Chirikov-Taylor Map and Chaos Theory

BY: ENRIQUE LOPEZ, CARLOS SOLIS, HAZEL MOORE, NGUYEN LY

C H A O S

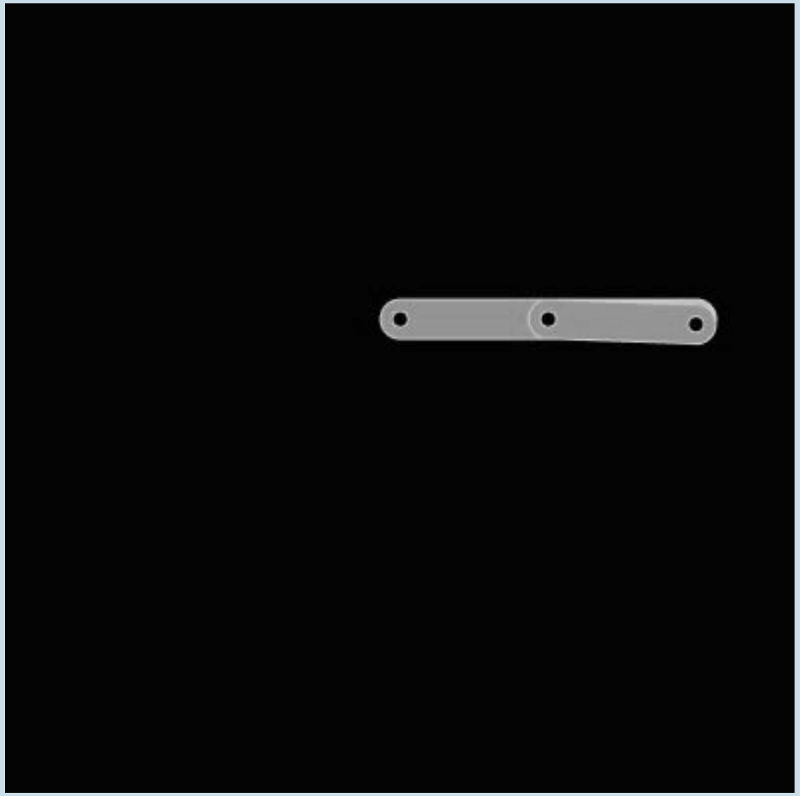
a “state of utter confusion” where “chance is supreme” [1]

~OR~

a dynamical system that exhibits the following characteristics [2]:

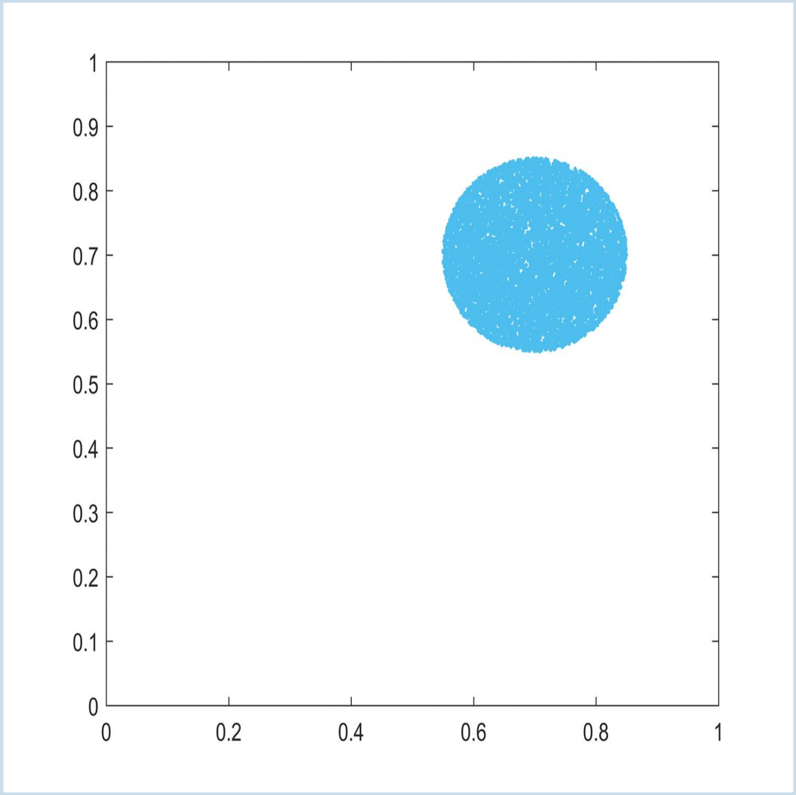
- sensitivity to initial conditions
- topological transitivity
- dense periodic orbits

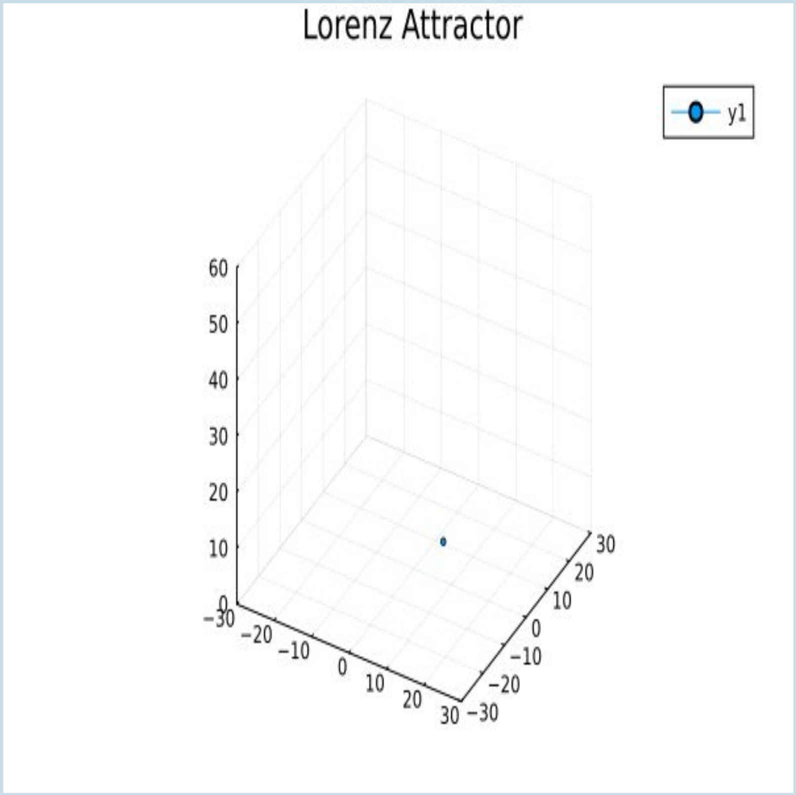
Characteristics of Chaos



Sensitivity to Initial Conditions

Topological Transitivity [3]





Dense Periodic Orbits [4]

So how do we make a chaotic system [5]?

01

Get a discrete system of any dimension:

CHIRIKOV MAP

LOGISTIC MAP

02

Get an infinite-dimensional continuous system:

NAVIER-STOKES EQUATIONS

03

Get a continuous, nonlinear finite-dimensional system with at least three dimensions.

