2024 Chaos & Complexity Assignment

SF Theoretical Physics
David Lawton
22337087

14 March 2024

Contents

1 Introduction **2**

2 Analytic Description **2**
2.1 Fixed Point P_1 3
2.2 Fixed Point P_2 3

3 Computational Description **3**

4 Appendix **8**
4.1 Iterative Plotting 8
4.2 Animated Plotting 8

1 Introduction

In this assignment, I studied the Standard Map, first analytically, then computationally.

The Chirikov standard map is an area-preserving map for two canonical dynamical variables, p^i and q^i, that is to say it is a mapping which treats the phase fluid of the (p^i,q^i) space as incompressible. It is described by the iterative map ψ

$$\psi : (p^i, q^i) \to (p^i, q^i) \tag{1}$$

where ψ is defined as

$$\begin{aligned} p_{n+1} &= p_n + k \cdot \sin(q_n) \\ q_{n+1} &= q_n + p_{n+1} \end{aligned} \tag{2}$$

and p_n and q_n are restricted to the interval $[0, 2\pi)$, $k \in \mathbb{R},\ k > 0$.

The map has uses throughout many areas of physics, including but not limited to electromagnetism, astrophysics, particle accelerators, and perhaps most of all, quantum mechanics.

There are also several open problems concerning the map that are yet to be solved.

2 Analytic Description

The first step in the analytic description of the Standard Map was to locate the fixed points. These are the point which are mapped to themselves. By setting $p_{n+1} = p_n$, and $q_{n+1} = q_n$, it is trivial to deduce the two fixed points present.

$$P_1 = (0, 0),\ P_2 = (0, \pi) \tag{3}$$

Next we determined the Jacobi matrix, J, of the map. The Jacobi matrix of a map, $\vec{f}(\vec{x})$, $\vec{x} = (x_1, x_2)$, is defined as

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix} \tag{4}$$

For the Standard Map we define f_1 and f_2 by,

$$\begin{aligned} f_1 &= q_n + p_n + k \cdot \sin(q_n) \\ f_2 &= p_n + k \cdot \sin(q_n) \end{aligned} \tag{5}$$

The Jacobi matrix for this map is then

$$J(\psi) = \begin{pmatrix} 1 + k \cdot \cos(q_n) & 1 \\ k \cdot \cos(q_n) & 1 \end{pmatrix} \tag{6}$$

Following from this, we must find the eigenvalues of the Jacobi matrix. To find the eigenvalues, we first calculated the characteristic equation.

$$\begin{aligned} \det(J - \lambda \mathbb{I}) &= 0 \\ \therefore\ \lambda^2 - (2 + k\cos(q_n))\lambda + 1 &= 0 \end{aligned} \tag{7}$$

filling in for P_1, P_2 (Eq. 3) we get two sets of two eigenvalues.
For P_1,

$$\lambda_{1,2} = \frac{1}{2}\left(2 + k \pm \sqrt{k(k+4)}\right) \tag{8}$$

and for P_2,

$$\lambda_{1,2} = \frac{1}{2}\left(2 - k \pm \sqrt{k(k-4)}\right) \tag{9}$$

2.1 Fixed Point P_1

From the characteristic equation at P_1, Eq. 8, one can analyse the stability of the fixed point. The criterion for stable fixed points in a map is $|\Re\mathrm{e}[\lambda_1]|, |\Re\mathrm{e}[\lambda_2]| < 1$. The eigenvalues of the first fixed point do not meet the fixed point stability criterion.

$$\begin{aligned} \lambda_1 &= \frac{1}{2}\left(2 + k + \sqrt{k(k+4)}\right) > 1, \ \forall\, k > 0, \ k \in \mathbb{R} \\ \lambda_2 &= \frac{1}{2}\left(2 + k - \sqrt{k(k+4)}\right) \ \ \therefore\ 0 < \lambda_2 < 1, \ \forall\, k > 0, \ k \in \mathbb{R} \end{aligned} \tag{10}$$

These values imply that P_1 is a repeller for all values of k.

