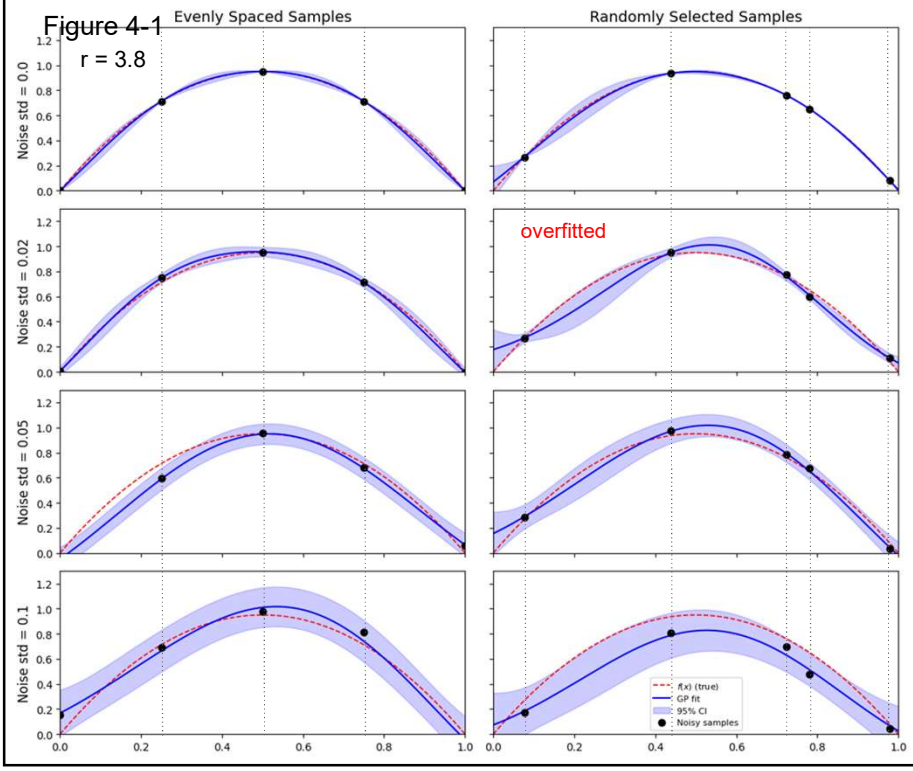
Results

- With α tuned to noise variance, GPR captured the logistic map curve reliably.
- **Even with very sparse samples**, the GP model reconstructed global dynamics.
- Increasing noise widens uncertainty bands (95% CI), but overall structure preserved.

Key Insight

- In **real-world data**, true noise variance is usually unknown.
- Need to **test different α values** to find robust models under uncertainty.

7



Logistic Map with 5 data points

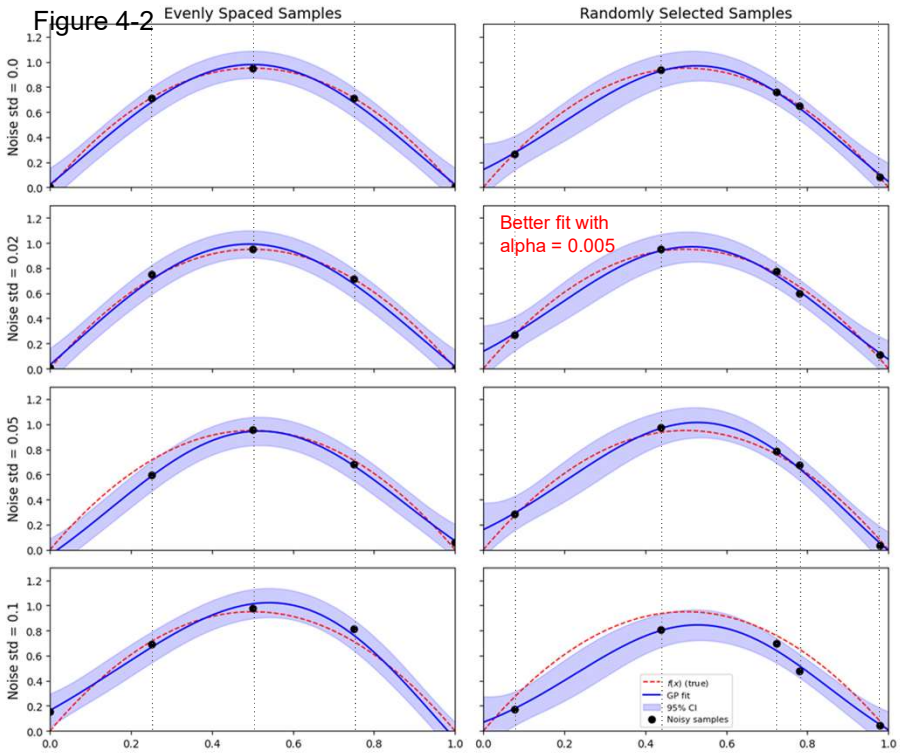
Hyperparameter **alpha** in GPR model: GPR model (sklearn) expect alpha to be the variance of the noise.

- alpha = 0 will force the prediction to go exactly through training points (danger of overfit)
- alpha > 0 will allow smoothing (avoid overfitting)
- I used square of the noise standard deviation for each case for GPR to learn a smooth function that balances data fit.

In real world, I don't think we know the true noise variance. So, I think we should test different alpha with the real-world data.

8

Figure 4-2



Logistic Map with 5 data points

Constant alpha = 0.005 for all the models.

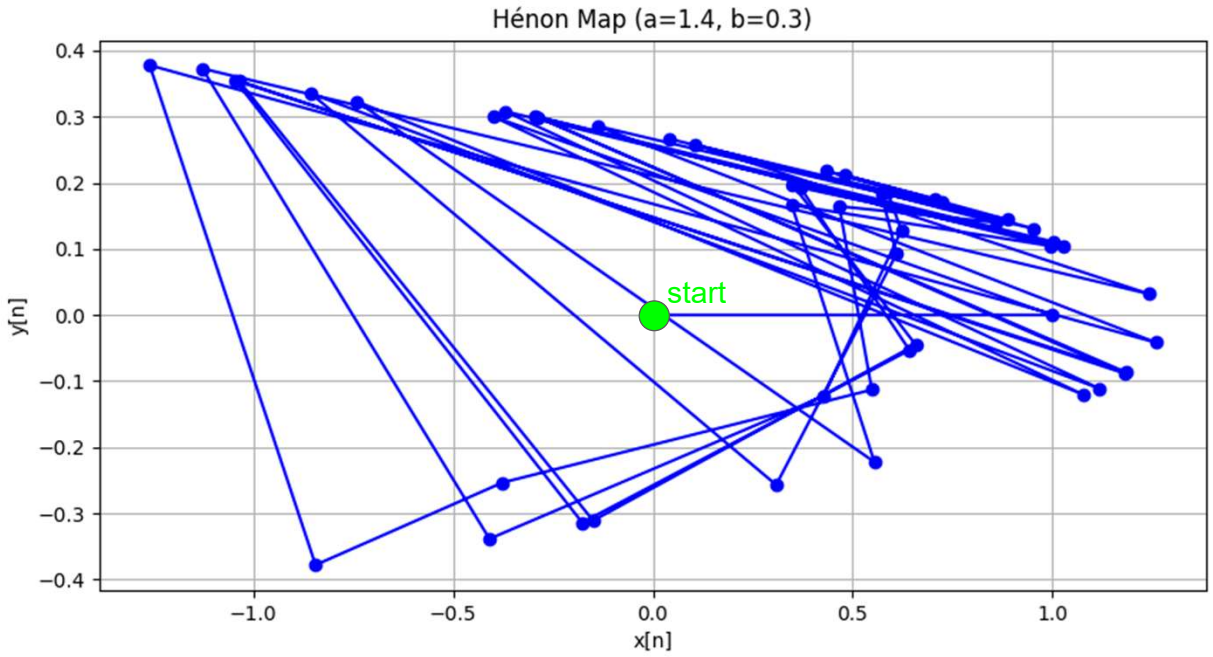
- A larger alpha helped the GPR model generalize better and avoid overfitting to noisy data.

9

Henon Map and measurement function

- a = 1.4, b = 0.3
- Delay (tau) = 1

10

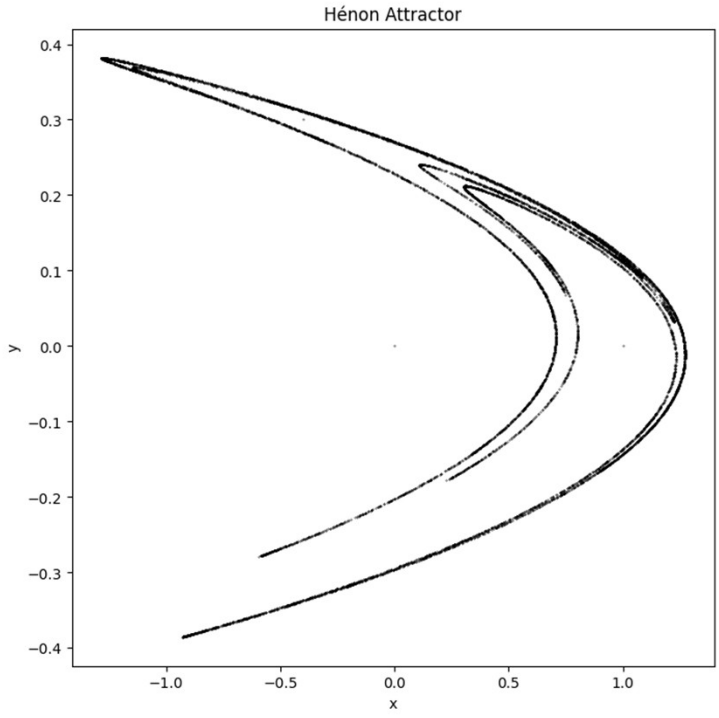


11

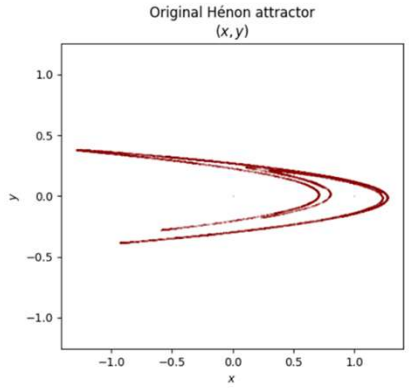
Henon Map

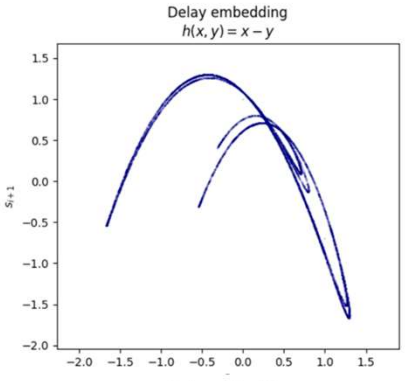
$$\begin{cases} x_{n+1} = 1 - ax_n^2 + y_n \\ y_{n+1} = bx_n. \end{cases}$$

a = 1.4
b = 0.3



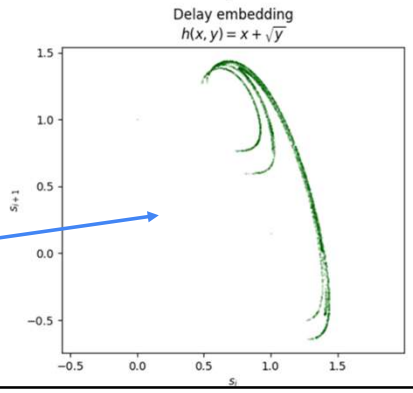
12

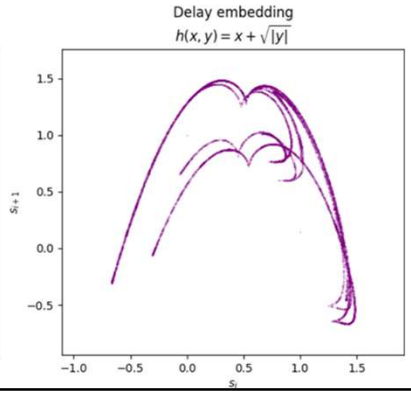




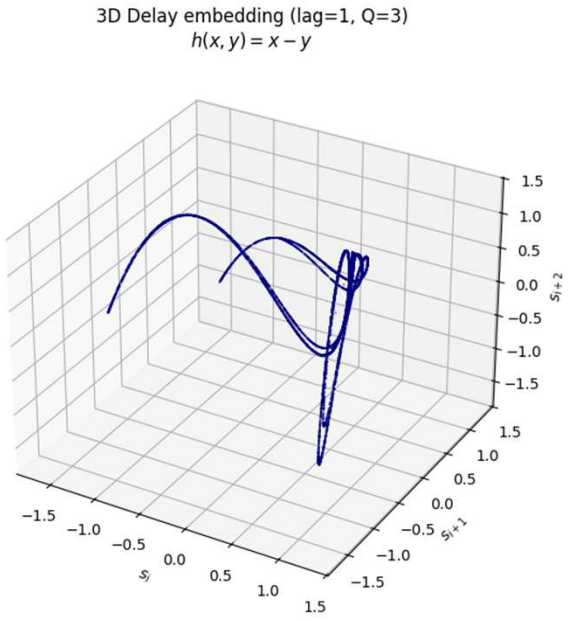
The measurement functions changed the direction of the spread to be more vertical

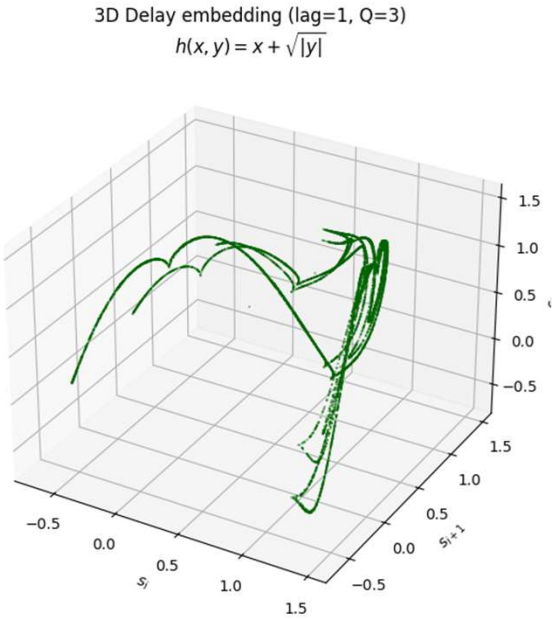
The square root of y introduced NaN values in h(x,y) because the original Henon data contains negative values.