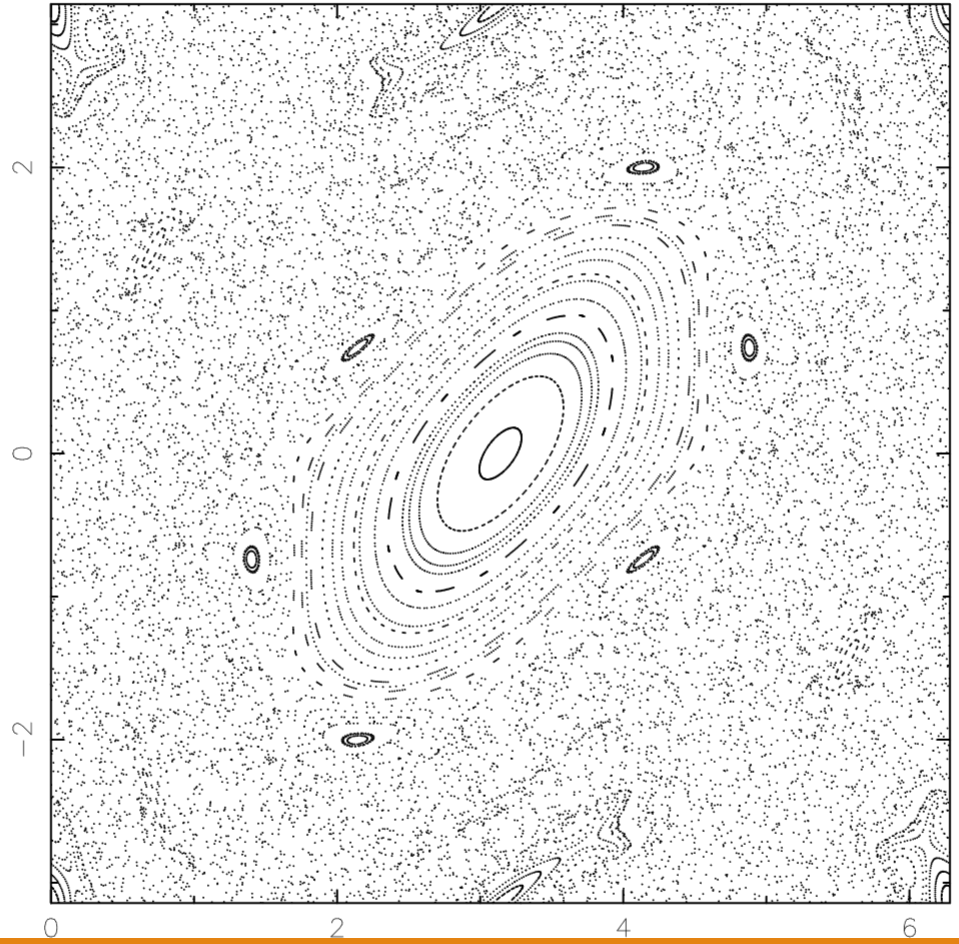
CHUA'S CIRCUIT

LORENTZ MODEL

Standard Map

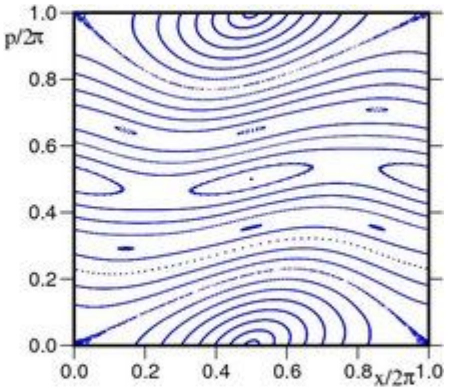
$$I_{n+1} = I_n + K \sin \theta_n$$
$$\theta_{n+1} = \theta_n + I_{n+1}$$

- Chaotic map – an evolution function (or, map) whose outputs go back into its input, that exhibits some form of chaotic behavior.
- “Standard” Map – many dynamical system problems simplify down to the standard map, hence, standard

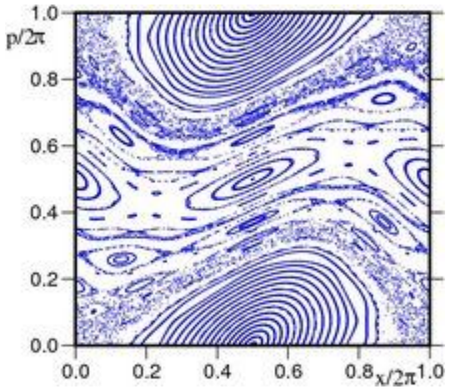


Transition to Chaos

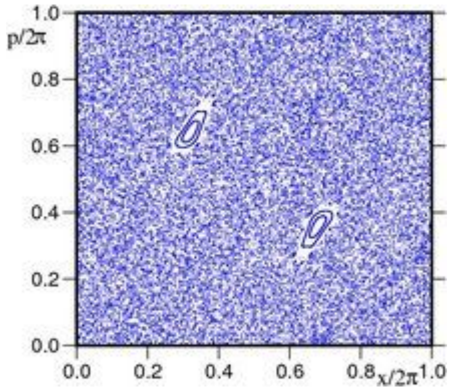
- According to the Kolmogorov-Arnold-Moser theorem, with small enough perturbation K, an invariant curve remains, meaning that the system will not devolve to chaos.
- The critical parameter at which chaos happens is not exactly known, but is between ~.971-.985. [6]



K = 0.5



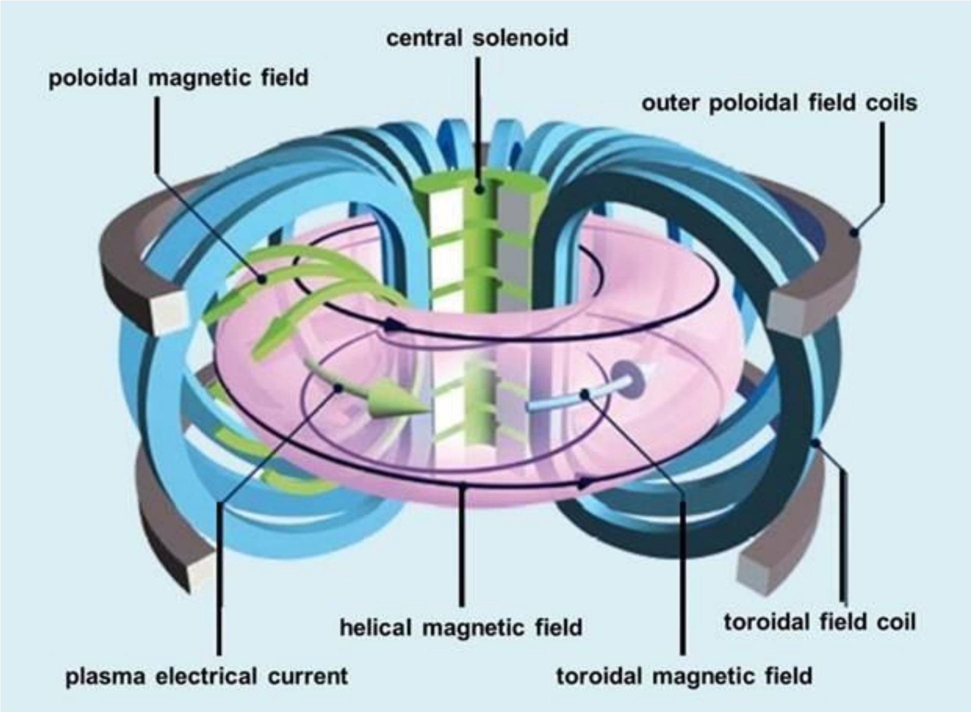
K ≈ 0.971



K = 5

Application

- First described in a paper on nonlinear resonance and stochasticity by Chirkov
- Applications include particle accelerator dynamics, confined plasmas, comet dynamics in a solar system, charged particle confinement, etc.



Code implementation

The code is split into the implementation of the map, the plotting, and the data analysis.