2.2 Fixed Point P_2

Using the aforementioned stability criterion for fixed points, the second fixed points eigenvalues can be analysed.

$$\begin{aligned} \lambda_1 &= \frac{1}{2}\left(2 - k + \sqrt{k(k-4)}\right) \\ \lambda_2 &= \frac{1}{2}\left(2 - k - \sqrt{k(k-4)}\right) \end{aligned} \tag{11}$$

These eigenvalues are more complicated as they are only complex for certain values of k. However the stability criterion is met at some values of k.

$$\begin{aligned} \text{For } k < 4: \ |\Re\mathrm{e}[\lambda_1]| &< 1 \\ |\Re\mathrm{e}[\lambda_2]| &< 1 \end{aligned} \tag{12}$$

Since we are only interested in stable fixed points, we will not discuss P_2 for values of $k > 4$. Although it would appear to satisfy the criteria for a node for all those values.

3 Computational Description

In my computational description of the standard map I took two separate approaches. The first approach concerned plotting the first thousand iterations for a given k on a single plot. this gives a good illustration of fixed points and bounded loops, et cetera. The second approach was to animate the system through iterations.

The first case to observe is for small k (Fig. **1**). By looking at the mathematical description, one would expect points to move along the q direction only, and it's evident that each point moves only across the space horizontally, when looking at the figure, except for points with p_n near 0 or 2π. For this value of k I have only included 100 iterations as it illustrates well how much the movement depends on the p_n component.

The second case (Fig. **2**) was for a slightly larger k such that it is not negligible for any point.The fixed point P_2 becomes obvious here with the elliptical motion centered around it. The lines not inside the radius of the circle are those which the points on them move along.

In the third case (Fig. **3**) k was increased slightly once again. Here, some extremely interesting evident closed loops, not corresponding to fixed points, emerge between each curve between the main loop, and yet still there is little to no mixing of initial horizontal lines (loops are made almost completely of one colour). It is notable that the curve bounding the main loop that passes through the origin begins to degenerate here. The fourth case (Fig. **4**)corresponds to a theoretically important value of k, $k = 0.971635....$ It is a critical irrational value of k for which chaos begin to emerge. This is clear in the mixing around the main loop. Secondary loops form also around all of the previously existing ones, one might wonder if a fractal shape might emerge if even more points were plotted. The fifth case (Fig. **5**), while similar to the fourth,

I found interesting due to the slight changes, in that the secondary loops that formed are becoming smaller, and the chaos rapidly increases. Also notable are the new secondary loops forming from the chaos of the previous case.

The sixth case (Fig. **6**), the 'stream' through the centre has disintegrated almost completely into chaos, with only the largest of the previous loops keeping any kind of shape, yet still being manipulated almost into quadrilaterals. The new secondary loops are also fully emerged, and the sides of the main loop have begun to flatten.

In the final case (Fig. **7**), almost every loop has disappeared, with only the largest in the stream and the main one and its secondaries remaining.