13



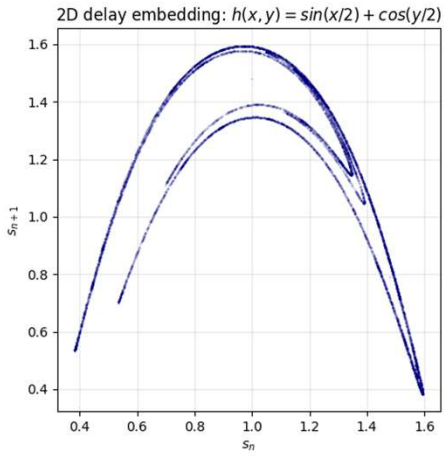


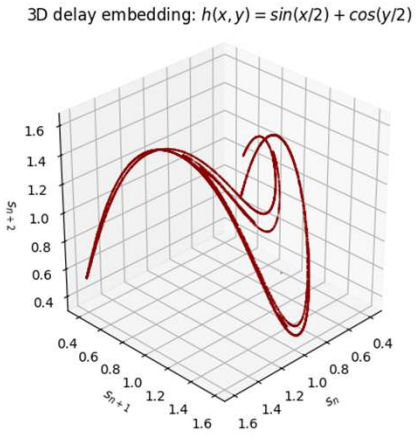
14

2D and 3D Delay Reconstructions of Henon Map
With various measurement functions

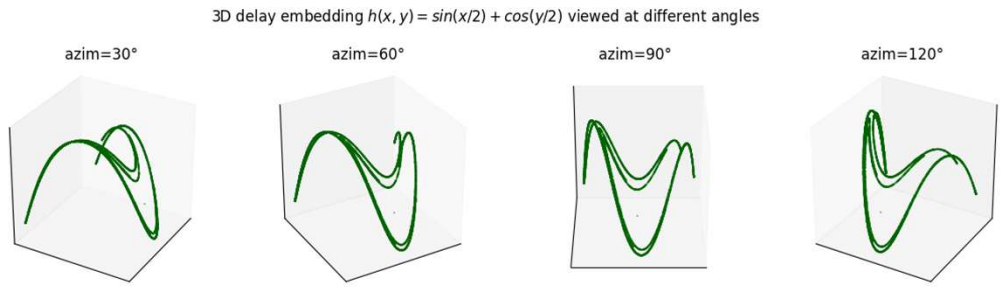
- a = 1.4, b = 0.3
- Delay (tau) = 1

15

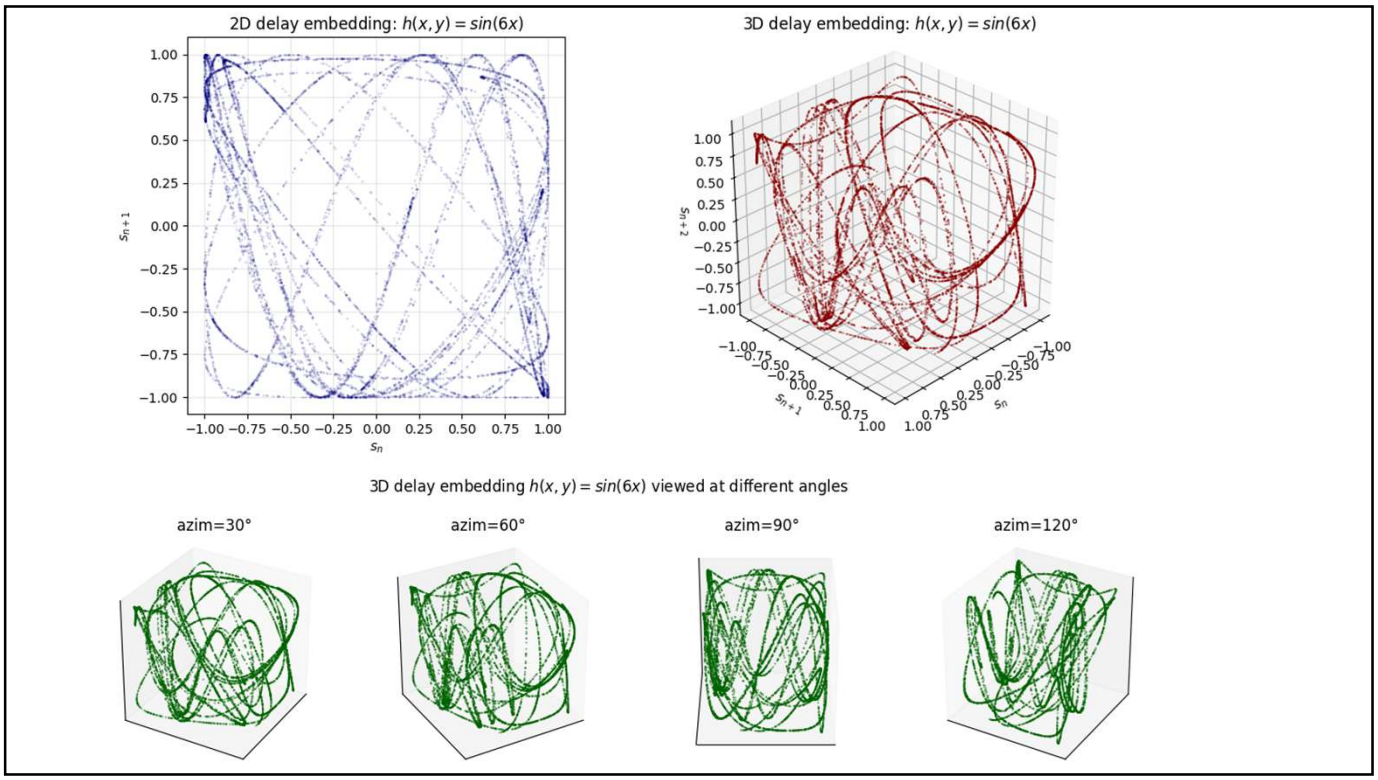




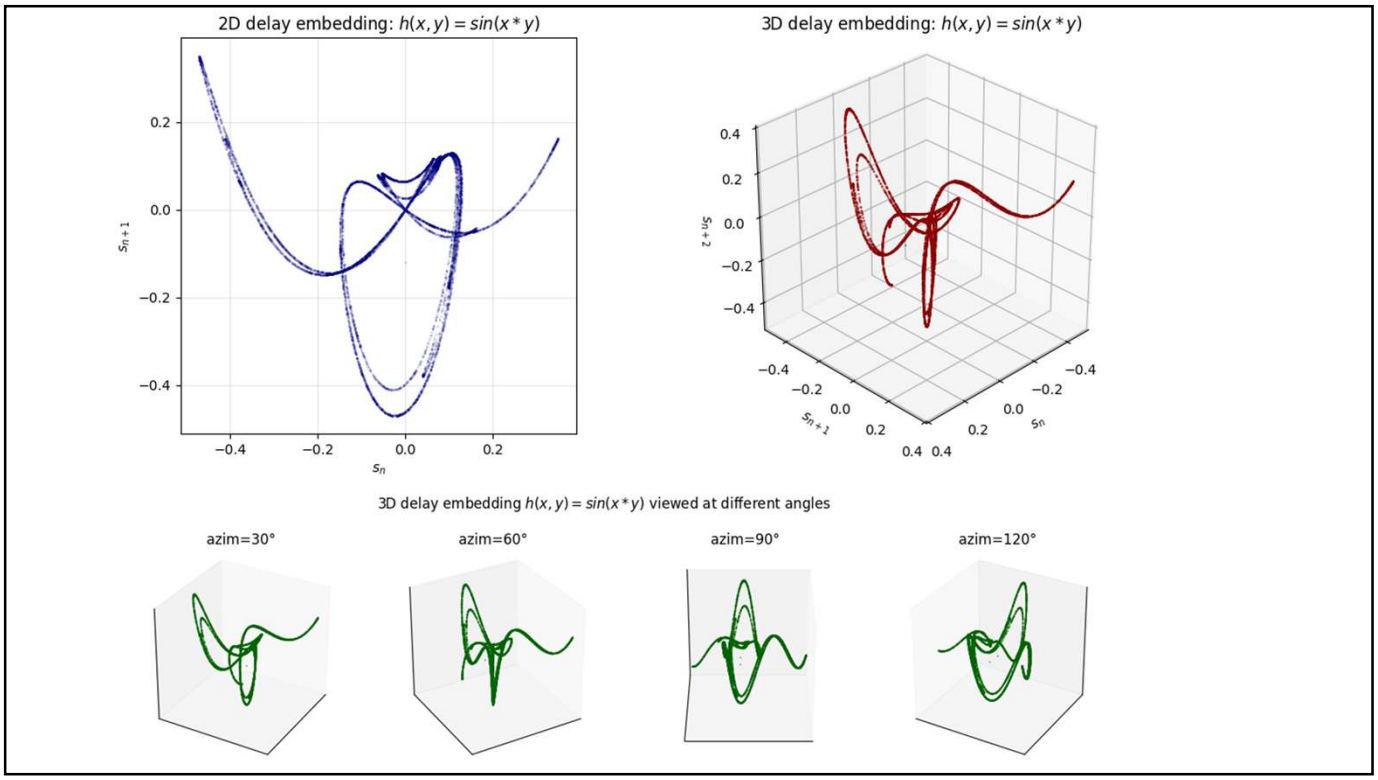
- Similar to the original Henon map
- No intersection?
- Used both x and y