4. Suggested Workflow in `map.ipynb`

1. Simulate using `StandardMap`.
2. Pass `aMap.runs` into `MapEvaluator`.
3. Use functions from `mapPlot.py` to visualize.

```
map
    __init__.py
    standardMap.py
plots
    __init__.py
    mapEval.py
    mapPlot.py
tests
    __init__.py
    test_MapEvaluator.py
    test_standardMap.py
```

Main class: `standardMap`

- Stores the time trajectory of the two state variables
- Supports command-line querying and CSV outputs

(class) StandardMap

A class to store the trajectories of the Chirikov-Taylor Map. Available to use from a command line interface.

Attributes

K : *float*
A nonnegative "kick value" that provides an angular momentum boost. See `simulate` for more details.

nIters : *int*
The number of iterations to travel through. Must be positive.

seed : *int or None*
Used to store information about NumPy's random number generator.

runs : *list of dict*

Methods

simulate(option="append", ic=1)

Iterates through the map from an initial condition or a batch of initial conditions.

metadata(**options)

Returns information about runs or a list of indices satisfying a search term.

clearRuns(**options)

Removes specified runs from the `runs` list.

write(**options)

Writes the specified runs to a `.csv` file using NumPy's `savetxt` function.

read(fname, **options)

Reads in `.csv` files to store in the `runs` list.

1. Running a Simulation

```python
from map.standardMap import StandardMap

# Create a StandardMap instance
aMap = StandardMap(K=0.5, nIters=2000, seed=42)

# Run a simulation with 100 random initial conditions
aMap.simulate(ic=100)

# The results are stored in aMap.runs
print(len(aMap.runs))
print(aMap.runs[0]["run"].shape)
```

Data analysis class: `MapEvaluator`

- Extracts catalogued runs from `standardMap` to produce:
  - phase space trajectories
  - “steady-state” behavior (last n iterations)
  - bifurcation plot arrays

2. Evaluating Simulation Data

```python
from plot.mapEval import MapEvaluator

evaluator = MapEvaluator(aMap.runs)
```

Extracting arrays:

```python
theta = evaluator.getTheta(run_idx=0)
I_vals = evaluator.getI(run_idx=0)
theta_tail = evaluator.thetaTail(0, n_tail=200)
I_tail = evaluator.ITail(0, n_tail=200)
```

Bifurcation:

```python
K_theta, theta_bif = evaluator.thetaBifData(n_tail=200)
K_I, I_bif = evaluator.IBifData(n_tail=200)
```

Phase space:

```python
I_phase, theta_phase = evaluator.phaseSpaceData(run_idx=0, n_tail=200)
```

```
(class) MapEvaluator
```

Helper class for analyzing batches of runs of the Standard (Chirikov) Map.

Parameters

runs : *list of dict*

Typically `aMap.runs` from a `standardMap` instance. Each dict is expected to contain at least the keys `"K"` (float) and `"run"` (np.ndarray of shape (nSim, 2, nIters)).

Premade plotting script: `mapPlot.py`

- Uses mapEvaluator results to produce publication-quality figures: phase space plot, bifurcation diagram, steady-state behavior

```
(function) def plot_phase_tail(
    evaluator: Any,
    run_idx: int = 0,
    n_tail: int = 100,
    point_size: float = 0.1,
    title: str = "Phase Space (tail points)"
) -> None
```

Plot a phase-space diagram using the tail data from a MapEvaluator.

This function uses the last `n_tail` points from all trajectories in a single run and plots them as a scatter cloud in (theta, I) space.

Parameters

evaluator : *MapEvaluator*
Instance constructed from a list of standard map runs. Expected to provide a `phaseSpaceData(run_idx, n_tail)` method that returns flattened I and theta arrays.

run_idx : *int, optional*
Index of the run in the underlying runs list. Default is 0.

n_tail : *int, optional*
Number of final iterations to use from each trajectory. Default is 100.

point_size : *float, optional*
Marker size passed to `plt.scatter`. Default is 0.1.

title : *str, optional*
Plot title. Default is "Phase Space (tail points)".

3. Plotting

```
from plot.mapPlot import (
    plot_phase_generic,
    plot_phase_tail,
    plot_bifurcation_theta,
    plot_bifurcation_I,
)
```

Phase-space plot:

```
run0 = aMap.runs[0]["run"]
plot_phase_generic(run0, mode="phase", point_size=0.1)
```

Tail-only:

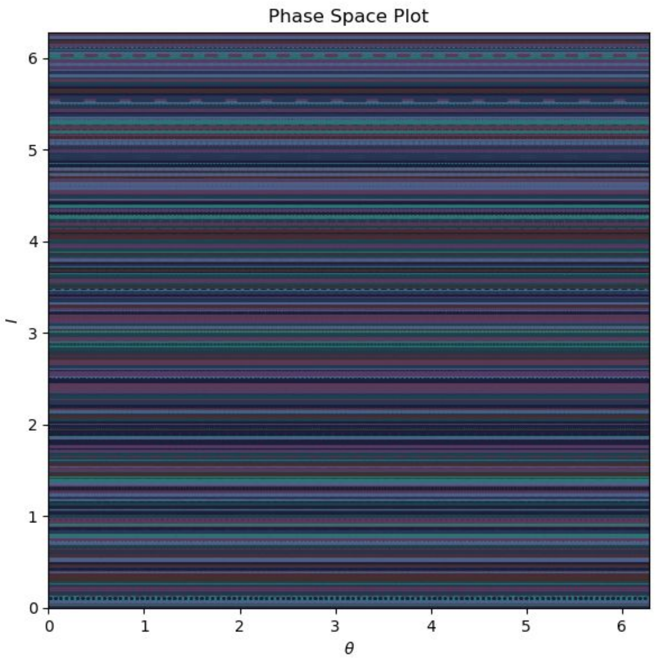
```
plot_phase_tail(evaluator, run_idx=0, n_tail=200, point_size=0.1)
```

Bifurcation:

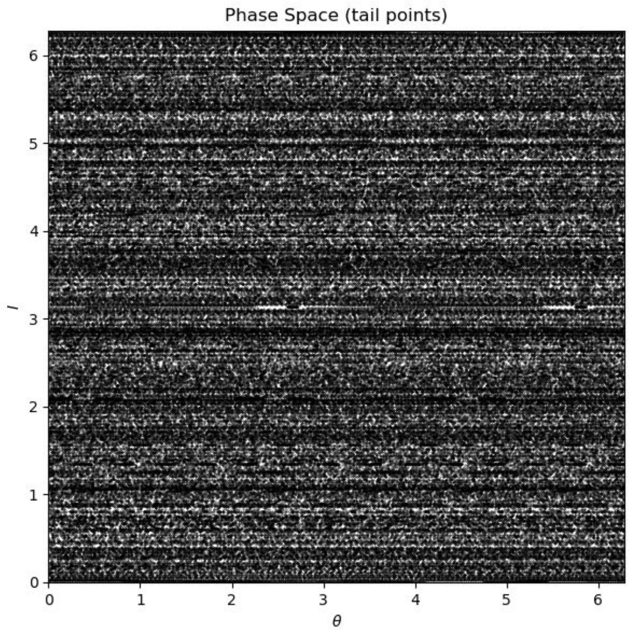
```
plot_bifurcation_theta(evaluator, n_tail=200)
plot_bifurcation_I(evaluator, n_tail=200)
```