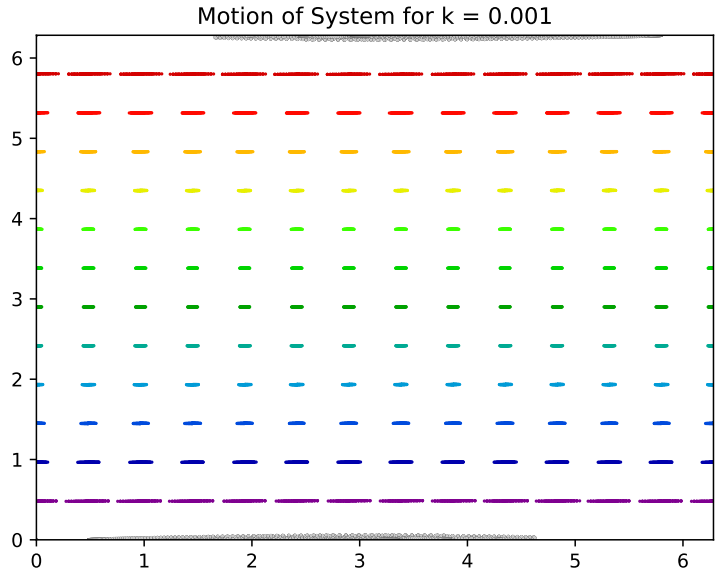

Figure 1: Plot through 100 iterations for $k = 0.001$

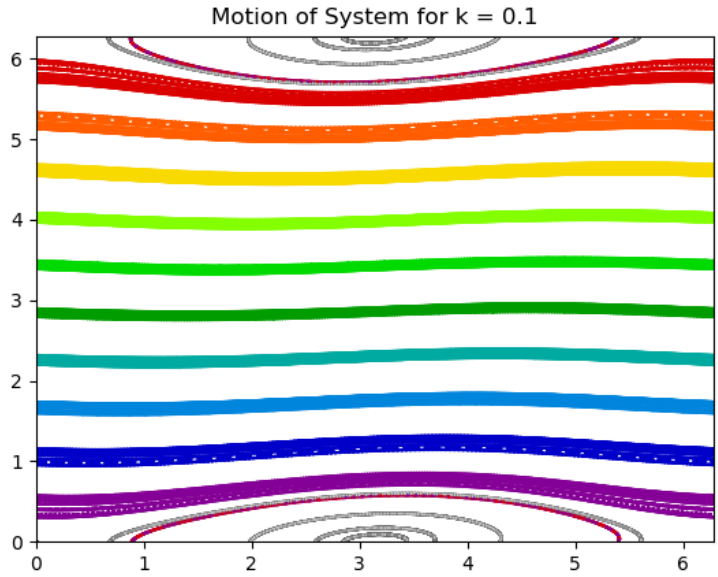

Figure 2: Plot through 1000 iterations, elliptical motion around P_2 is clear.

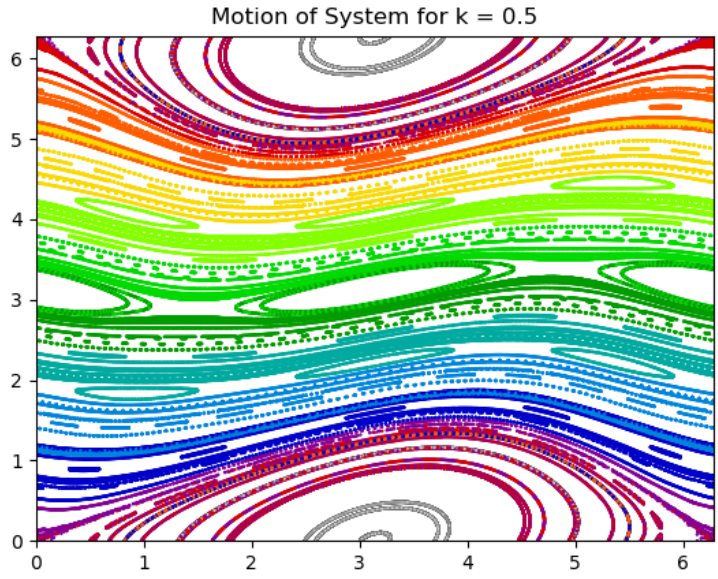

Figure 3: lot through 1000 iterations, elliptical motion around P_2 is clear.

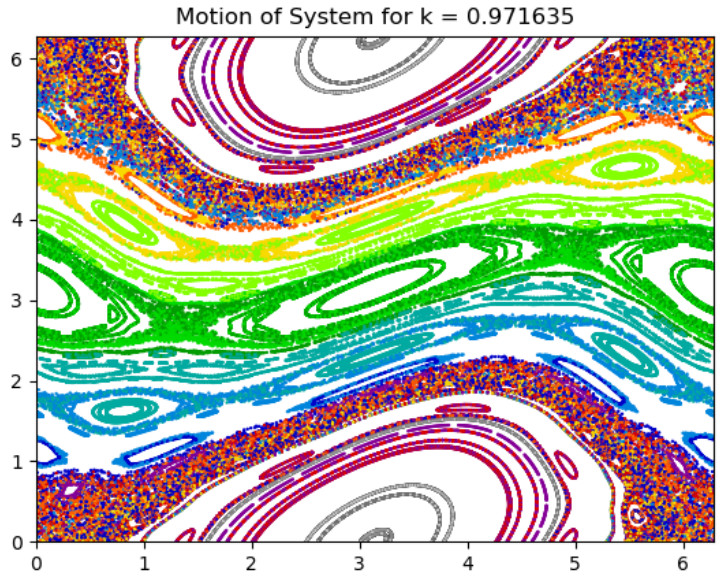

Figure 4: Plot through 1000 iterations, first instance of emerging chaos.