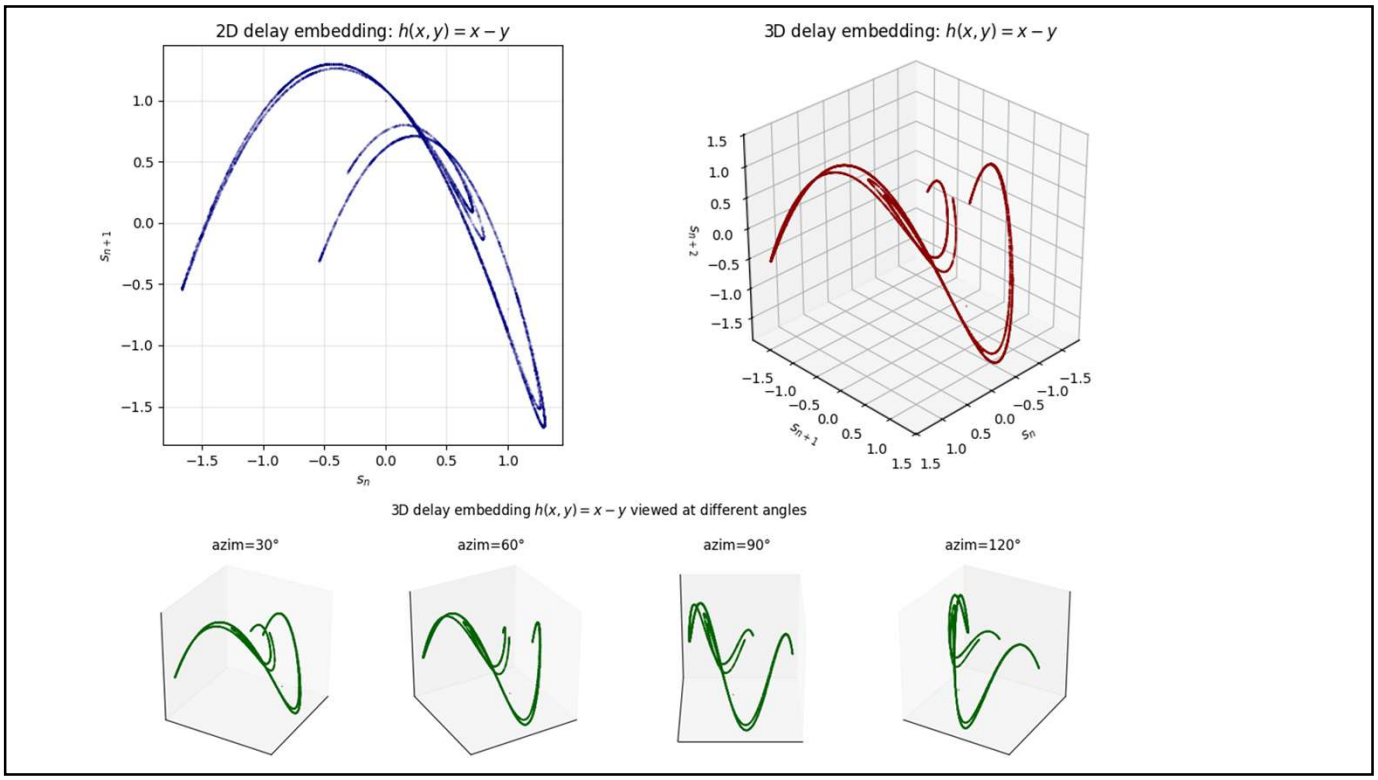
16


2D delay embedding: $h(x,y) = sin(6x)$
3D delay embedding: $h(x,y) = sin(6x)$
s_{n+1}
s_n
s_{n+2}
3D delay embedding $h(x,y) = sin(6x)$ viewed at different angles
azim=30°
azim=60°
azim=90°
azim=120°

17


2D delay embedding: $h(x,y) = sin(x*y)$
3D delay embedding: $h(x,y) = sin(x*y)$
s_{n+1}
s_n
s_{n+2}
3D delay embedding $h(x,y) = sin(x*y)$ viewed at different angles
azim=30°
azim=60°
azim=90°
azim=120°

18

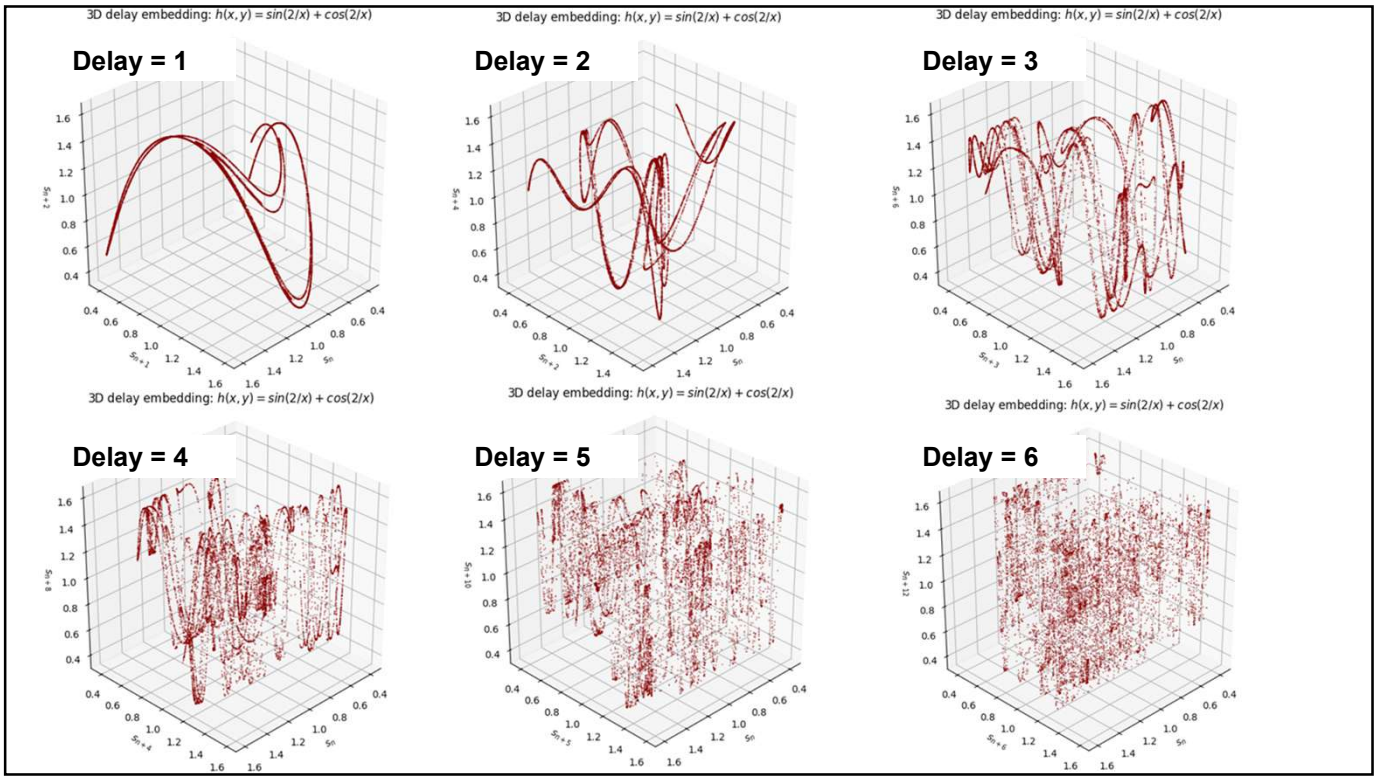


19

3D Delay Reconstructions of Henon Map With various delays (tau)

- a = 1.4, b = 0.3
- Measurement function = sin(2/x) + cos(2/x)

20



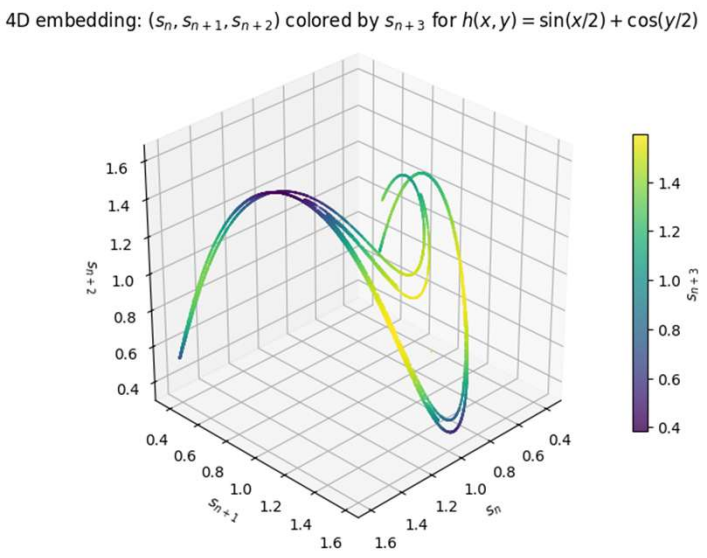
21

4D Delay Reconstructions of Henon Map

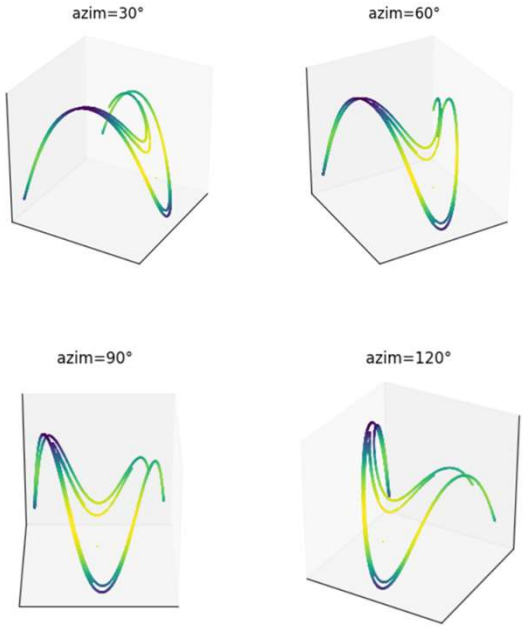
- a = 1.4, b = 0.3
- Delay = 1
- Measurement function = sin(2/x) + cos(2/x)

- Henon Map dimension is about 1.26
- 2 * D +1 = 3.52 (The recommended embedding dimension would be **4**)

22



Good because there is no self-intersection



23

GPR with Henon Map

1. Prepare 1,000 data points from Henon Map (a = 1.4, b=0.3)
   - Initial point $(x_0, y_0) = (0, 0)$
2. Train data: randomly selected 15 data points from 1,000 data
   - (x_n, y_n)
3. Target: next 15 under Henon Map
   - (x_{n+1}, y_{n+1})

24

GPR with Henon Map

4. Targets for two models: next 15 data points under Henon Map
   - First model: $(x_n, y_n) \rightarrow x_{n+1}$
   - Second Model: $(x_n, y_n) \rightarrow y_{n+1}$
   - Gaussian kernel function with a linear basis function (Radial Basis Function (RBF) kernel) was used
   - The paper, SSR-GPR, also used RBF kernel.

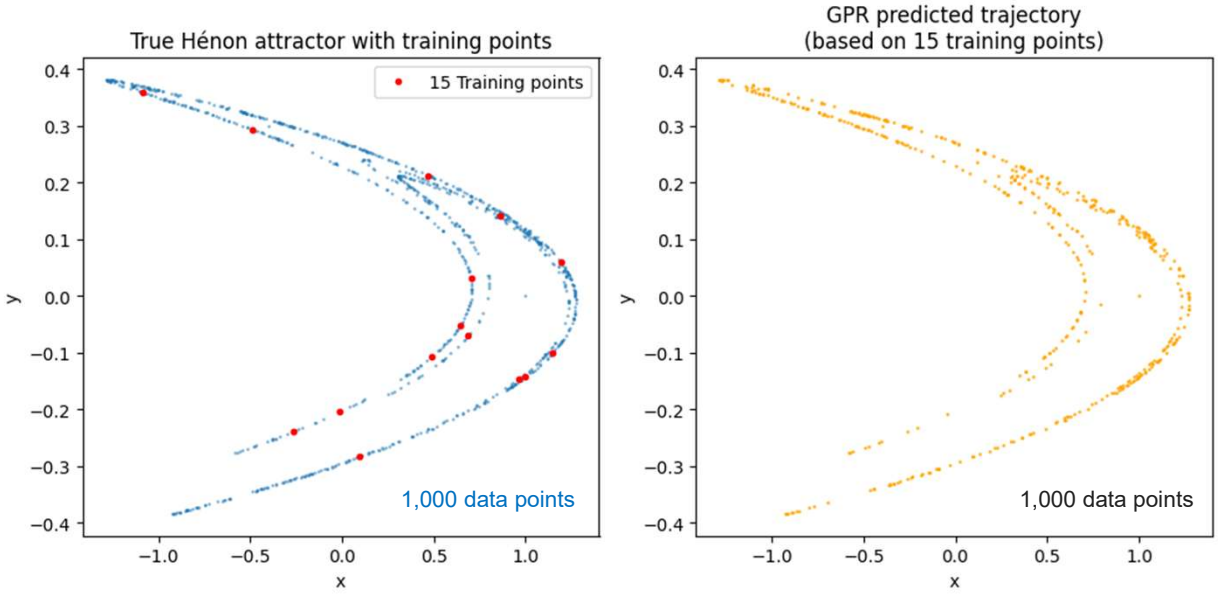
25

GPR with Henon Map

X_Train			Y_Train (Target)		
Randomly selected n_th	x_n	y_n	n+1 th	x_n+1	y_n+1
658	[0.68646828	-0.07079229]	659	0.269474	0.20594
222	[-0.48632194	0.29328666]	223	0.962174	-0.145897
975	[-0.261289	-0.23882194]	976	0.665597	-0.078387
250	[0.70236398	0.03148172]	251	0.34084	0.210709
833	[1.19226984	0.0597499]	834	-0.93036	0.357681
417	[0.47026821	0.2118148]	418	0.902202	0.14108
816	[0.09595117	-0.28328101]	817	0.70383	0.028785
379	[1.14566615	-0.10112261]	380	-0.938694	0.3437
539	[-1.08455371	0.35893266]	540	-0.287827	-0.325366
11	[0.64224426	-0.05313851]	12	0.369393	0.192673
349	[0.864024	0.14023304]	350	0.095081	0.259207
629	[-0.01307676	-0.20437665]	630	0.795384	-0.003923
935	[0.99886186	-0.14325048]	936	-0.540066	0.299659
223	[0.96217402	-0.14589658]	224	-0.441987	0.288652
7995	[0.48804933	-0.10715004]	7996	0.559381	0.146415