Data Analysis

2000 random (I, θ) initial points, 2000 iterations



K = 0.0

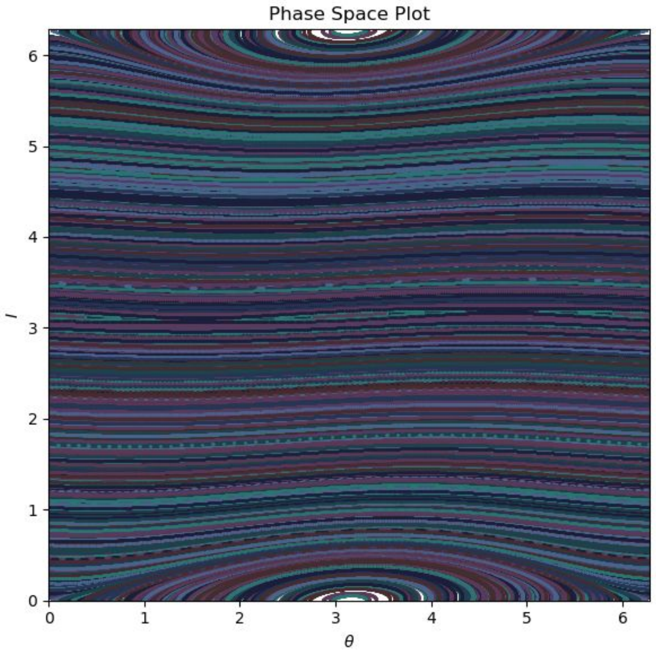


K = 0.0

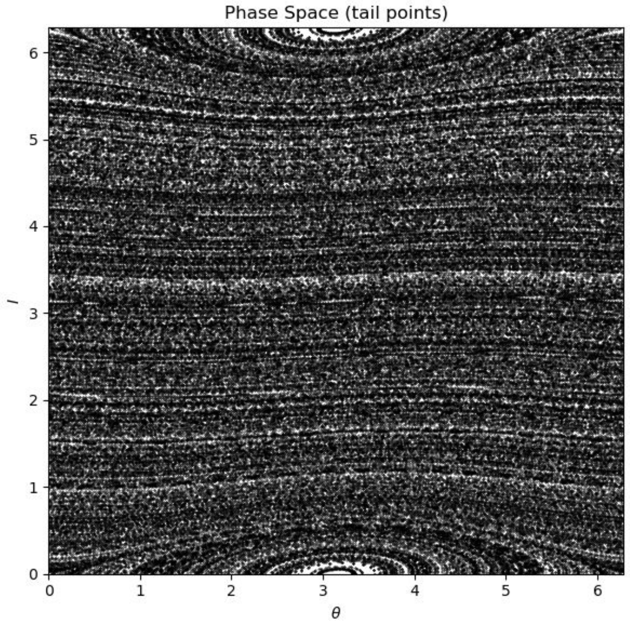
Last 100 time steps

Data Analysis

2000 random (I, θ) initial points, 2000 iterations



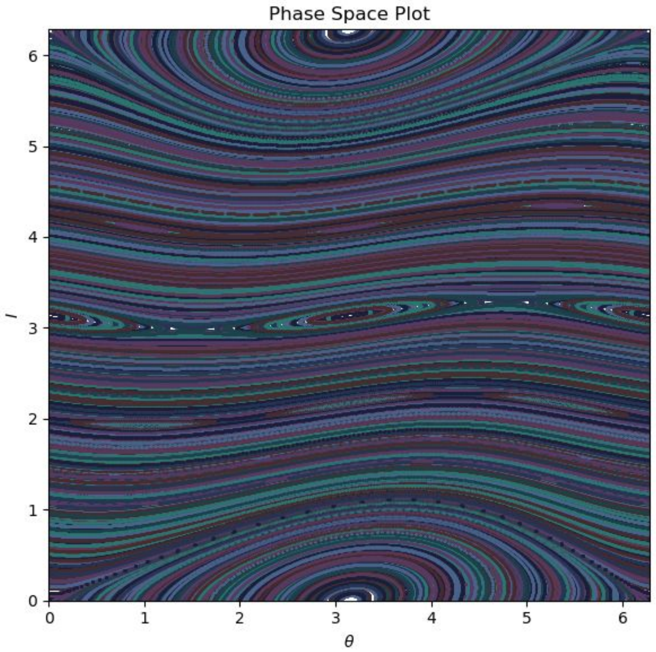
K = 0.1



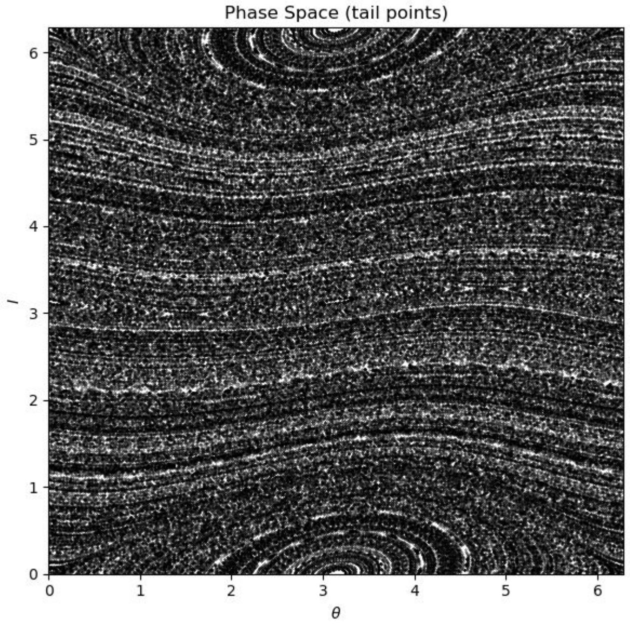
K = 0.1
Last 100 time steps

Data Analysis

2000 random (I, θ) initial points, 2000 iterations



K = 0.3

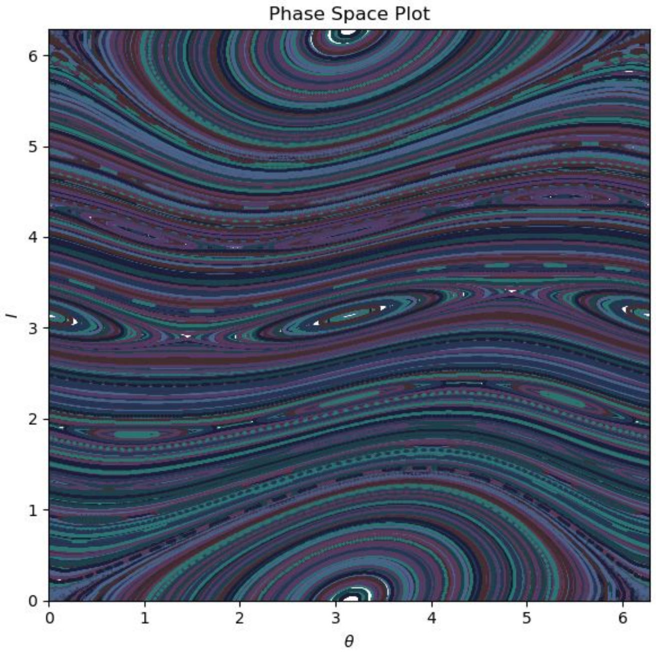


K = 0.3

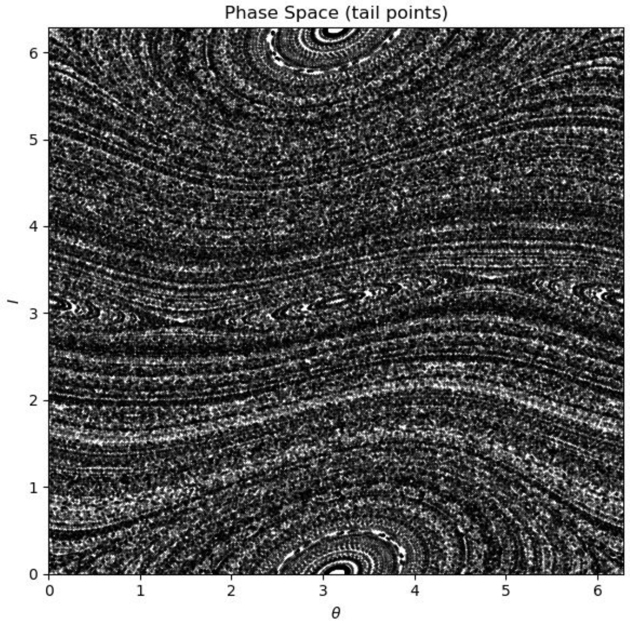