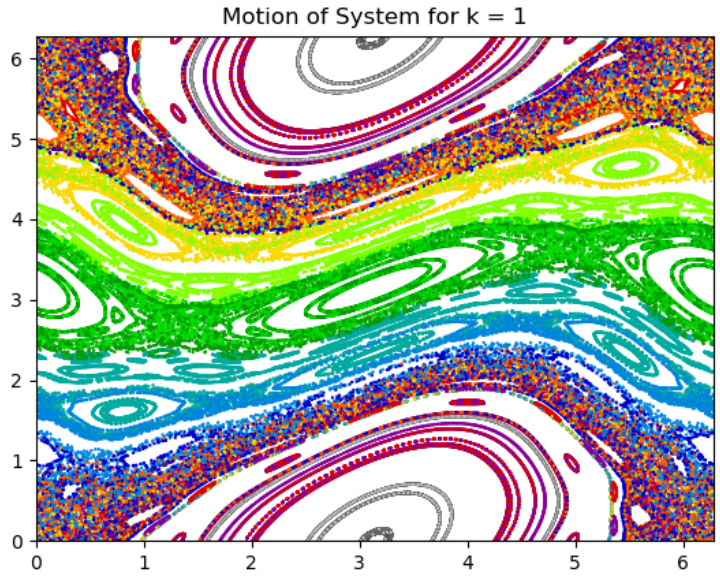

Figure 5: Plot through 1000 iterations, quite a rapid change in chaos in such a small change from the previous k.

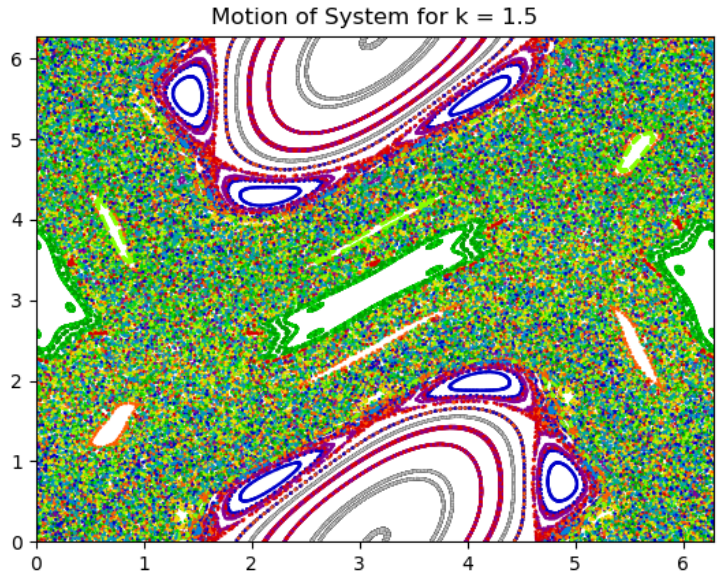

Figure 6: Plot through 1000 iterations, the secondary closed loops of the main loop are likely the most interesting thing emerging from the chaos.