26

GPR with Henon Map



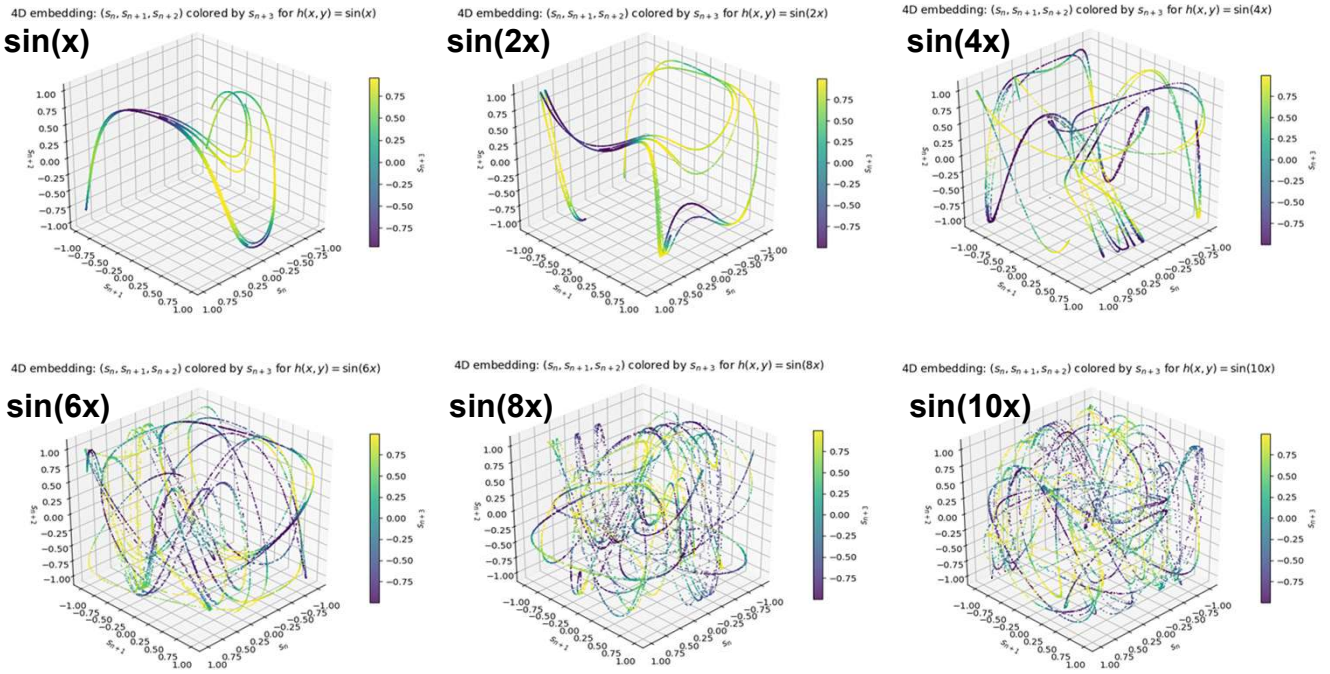
27

4D Delay Reconstructions of Henon Map

- a = 1.4, b = 0.3
- Delay = 1
- Try high frequency measurement functions

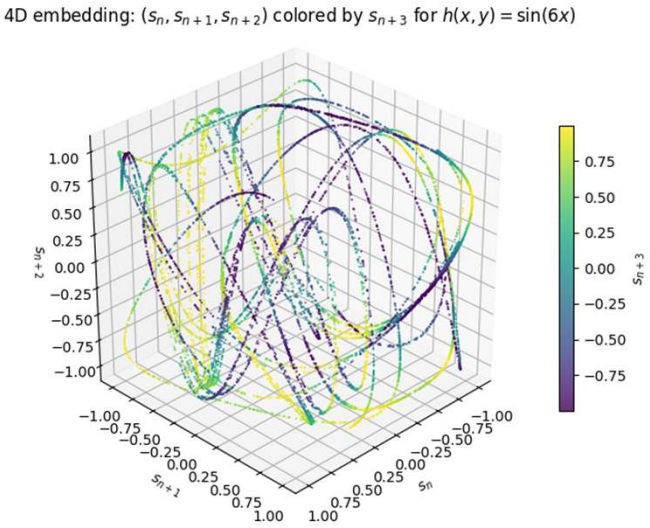
- Henon Map dimension is about 1.26
- 2 * D +1 = 3.52 (The recommended embedding dimension would be **4**)

28



29

4D embedding, h(x) = sin(6x)



1. Is it fractal?

- Box-counting dimension: 1.378
- non-integer and between 1 and 2

2. Self-Intersection?

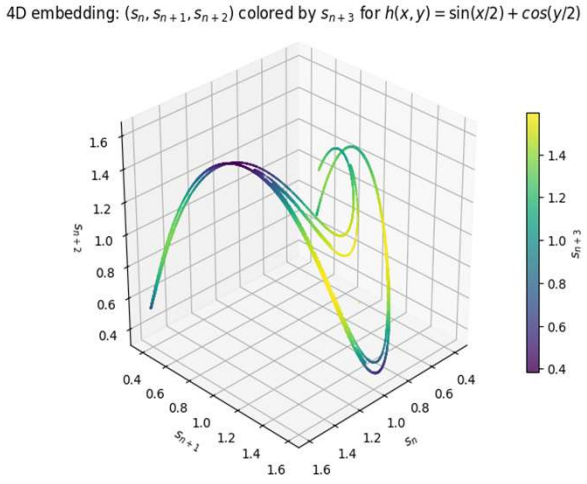
- Performed False Nearest Neighbor analysis (FNN)
- It determines the appropriate embedding dimension, which won't have self-intersections.
- FNN fraction = 0.0001 (1 False out of 19,996 checked)
- This means only 1 out of 19,996 of points are identified as false nearest neighbors at the embedding dimension 4. (unfolded correctly)
- However, true nearest neighbors could still exist → intersections may remain

3. Is sin(6x) a good measurement function?

- Based on the two tests above, it could be
- However, it is high-frequency function → more sensitive to small changes in x.
- May not be appropriate for real world examples (weather system changes).

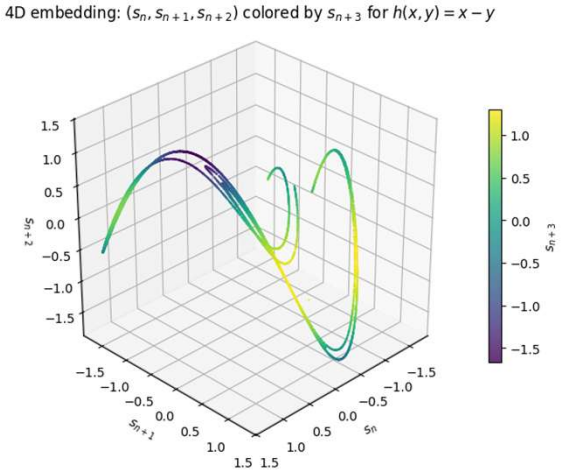
30

h(x) = sin(x/2) + cos(y/2)



- Box-counting dimension = 1.277
- FNN fraction = 0.0000 (almost zero)

h(x) = x - y



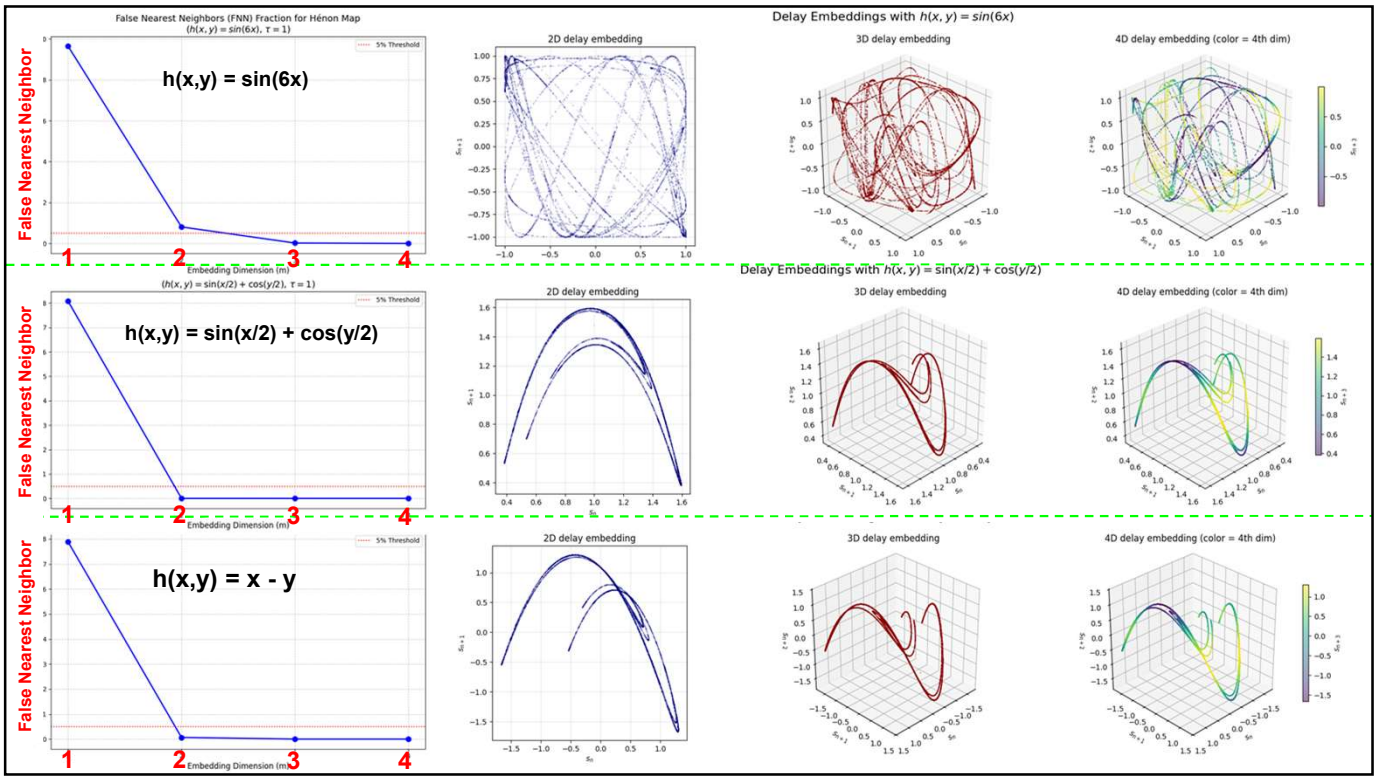
- Box-counting dimension = 1.262
- FNN fraction = 0.0000 (almost zero)

Both could work well as measurement functions

31

FNN Test Across Measurement Functions and Dimensions

32

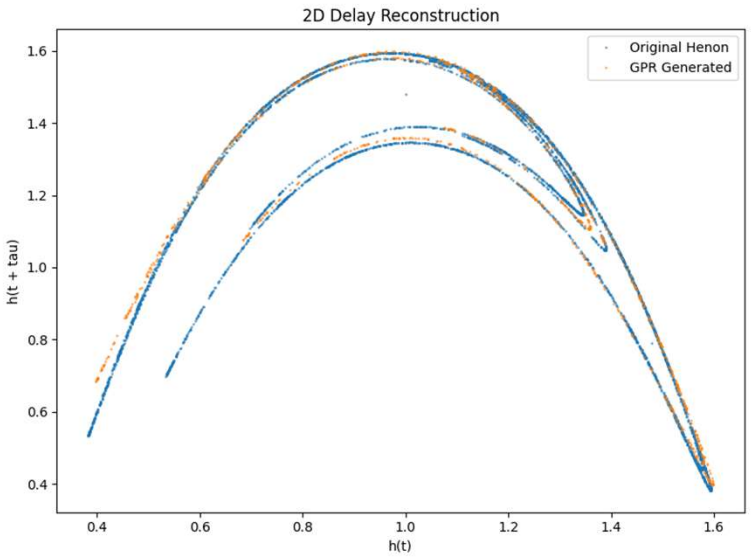


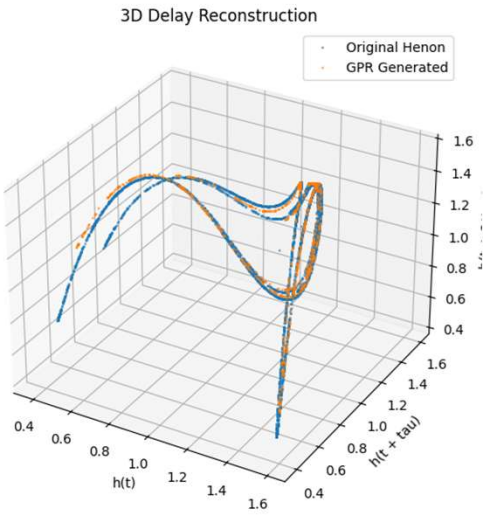
33

2. Comparing Delay Reconstructions of Henon Map vs GPR Approximation

1. Generated time series using the Hénon map (a = 1.4, b = 0.3)
1. Trained Gaussian Process Regression (GPR) using only 10 Henon map points
2. Iteratively predicted new trajectories using GPR as an approximate model
   - alpha = 0.001 to allow smoothing while avoiding overfitting
3. Applied measurement function: h(x, y) = sin(x/2) + cos(y/2)
4. Performed 2D and 3D delay reconstructions (lag = 1) on both original and GPR-generated data
5. Distances were less than 6% of the maximum possible in embedding space => indicating strong structural similarity