Last 100 time steps

Data Analysis

2000 random (I, θ) initial points, 2000 iterations



K = 0.5

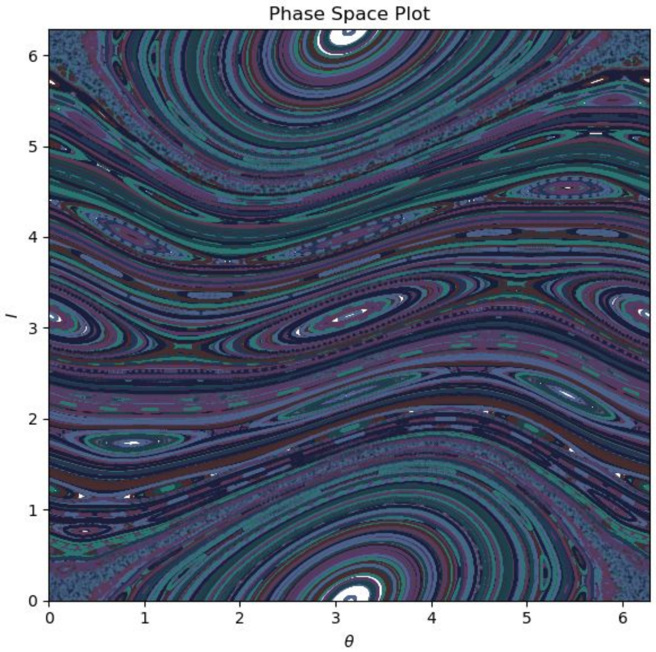


K = 0.5

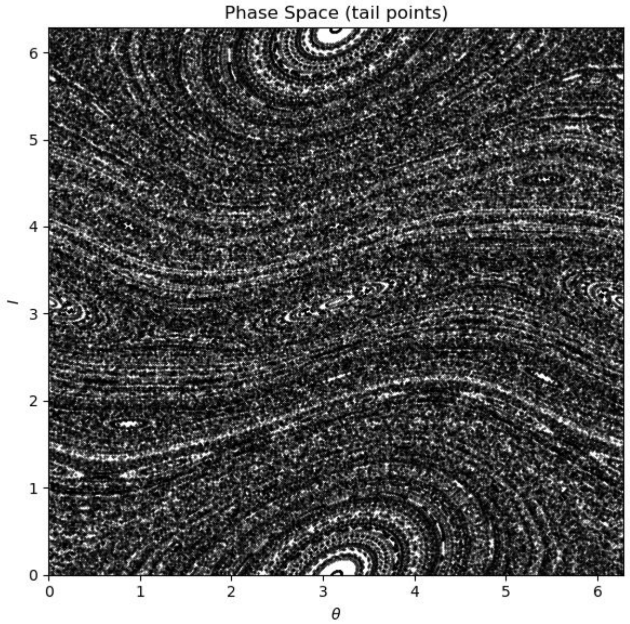
Last 100 time steps

Data Analysis

2000 random (I, θ) initial points, 2000 iterations



K = 0.7

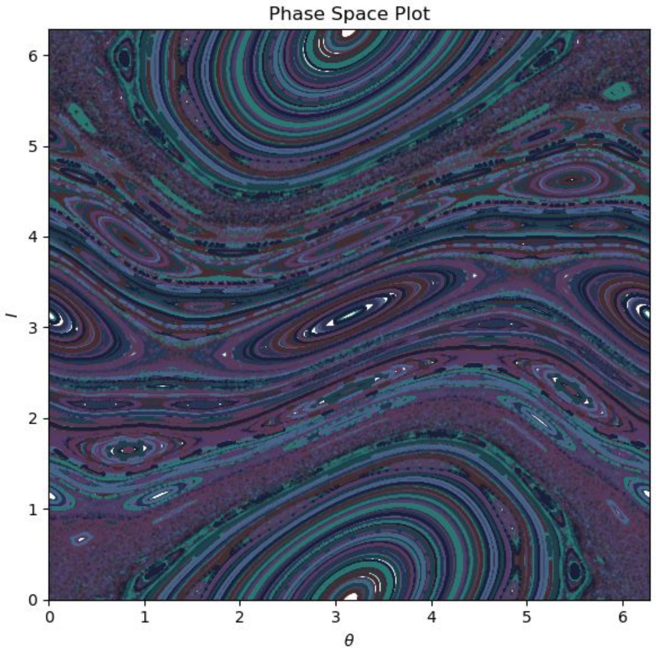


K = 0.7

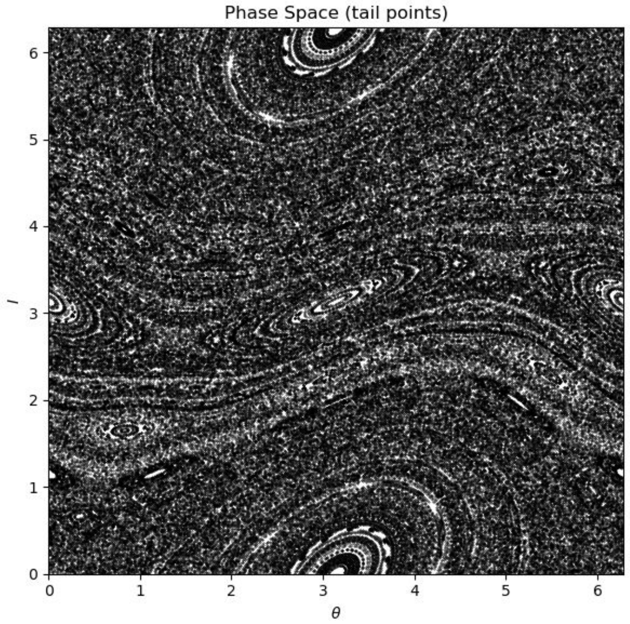
Last 100 time steps

Data Analysis

2000 random (I, θ) initial points, 2000 iterations



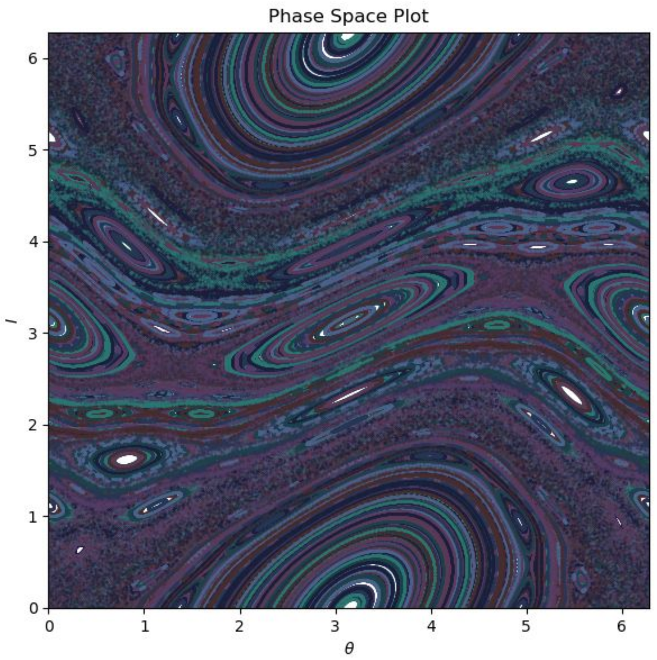