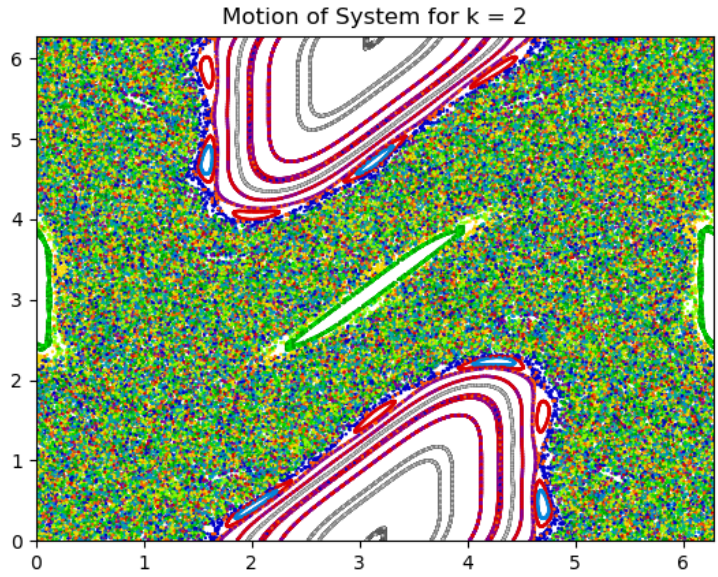

Figure 7: Plot through 1000 iterations, chaos will clearly engulf the map for larger values of k

3.1 Animations

After doing these iterative plots, i made animation with the code for them described below. I found that the showed the motion of the system a bit better, but not patterns as much. They can be found attached in the upload, as it is difficult to add or view .GIF or .MP4 files in Latex, and that is beyond my abilities.

4 Appendix

4.1 Iterative Plotting

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.animation as animation

#Mapping function
def mapfunc(x, y, k):
    p = np.sin(x)
    x = x + y + (k * p)
    y = y + (k * p)
    return x, y


#Evenly spaced set of points between 0 and 2*pi
points = np.linspace(0, 2*np.pi, 12)
x, y = np.meshgrid(points, points)
colour = y.ravel()


#Plotting function
def plotter(k):
    global x,y
    fig, ax = plt.subplots()
    ax.set_xlim(0, 2*np.pi)
    ax.set_ylim(0, 2*np.pi)
    plt.title(f'Motion of System for k = {k}')
    for i in range(1000):
        x, y = mapfunc(x, y, k)
        x = np.mod(x, 2*np.pi)
        y = np.mod(y, 2*np.pi)
        scat = ax.scatter(x, y, s=0.5, c=colour, cmap='nipy_spectral')
        scat.set_offsets(np.c_[x.ravel(), y.ravel()])
    plt.savefig(f'Python/Chaos/ChaosAssignmentstillk{k}.png')


#Create plot for list of k values
for k in [10**-3, 0.1, 0.5, 0.971635, 1, 1.5, 2]:
    plotter(k)
    print(f'k = {k} done')
```

4.2 Animated Plotting

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation


# mapping function
def mapfunc(x, y, k):
    p = np.sin(x)
    x = x + y + (k * p)
    y = y + (k * p)
    return x, y


# Initialization function
def init():
    ax.set_xlim(0, 2*np.pi)
    ax.set_ylim(0, 2*np.pi)
    scat.set_offsets(np.c_[x.ravel(), y.ravel()])  # set the initial state of the scatter
    return scat,


# Animation function
def animate(i):
    global x, y
    if i == 0:
        return scat,
    x, y = mapfunc(x, y, k)
    x = np.mod(x, 2*np.pi)  # wrap x to the interval [0, 2$\pi$]
    y = np.mod(y, 2*np.pi)  # wrap y to the interval [0, 2$\pi$]
    scat.set_offsets(np.c_[x.ravel(), y.ravel()])
    return scat,


# evenly spaced set of points between 0 and 2*pi
points = np.linspace(0, 2*np.pi, 15)
x, y = np.meshgrid(points, points)
colours = y.ravel()



def create_animation(k):
# figure
    global ax, scat
    fig = plt.figure()
    ax = fig.add_subplot(111)
    plt.title(f'Motion of System for k = {k}')
    scat = ax.scatter(x, y, s=0.5, c=colours, cmap='nipy_spectral')
# Create animation
    ani = animation.FuncAnimation(fig, animate, init_func=init, frames=100, interval=10, b
```

```
    ani.save(f'ChaosAssignmentGIFk{k}.gif',
           writer = 'ffmpeg', fps = 8)

for k in [10**-3, 0.1, 0.5, 1, 1.5, 2]:
    create_animation(k)
    print(f'k = {k} done')
```