34





- Hausdorff Distance for 2D Attractors: 0.1515
- Hausdorff Distance for 3D Attractors: 0.4072

35

Evaluating the Performance of GPR for Delay-Embedded Henon Attractors Using the Wasserstein Distance

Method 1 (the previous method presented last week - **not** realistic for real world situation)

1) GPR on Henon data directly
   - Randomly select samples
   - two GPR model for x and y separately
2) Apply measurement function
3) Delay Embedded Reconstruction (2D and 3D)

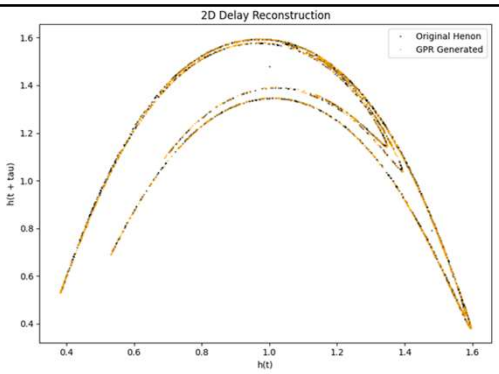
Method 2 (the new method - **realistic** for real world situation)

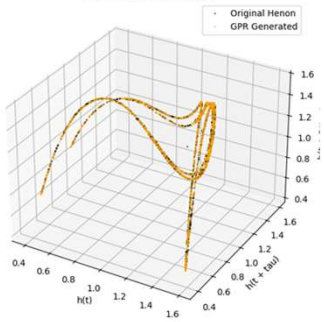
1) Apply measurement function on Original Henon data => Random Sample from the measured data
2) Delay Embedded Reconstruction (2D and 3D)
3) GPR on 2D and 3D data
   - 2D model: 2 dimension data for both X_train (input) and y_train (output)
   - 3D model: 3 dimension data for both X_train (input) and y_train (output)

36

GPR on
20 samples

Method 1

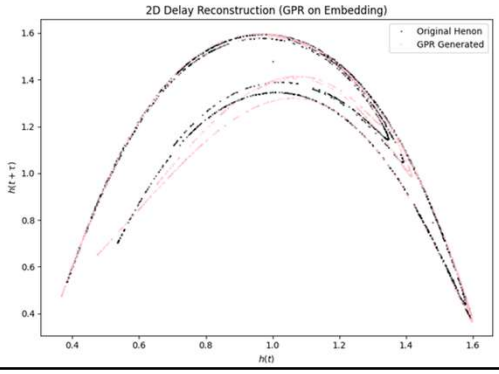


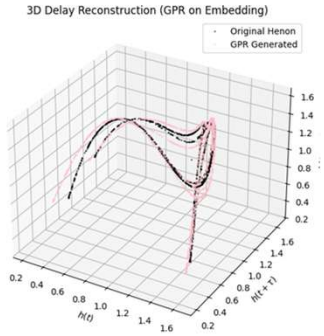


Wasserstein Distance

- 2D: 0.0143
- 3D: 0.0215

Method 2





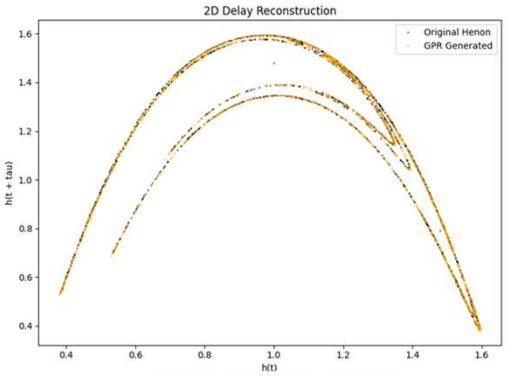
Wasserstein Distance

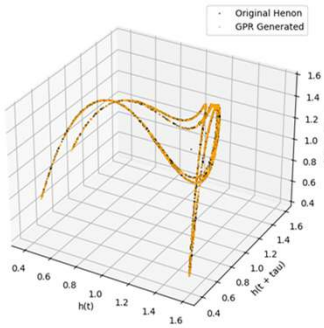
- 2D: 0.0343
- 3D: 0.0338

37

GPR on
30 samples

Method 1

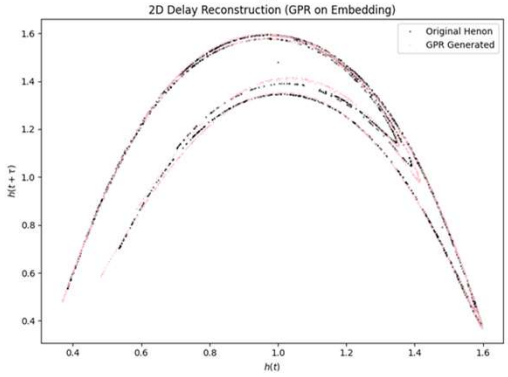


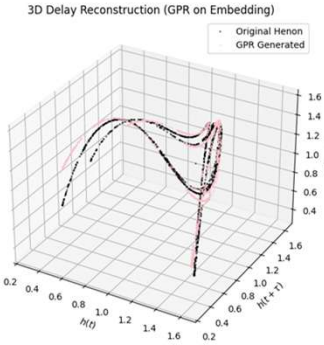


Wasserstein Distance

- 2D: 0.0096
- 3D: 0.0141

Method 2





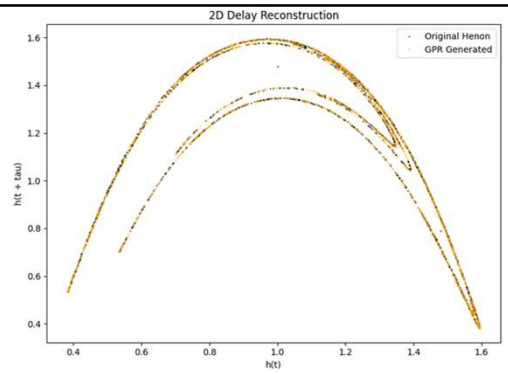
Wasserstein Distance

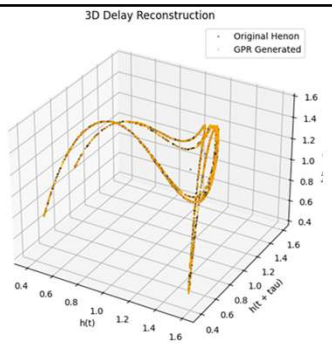
- 2D: 0.0273
- 3D: 0.0446

38

GPR on
40 samples

Method 1

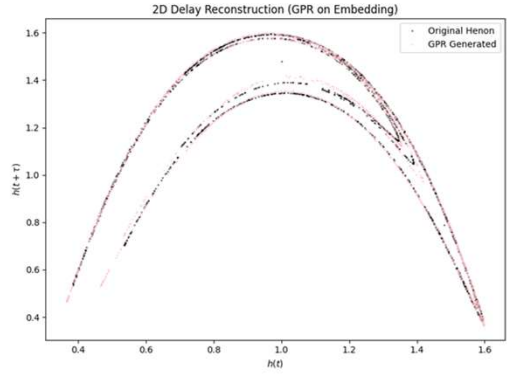


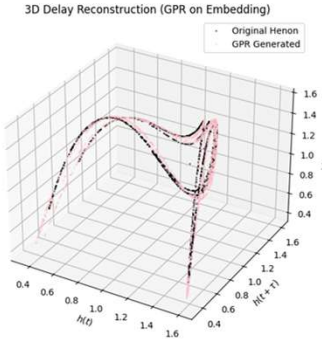


Wasserstein Distance

- 2D: 0.0112
- 3D: 0.0168

Method 2





Wasserstein Distance

- 2D: 0.0119
- 3D: 0.0236

39

Wasserstein Distance

I learned that there are different approaches for calculating the Wasserstein Distance when it is higher dimensions like 2D or 3D.

What I did:

- Used the same python package (scipy.stats import wasserstein_distance) that Alexander used for Logistic Map (not higher dimension)
- Computed the average of 1-D Wasserstein distances of each coordinate for 2D and 3D.
- But I learned that in multiple dimensions, Wasserstein depends on the joint geometry (e.g., correlations, orientation, shape), not just the per-coordinate marginals.

Alternatives:

- Use POT library to compute EMD (earth mover's distance) on the pairwise Euclidean cost matrix => **I can try this next week.**

40

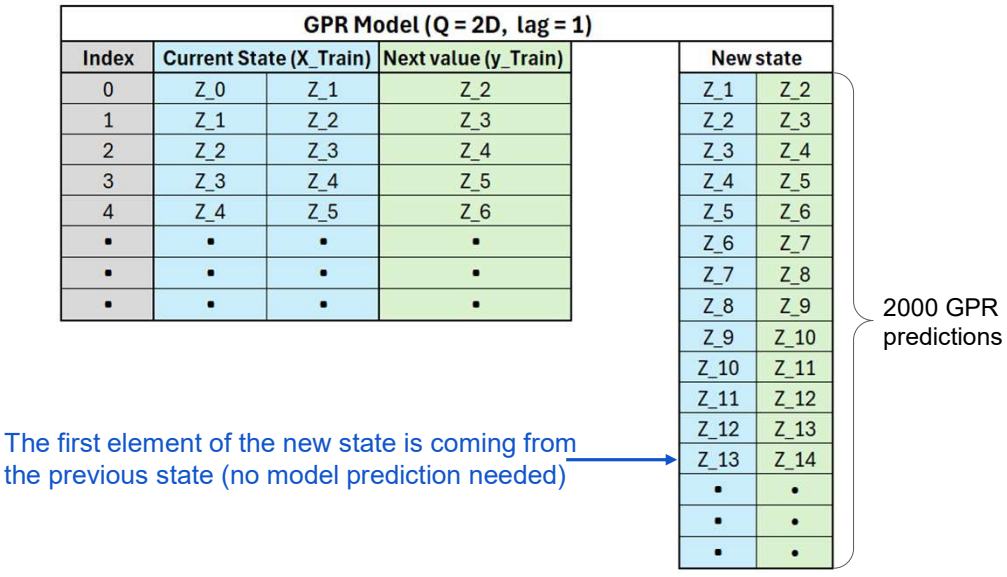
Process for Delay Reconstruction and GPR

- **System Parameters:** The Henon map was generated with standard parameters **a=1.4** and **b=0.3**.
- **Measurement:** A single 1D time series was created using the measurement function **h(x, y) = sin(x/2) + cos(y/2)**
- **Delay Embedding:** The 1D measured time series was used to create a multi-dimensional state space through delay embedding with a delay parameter of **1**. This process created two reconstructed attractors: a 2D attractor and a 3D attractor.
- **Training Data Sampling:** From the reconstructed 2D and 3D attractors, **20, 30, 40 pairs of training samples were randomly selected** to train the GPR model.
- **Iterative Process:** To ensure the results are not dependent on a single random sample selection, the entire process—from random sampling to GPR model training and new attractor generation—was run **100 times for each pair.**
- **Wasserstein Distance** was calculated based on the **2,000** GPR generated points vs **2,000** points from the original map.

41

GPR Model Structure for 2D at lag 1

I iterated this process for 100 times with 20, 30, and 40 random samples and calculated average Wasserstein distance for obtaining reliable results.



42

GPR Model Structure for 3D at lag 1

I iterated this process for 100 times with 20, 30, and 40 random samples and calculated average Wasserstein distance for obtaining reliable results.