K = 0.9



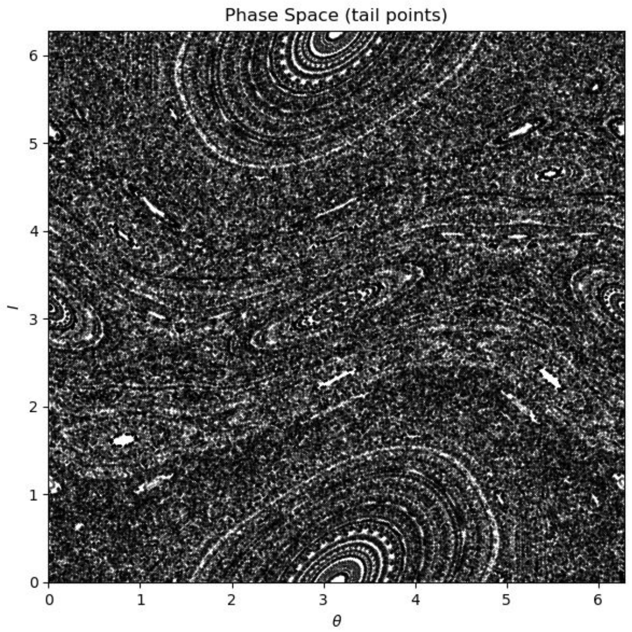
K = 0.9
Last 100 time steps

Data Analysis

2000 random (I, θ) initial points, 2000 iterations



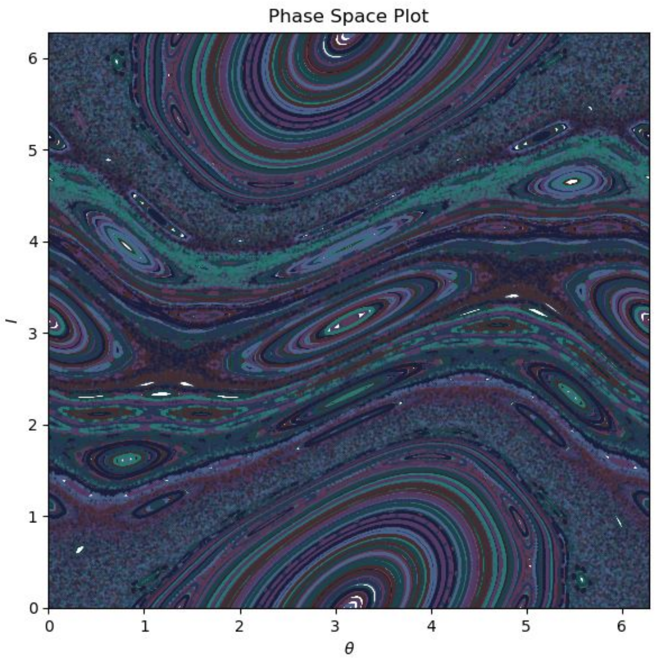
K = 0.9715



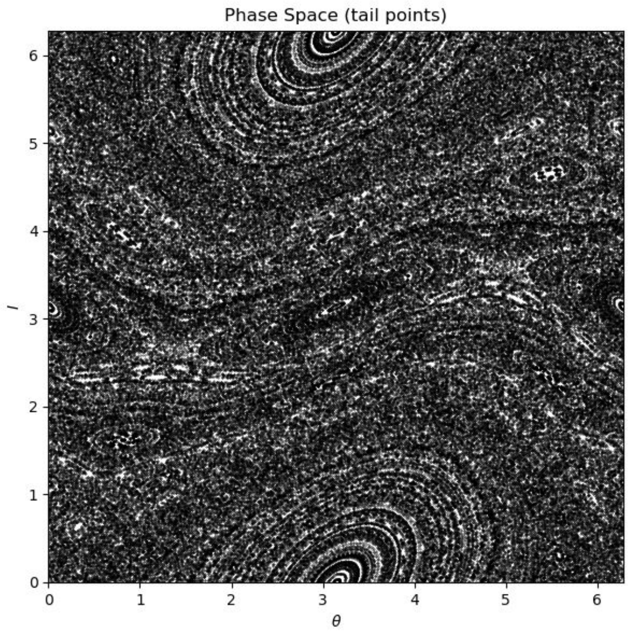
K = 0.9715
Last 100 time steps

Data Analysis

2000 random (I, θ) initial points, 2000 iterations



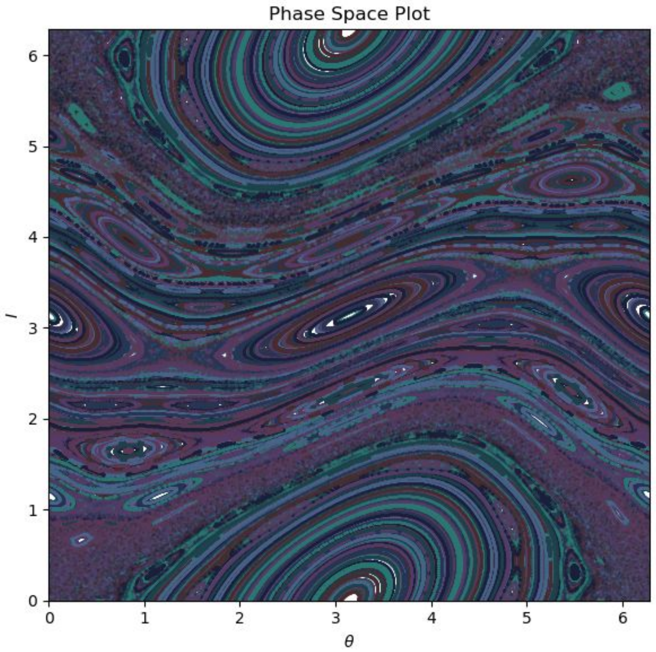
K = 0.9716



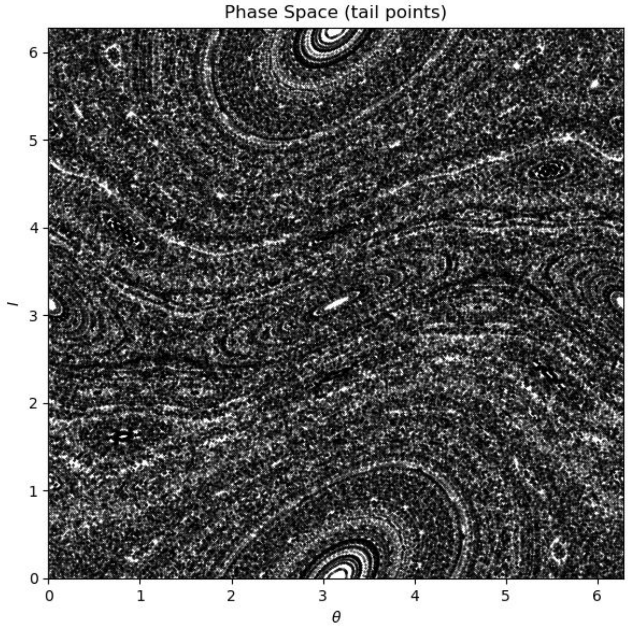
K = 0.9716
Last 100 time steps

This is where Greene estimated chaos to occur: ~0.971635406 [7]