GPR Model (Q = 3, lag = 1)								
Index	Current State (X_Train)			Next value (y_Train)	New state			
0	Z_0	Z_1	Z_2	Z_3	Z_1	Z_2	Z_3	
1	Z_1	Z_2	Z_3	Z_4	Z_2	Z_3	Z_4	
2	Z_2	Z_3	Z_4	Z_5	Z_3	Z_4	Z_5	
3	Z_3	Z_4	Z_5	Z_6	Z_4	Z_5	Z_6	
4	Z_4	Z_5	Z_6	Z_7	Z_5	Z_6	Z_7	
•	•	•	•	•	Z_6	Z_7	Z_8	
•	•	•	•	•	Z_7	Z_8	Z_9	
•	•	•	•	•	Z_8	Z_9	Z_10	
					Z_9	Z_10	Z_11	
					Z_10	Z_11	Z_12	
					Z_11	Z_12	Z_13	
					Z_12	Z_13	Z_14	
					Z_13	Z_14	Z_15	
					•	•	•	
					•	•	•	
					•	•	•	

2,000 GPR predictions

The first and second elements of the new state are coming from the previous state (no model prediction needed)

43

Model development part of code

- alpha is for preventing the overfitting (hyperparameter for noise)
- when alpha = 0, the predictions will exactly pass through the training points.

```
# Train the GPR model to predict the single next value.
gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10, alpha=0.0001, normalize_y=False)
gp.fit(X_train, y_train_scalar)

# Use the trained model to generate a new time series by iteration.
predicted_attractor = np.zeros((n_predict_points, start_point.shape[0]))
predicted_attractor[0, :] = start_point

# Create the initial state vector for prediction
current_state_vector = start_point.copy()

# Generate points iteratively
for i in range(n_predict_points - 1):
    # Predict the NEXT scalar value of the time series from the current state vector.
    next_scalar_value = gp.predict(current_state_vector.reshape(1, -1))[0]

    # Construct the new state vector by shifting the old one and appending the new scalar.
    new_state_vector = np.roll(current_state_vector, -1)
    new_state_vector[-1] = next_scalar_value

    predicted_attractor[i+1, :] = new_state_vector
    current_state_vector = new_state_vector
```

Only predicts Z(t + 2) in the case of 2D

Z(t + 1) is shifted to the vector for the new state

[Z(t + 1), Z(t + 2)] is the new state vector

44

Results

1. Sample size = 20

- Average Wasserstein Distance for 2D Attractors: 0.2593 +/- 0.6443
- Average Wasserstein Distance for 3D Attractors: 0.2086 +/- 0.4796
- Average Normalized Wasserstein Distance for 2D Attractors: 0.0324 +/- 0.0805
- Average Normalized Wasserstein Distance for 3D Attractors: 0.0174 +/- 0.0400

2. Sample size = 30

- Average Wasserstein Distance for 2D Attractors: 0.1647 +/- 0.6245
- Average Wasserstein Distance for 3D Attractors: 0.3553 +/- 0.8473
- Average Normalized Wasserstein Distance for 2D Attractors: 0.0206 +/- 0.0781
- Average Normalized Wasserstein Distance for 3D Attractors: 0.0296 +/- 0.0706

3. Sample size = 40

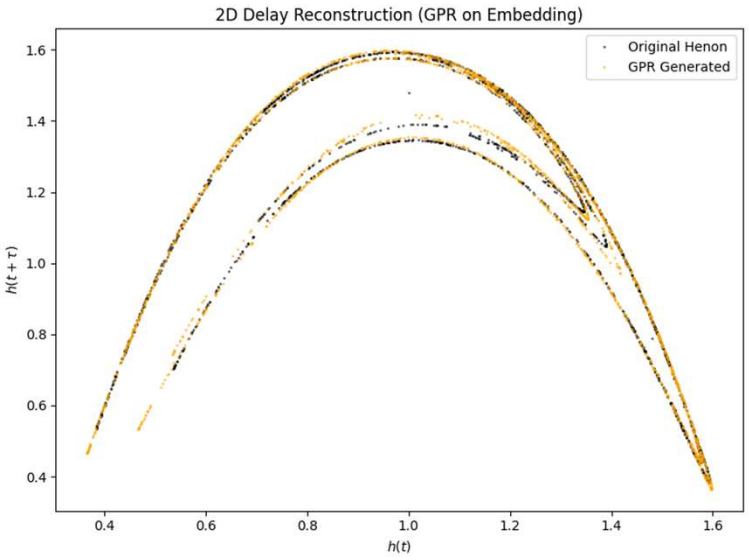
- Average Wasserstein Distance for 2D Attractors: 0.0357 +/- 0.1404
- Average Wasserstein Distance for 3D Attractors: 0.1536 +/- 0.4345
- Average Normalized Wasserstein Distance for 2D Attractors: 0.0045 +/- 0.0175
- Average Normalized Wasserstein Distance for 3D Attractors: 0.0128 +/- 0.0362

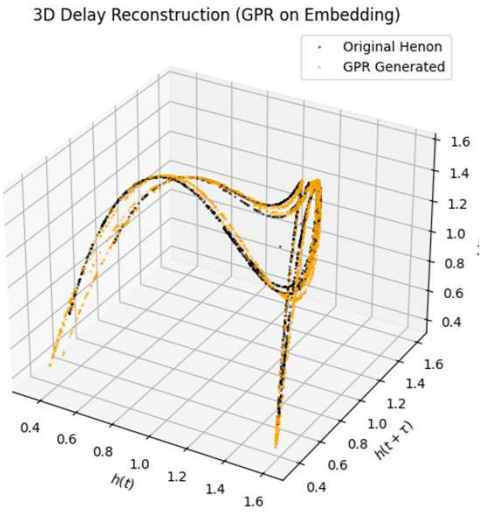
Increasing the number of random training samples generally improved the model's accuracy, with the smallest average normalized Wasserstein distance observed when using 40 samples. The slight increase in distance at 30 samples for the 3D attractor can probably be attributed to the variability of random sampling, where some subsets may be less representative than others.

45

Visual Results: Original vs GPR-Generated Attractors

Results when the sample size is 40 with the random seed set as ***np.random.seed(42)***





46

Process for Delay Reconstruction and GPR based on Noisy Data

- **System Parameters:** The Henon map was generated with standard parameters **a=1.4** and **b=0.3**.
- **Measurement:** A single 1D time series was created using the measurement function **h(x, y) = sin(x/2) + cos(y/2)**
- Noise with a **standard deviation of 0.05** was added to the measured data to simulate real-world measurement inaccuracies
- **Delay Embedding:** The 1D measured time series with noise was used to create a multi-dimensional state space through delay embedding with a delay parameter of **1**. This process created two reconstructed attractors: a 2D attractor and a 3D attractor.
- **Training Data Sampling:** From the reconstructed 2D and 3D attractors, **40, 100, 200 pairs of training samples were randomly selected** to train the GPR model.
- **Iterative Process:** To ensure the results are not dependent on a single random sample selection, the entire process—from random sampling to GPR model training and new attractor generation—was run **100 times for each pair.**
- **Wasserstein Distance (POT library)** was calculated based on the **2,000** GPR generated points vs **2,000** points from the original map

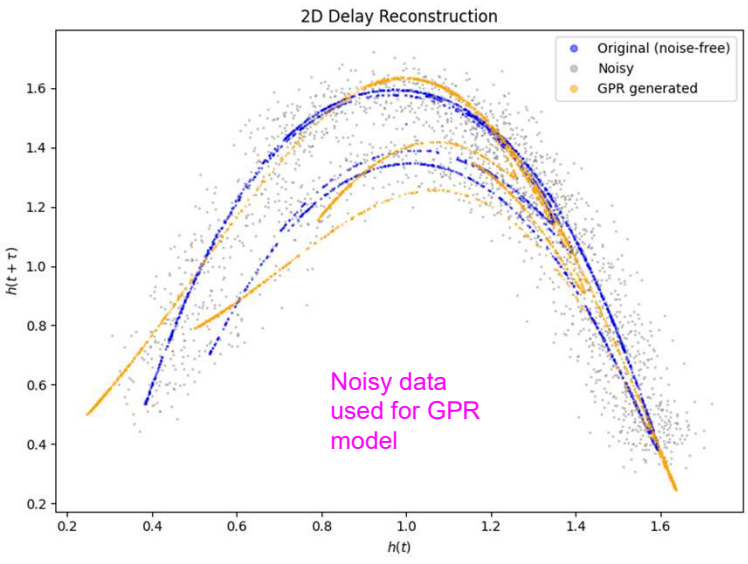
47

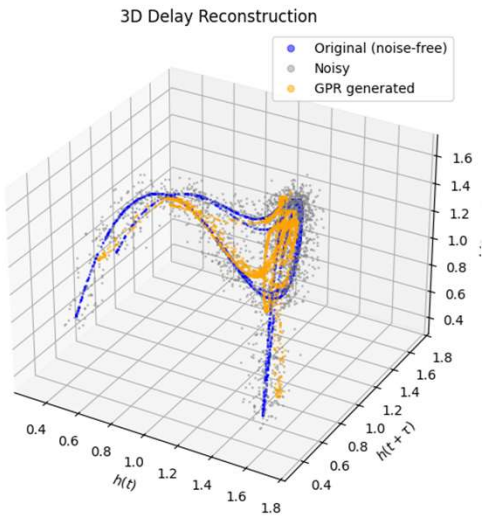
Example result from one random training set

- Sample Size: **40**
- Wasserstein for 2D: **0.3633** (Normalized: **0.0642**)
- Wasserstein for 3D: **0.4027** (Normalized: **0.0581**)
- np.random.seed(**50**)

40 sample result:

- It can draw the big boomerang shape, but it misses lots of details.
- Some areas have no orange points because the model didn't learn enough there.
- Overall, the copy of the attractor is rough and incomplete.





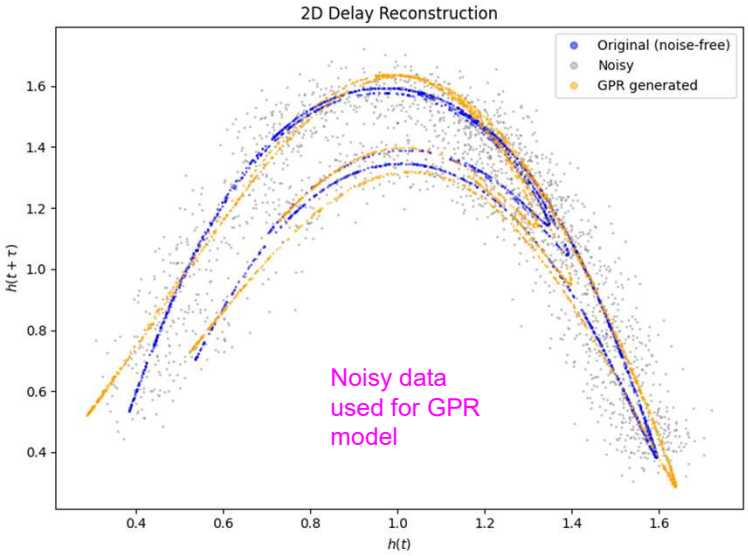
48

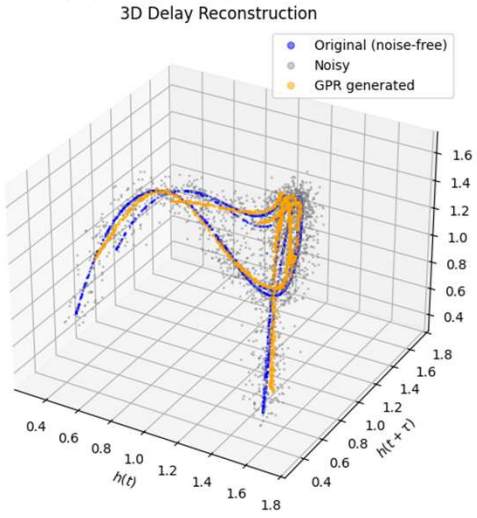
Example result from one random training set

- Sample Size: **100**
- Wasserstein for 2D: **0.3020** (Normalized: **0.0534**)
- Wasserstein for 3D: **0.3035** (Normalized: **0.0438**)
- np.random.seed(**50**)

100 sample result:

- The orange GPR attractor now follows the blue curve more closely.
- Fewer gaps compared to the 40 sample result: the model fills in more of the boomerang.
- Still not perfect: some small mismatches remain, but overall much smoother and more accurate than before. (especially for 3D)





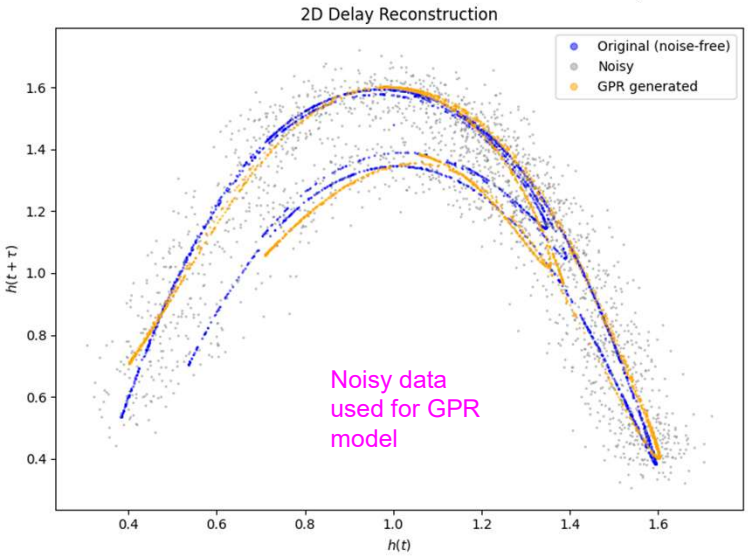
49

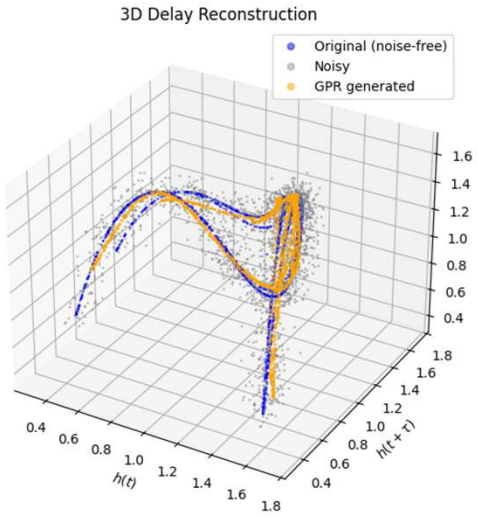
Example result from one random training set

- Sample Size: **200**
- Wasserstein for 2D: **0.2642** (Normalized: **0.0467**)
- Wasserstein for 3D: **0.2807** (Normalized: **0.0405**)
- np.random.seed(**50**)

200 sample result:

- The orange GPR attractor matches the blue curve very closely, especially in 3D.
- The shape of the attractor is well reconstructed.
- Distances are the lowest so far, showing the best accuracy among the three experiments.





50

Comparison of 40, 100, and 200 Sample Models (100 Trials Each)

1. Sample size = 40

- Average Wasserstein Distance for 2D Attractors: 0.4171 ± 0.1768
- Average Wasserstein Distance for 3D Attractors: 0.4698 ± 0.1483
- Average Normalized Wasserstein Distance for 2D Attractors: 0.0737 ± 0.0313
- Average Normalized Wasserstein Distance for 3D Attractors: 0.0678 ± 0.0214

2. Sample size = 100

- Average Wasserstein Distance for 2D Attractors: 0.3729 ± 0.1435
- Average Wasserstein Distance for 3D Attractors: 0.3832 ± 0.1446
- Average Normalized Wasserstein Distance for 2D Attractors: 0.0659 ± 0.0254
- Average Normalized Wasserstein Distance for 3D Attractors: 0.0553 ± 0.0209

3. Sample size = 200

- Average Wasserstein Distance for 2D Attractors: 0.3330 ± 0.1258
- Average Wasserstein Distance for 3D Attractors: 0.3616 ± 0.1345
- Average Normalized Wasserstein Distance for 2D Attractors: 0.0589 ± 0.0222
- Average Normalized Wasserstein Distance for 3D Attractors: 0.0522 ± 0.0194

51

Salvinia Biomass Dynamics with Takens' Delay Embedding and GPR

1. Visualize Data (Log of Total Dry Wt and Logit Salvinia Damage)

- Plot time-series data for 4 different sites.

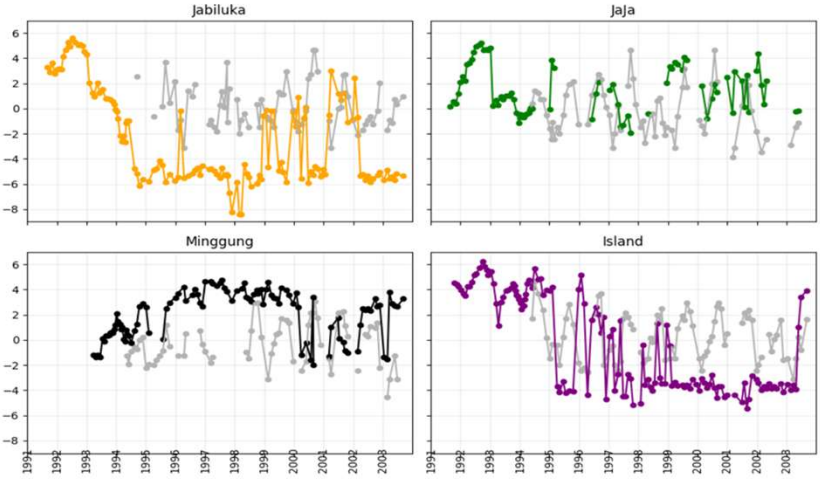
2. I then used the Jabiluka and Island data (which have no missing values) and applied delay embedding and GPR, despite the limited number of data points.

- Takens delay embedding + GPR

52

Plot time series data for 4 sites (Jabiluka, Jaja, Minggung, and Island)

- Colorful dots/line = log of Total dry wt. of Salvinia
- Grey dots/line = logit Salvinia damage (when available).
- Broken lines across large gaps (no imputation)
- Jabiluka and Island have relatively complete, high-quality data, making them well suited for testing Delay Embedding and GPR

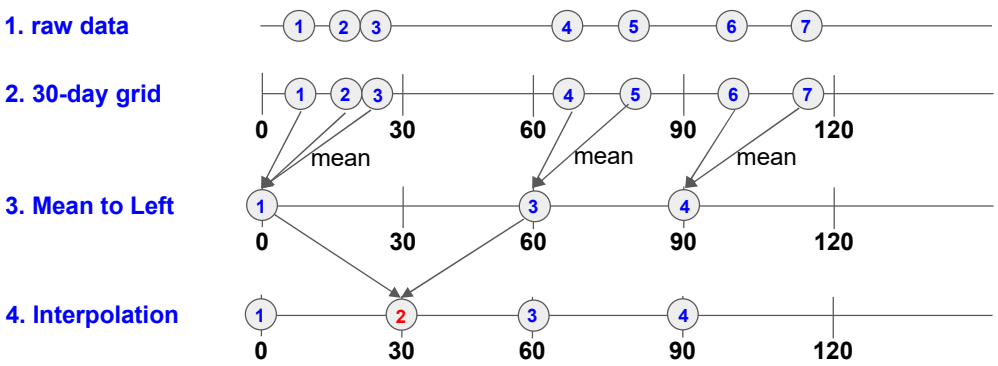


53

Takens delay embedding + GPR (Only Used log biomass of salvinia)

- Chose a uniform 30-day grid (resample + linear time interpolation) for even spacing.
- Ran state-space embeddings and one-step GPR forecasts on sites with no NAs (Jabiluka and Island)
- Tried parameter combos for each site: (Q=2, tau=1), (Q=3, tau=1), (Q=2, tau=2), (Q=3, tau=2).
- Produced time-series fits and state-space plots (2D for Q=2; 3D for Q=3).

Resampling + Interpolation Logic

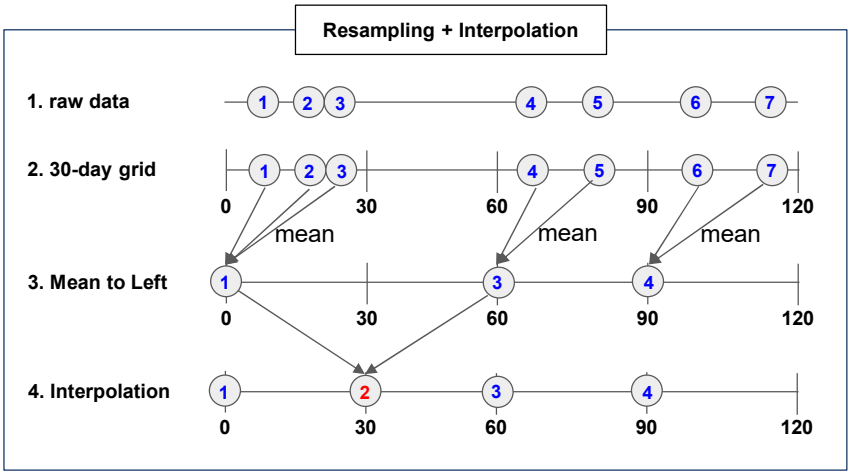


5. Delay Embedding with the uniformly recreated data and GPR (70% Train / 30% Test)

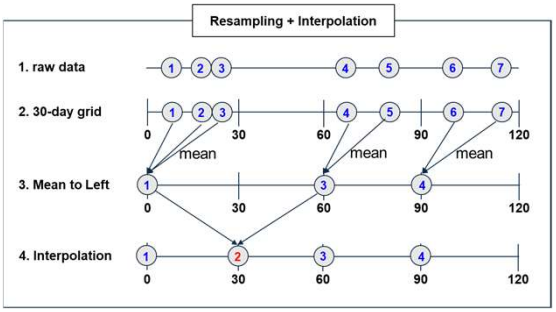
54

Takens delay embedding + GPR (Only Used log biomass of salvinia)

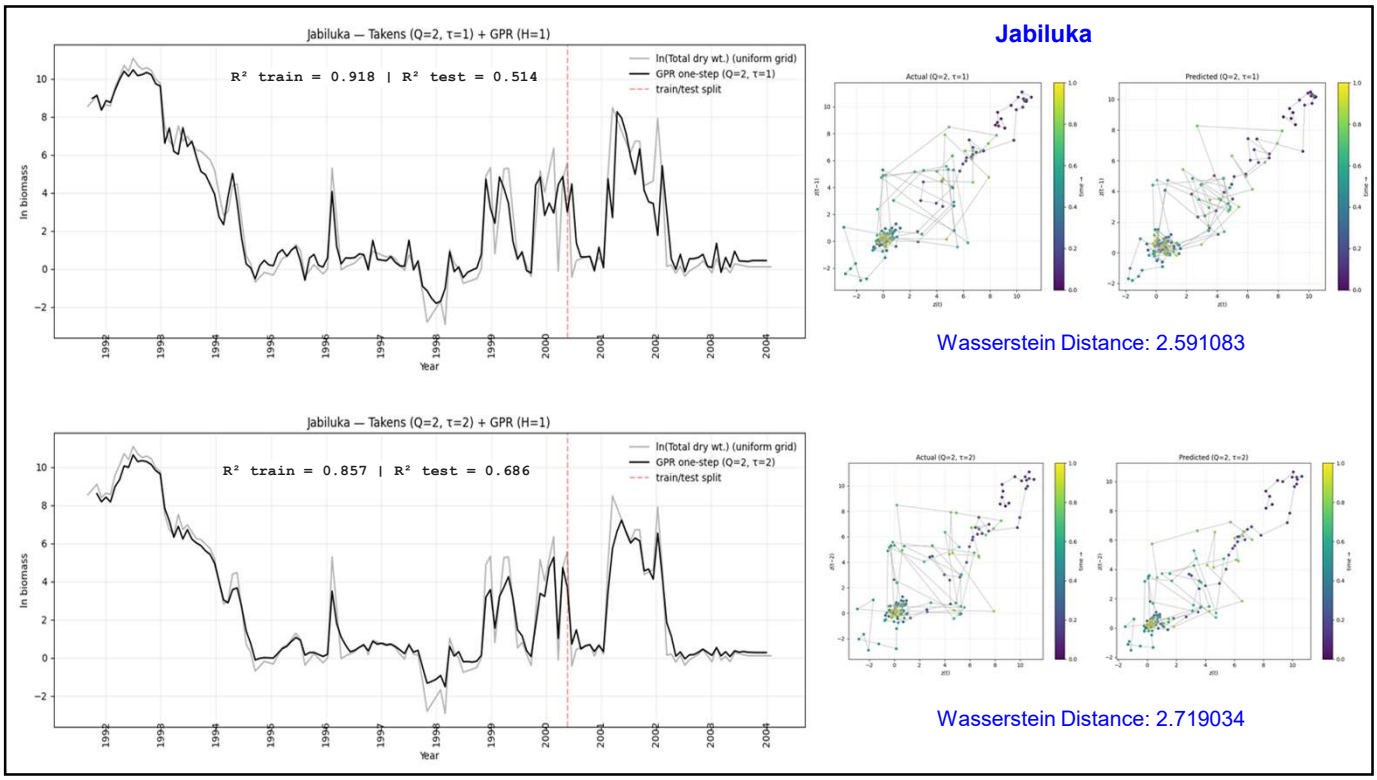
- Chose a uniform 30-day grid (resample + linear time interpolation) for even spacing.
- Ran state-space embeddings and one-step GPR forecasts on sites with no NAs (Jabiluka and Island)
- Tried parameter combos for each site: (Q=2, tau=1), (Q=3, tau=1), (Q=2, tau=2), (Q=3, tau=2).
- Produced time-series fits and state-space plots (2D for Q=2; 3D for Q=3).