Data Analysis

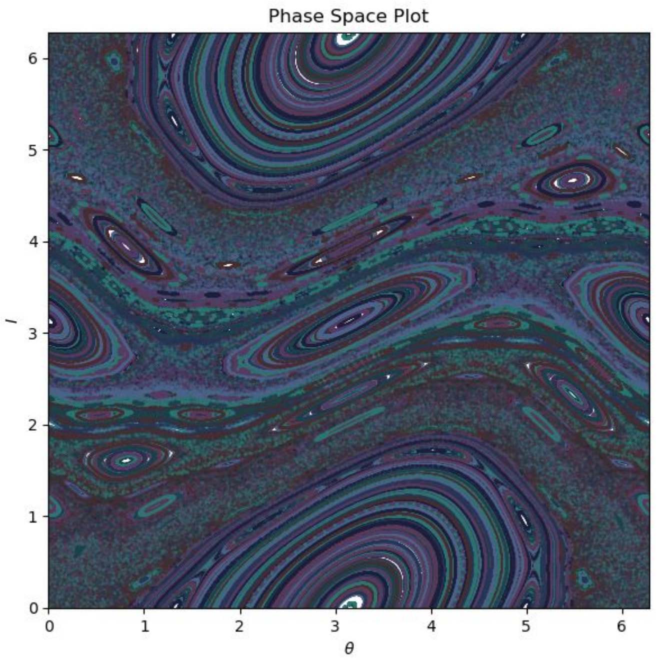


K = 0.9717

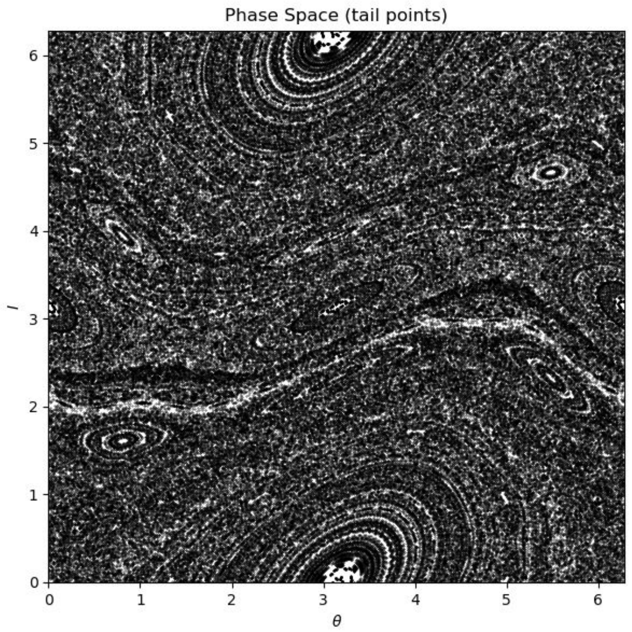


K = 0.9717
Last 100 time steps

Data Analysis

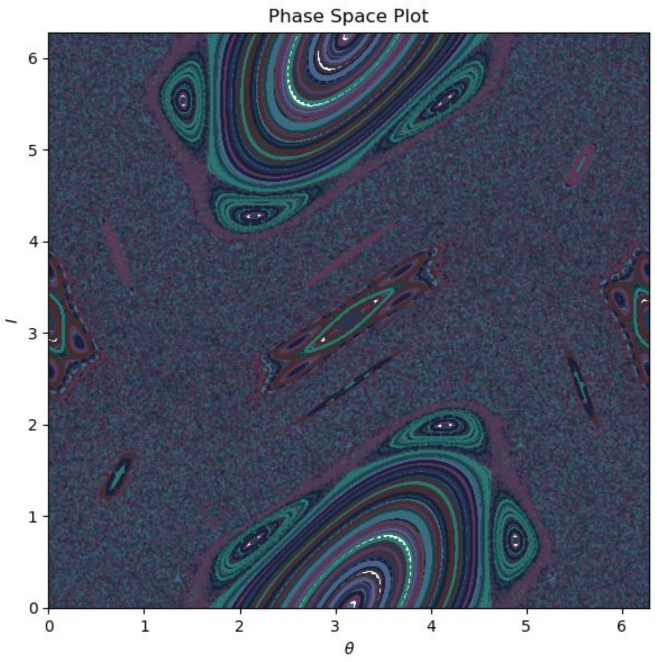


K = 1

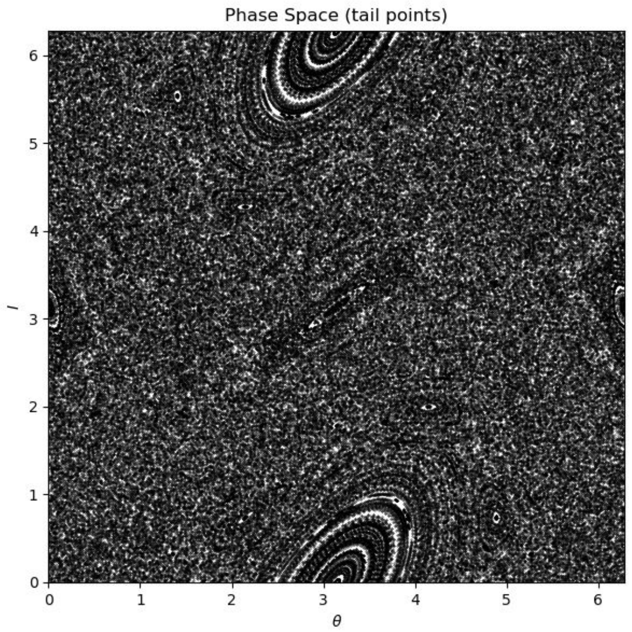


K = 1
Last 100 time steps

Data Analysis

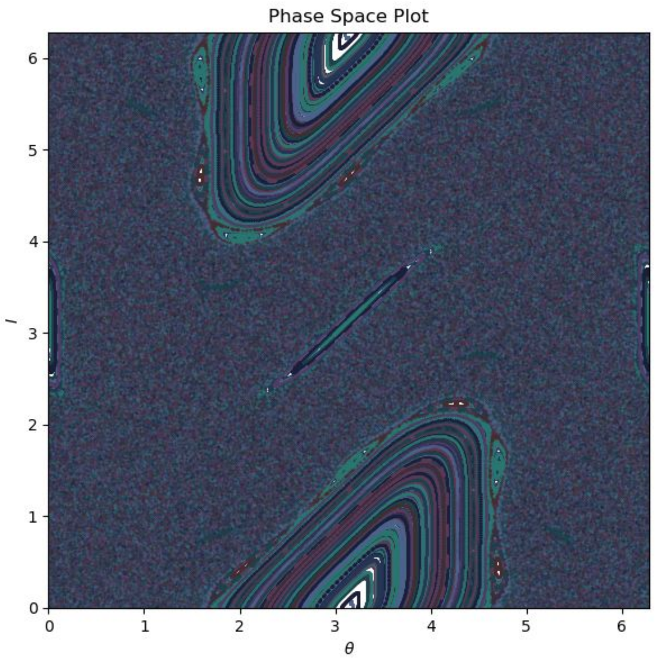


K = 1.5

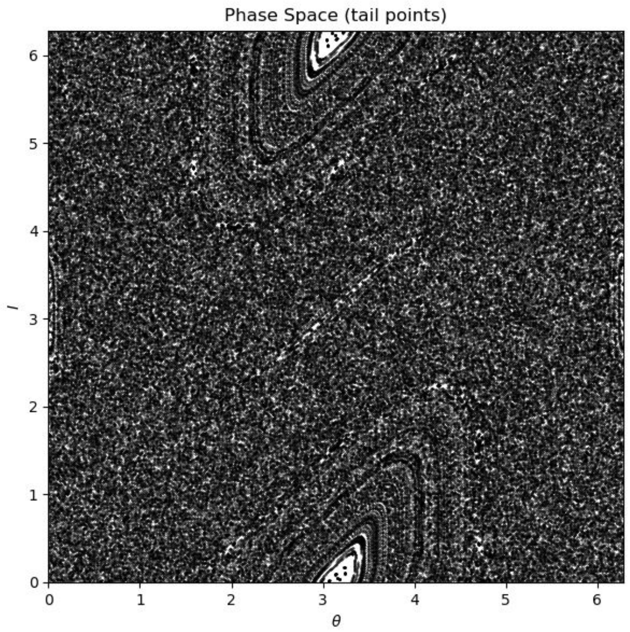


K = 1.5
Last 100 time steps

Data Analysis

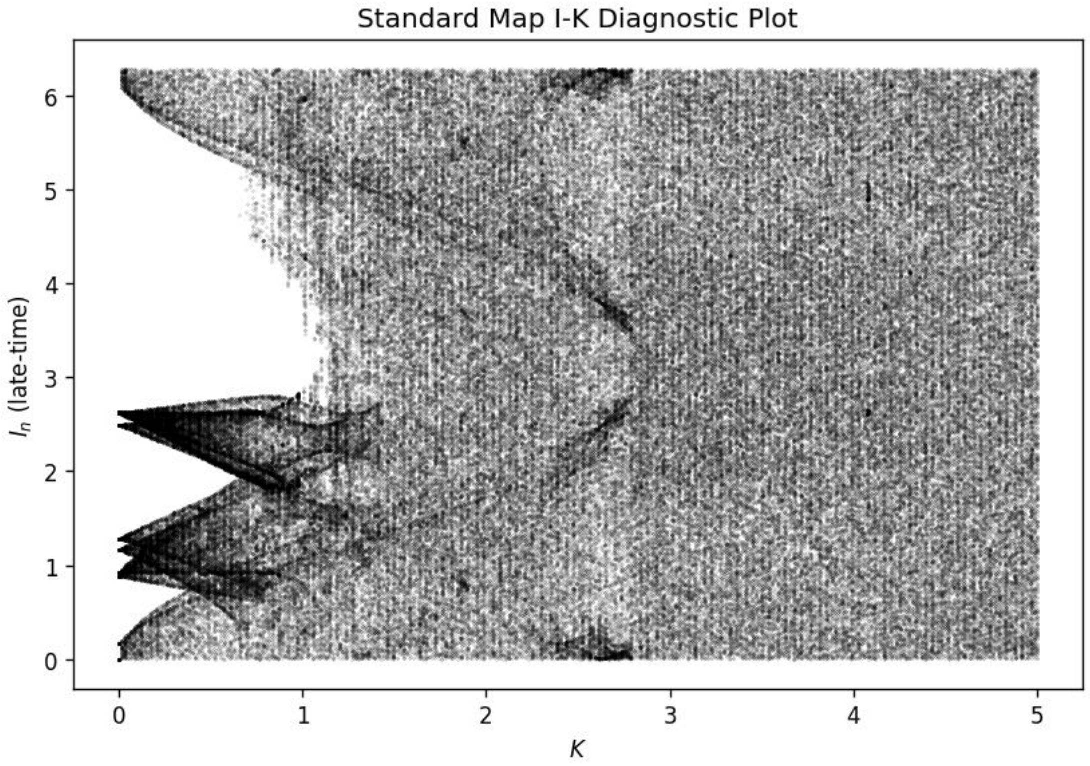


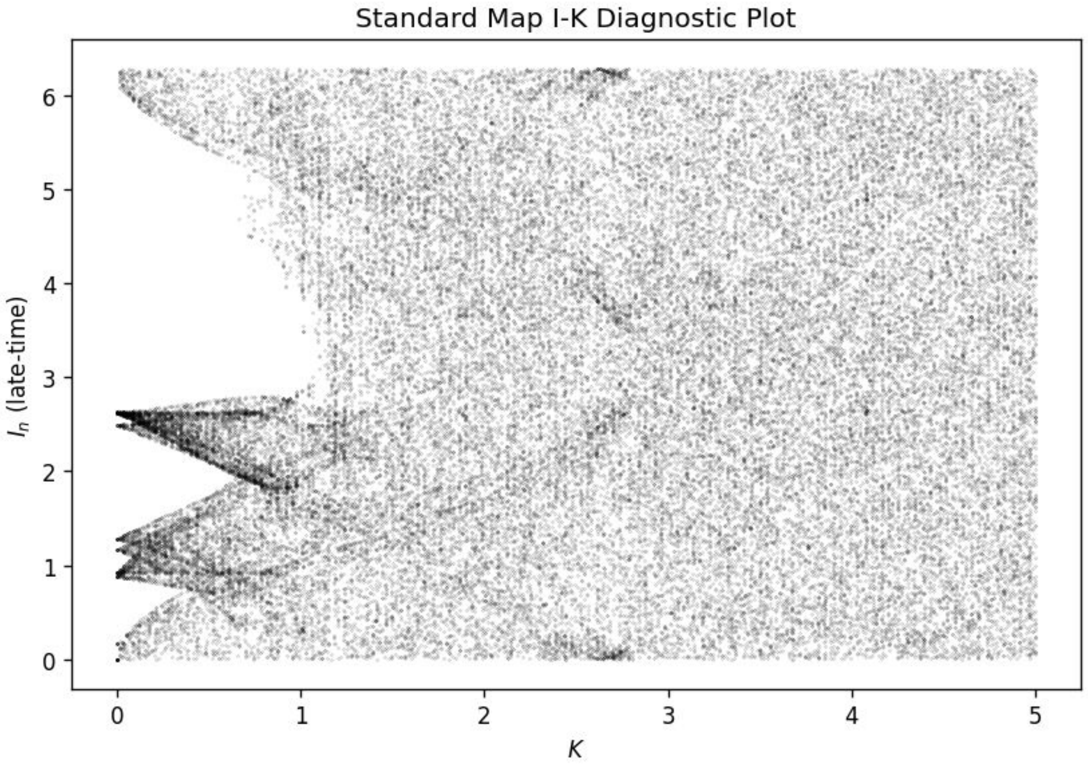
K = 2



K = 2
Last 100 time steps

Data Analysis





Future Improvements

- Combine `MapEvaluator` and `StandardMap` into one streamlined class
  - Allows for `MapEvaluator` to have command-line interactions
  - Easier to work with one collective module than several classes in separate folders
- Option to combine runs with identical attributes {K, nIters, seed}
  - Frees up memory to hold more varied runs
- Auto-saving option for generated plots
  - Useful when generating a series of phase space frames for an animation
  - More robust when used in external scripts

Research Directions

- Expand to a continuous-time system: the Kicked Rotator
- Study real examples of particle motion
- Study the entropy of the system, such as the Kolmogorov-Sinai entropy

References

[1] Merriam-Webster, “Chaos,” retrieved April 3, 2025, https://www.merriam-webster.com/dictionary/chaos

[2] Hasselblatt, Boris, Katok, Anatole, A First Course in Dynamics: With a Panorama of Recent Developments, (Cambridge University Press, 2003)

[3] N3kromancer732, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons

[4] Fincho64, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons

[5] Teschl, Gerald, Ordinary Differential Equations and Dynamical Systems, (Providence: American Mathematical Society, 2012)

[6] Boris Chirikov and Dima Shepelyansky (2008) Chirikov standard map. Scholarpedia, 3(3):3550.

[7] Weisstein, Eric W. "Standard Map." From MathWorld--A Wolfram Resource. https://mathworld.wolfram.com/StandardMap.html