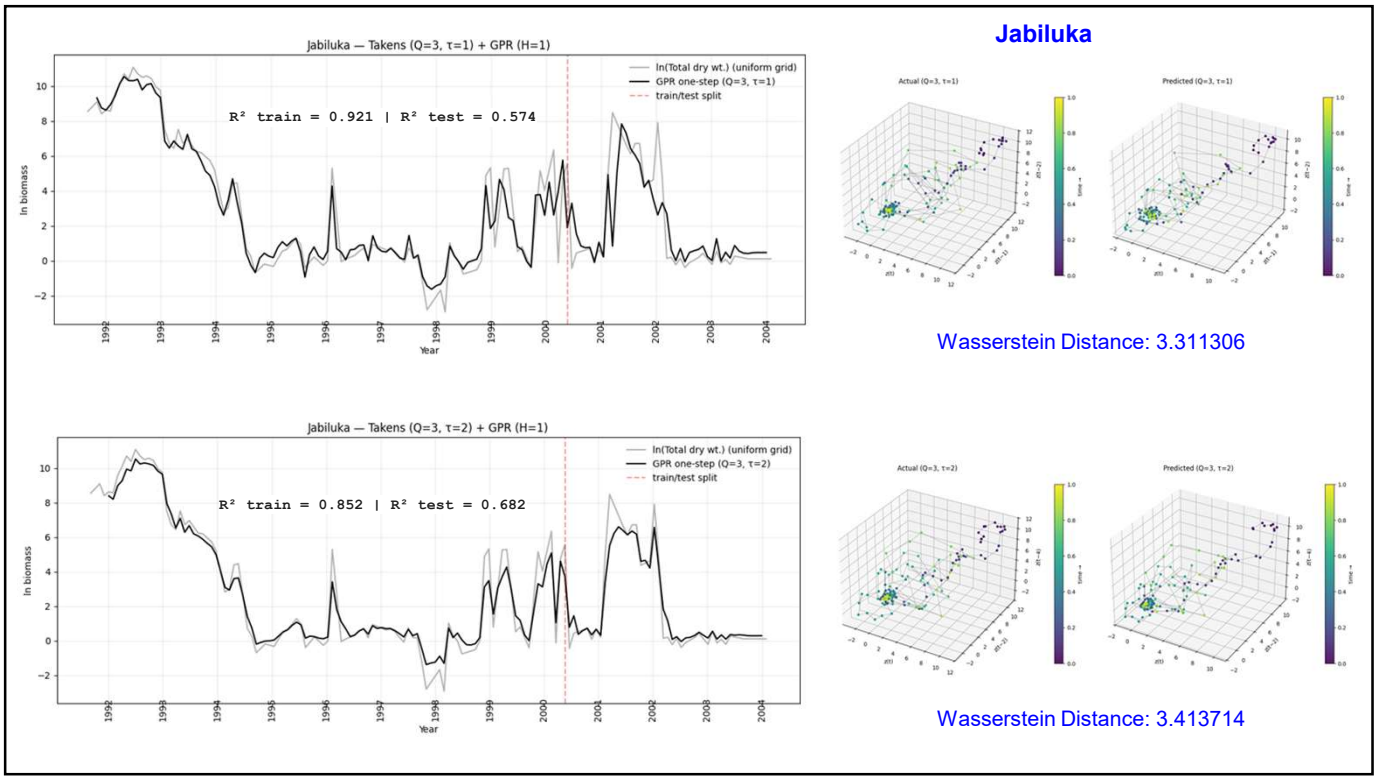
55



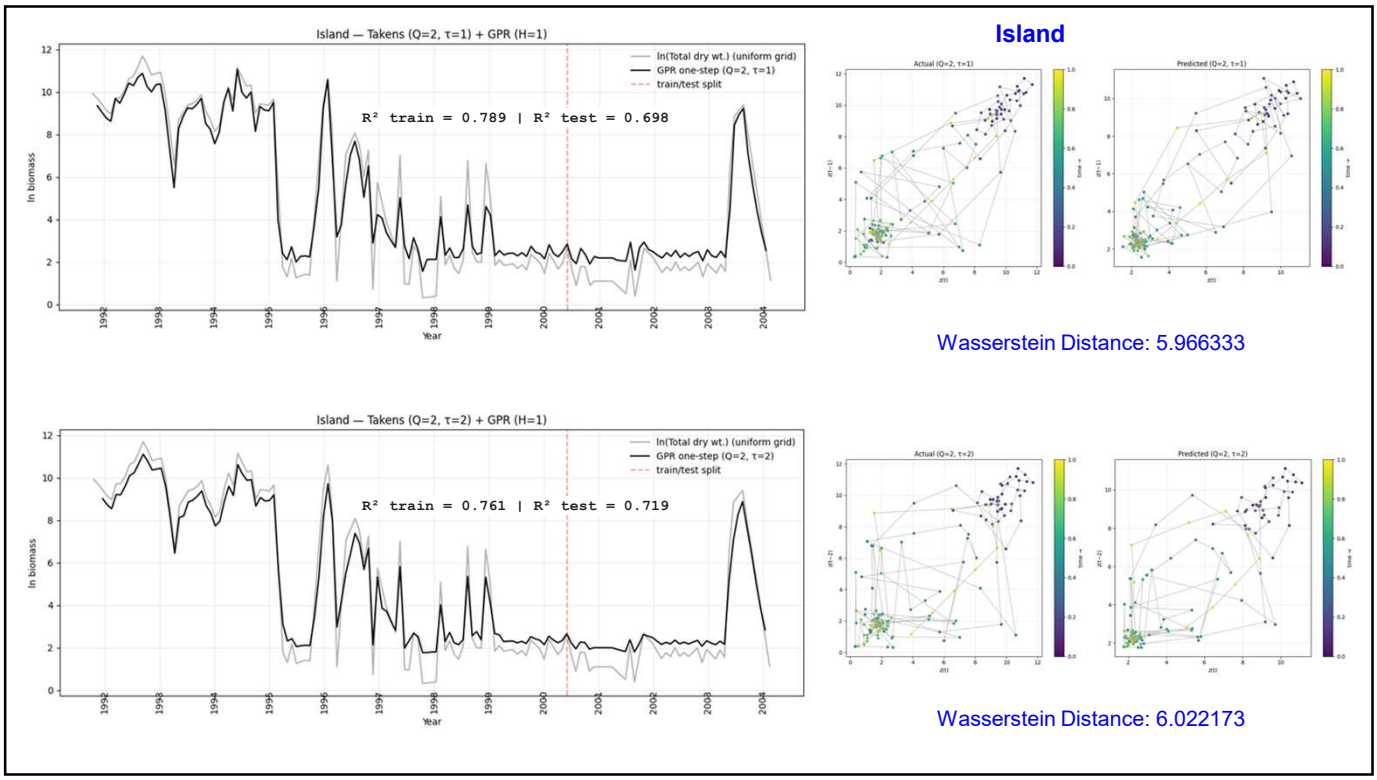
56


Jabiluka — Takens (Q=2, τ=1) + GPR (H=1)
R² train = 0.918 | R² test = 0.514
ln(Total dry wt.) (uniform grid)
GPR one-step (Q=2, τ=1)
train/test split
ln biomass
Year
Jabiluka
Actual (Q=2, τ=1)
Predicted (Q=2, τ=1)
Wasserstein Distance: 2.591083
Jabiluka — Takens (Q=2, τ=2) + GPR (H=1)
R² train = 0.857 | R² test = 0.686
ln(Total dry wt.) (uniform grid)
GPR one-step (Q=2, τ=2)
train/test split
ln biomass
Year
Actual (Q=2, τ=2)
Predicted (Q=2, τ=2)
Wasserstein Distance: 2.719034

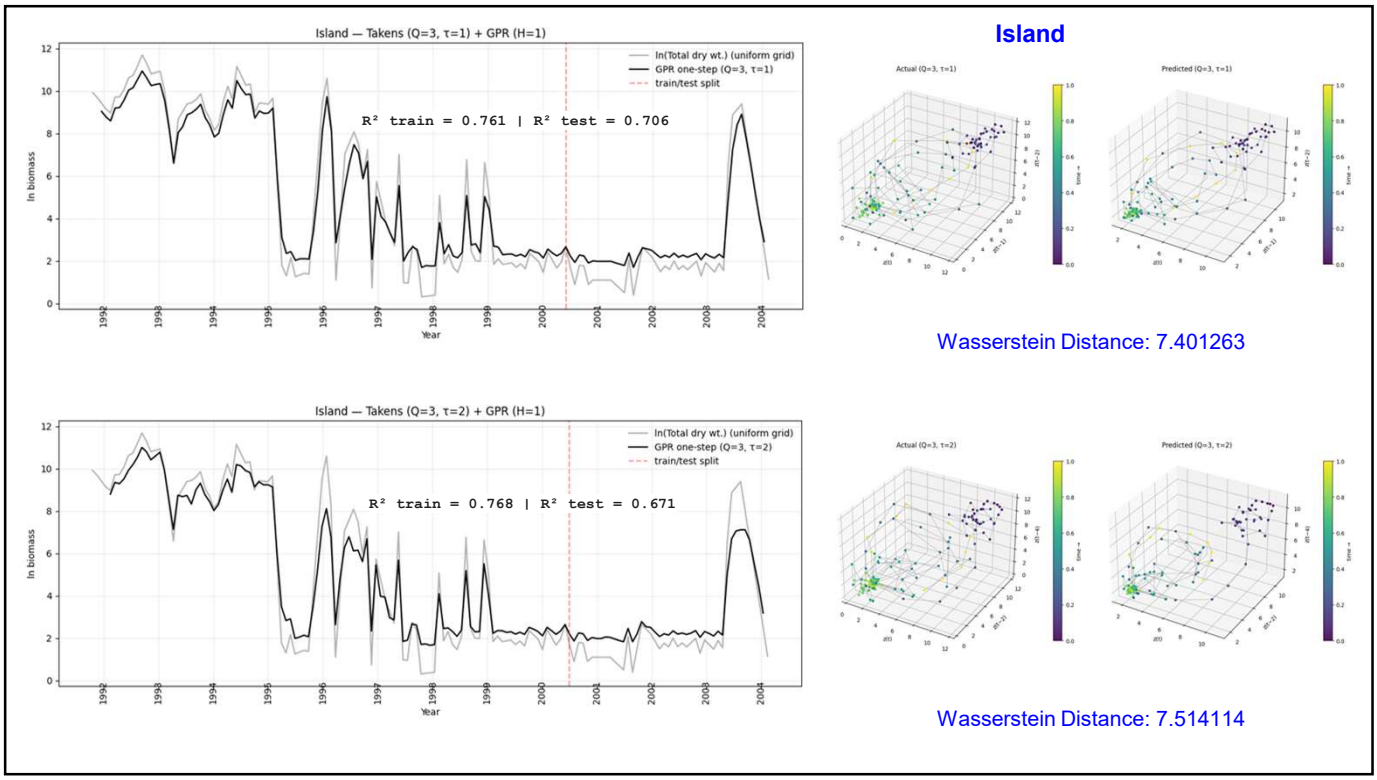
57


Jabiluka — Takens (Q=3, τ=1) + GPR (H=1)
R² train = 0.921 | R² test = 0.574
ln(Total dry wt.) (uniform grid)
GPR one-step (Q=3, τ=1)
train/test split
ln biomass
Year
Jabiluka
Actual (Q=3, τ=1)
Predicted (Q=3, τ=1)
Wasserstein Distance: 3.311306
Jabiluka — Takens (Q=3, τ=2) + GPR (H=1)
R² train = 0.852 | R² test = 0.682
ln(Total dry wt.) (uniform grid)
GPR one-step (Q=3, τ=2)
train/test split
ln biomass
Year
Actual (Q=3, τ=2)
Predicted (Q=3, τ=2)
Wasserstein Distance: 3.413714

58


Island — Takens (Q=2, τ=1) + GPR (H=1)
ln(Total dry wt.) (uniform grid)
GPR one-step (Q=2, τ=1)
train/test split
R² train = 0.789 | R² test = 0.698
ln biomass
Year
Island
Actual (Q=2, τ=1)
Predicted (Q=2, τ=1)
Wasserstein Distance: 5.966333
Island — Takens (Q=2, τ=2) + GPR (H=1)
ln(Total dry wt.) (uniform grid)
GPR one-step (Q=2, τ=2)
train/test split
R² train = 0.761 | R² test = 0.719
ln biomass
Year
Actual (Q=2, τ=2)
Predicted (Q=2, τ=2)
Wasserstein Distance: 6.022173

59


Island — Takens (Q=3, τ=1) + GPR (H=1)
ln(Total dry wt.) (uniform grid)
GPR one-step (Q=3, τ=1)
train/test split
R² train = 0.761 | R² test = 0.706
ln biomass
Year
Island
Actual (Q=3, τ=1)
Predicted (Q=3, τ=1)
Wasserstein Distance: 7.401263
Island — Takens (Q=3, τ=2) + GPR (H=1)
ln(Total dry wt.) (uniform grid)
GPR one-step (Q=3, τ=2)
train/test split
R² train = 0.768 | R² test = 0.671
ln biomass
Year
Actual (Q=3, τ=2)
Predicted (Q=3, τ=2)
Wasserstein Distance: 7.514114

60