Numerical Methods for Dynamical Systems
Evaluation of a logistic- & 2d standard map
UPC, FME

Written by
Ole Gunnar Røsholt Hovland

2023

Abstract

In this assignment, we will dynamical systems using the logistic map and the 2D standard map. For specified parameters, simulations reveal patterns in periodic points and invariant curves. Observations on the dynamics and the impact of changing 'a' are analyzed.

Contents

1 Part 1

In Part 1, we focus on the logistic map, represented by the equation F(x) = a * x(1 - x). This map has roots in ecological population models and exhibits varying dynamics based on parameter values. We will simulate and analyze the map using two specific sets of parameters: a = 2, x0 = 0.2, and a = 3.2, x0 = 0.4.

1.1 Creation of simulations

(i) Now we will simulate the standard map dynamics taking 500 iterates for each initial condition. Try with different initial conditions.

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

# Parameters
a = 2.5
x_0 = 0.3
N = 500

# logistic map function
def F(x, a):
    return a*x*(1-x)

# Simulate the map
def single_simulation(x_0, N, a):
    x = np.zeros(N)
    x[0] = x_0
    for i in range(1,N):
        x[i] = F(x[i-1], a)
    return x

# Plotting the map, as well as the identity line and function F
def plot_map(x_sim):
    x = np.linspace(0,1,100)
    plt.plot(x, x, 'k', markersize=1)
    plt.plot(x, F(x, a), 'r', markersize=1)
    plt.plot(x_sim[:-1], x_sim[1:], 'bo', markersize=1)
    plt.plot([x_sim[0],x_sim[0]], [0, x_sim[0]], 'b', markersize=1)
    for i in range(len(x_sim)-1):
        plt.plot([x_sim[i], x_sim[i]], [x_sim[i], x_sim[i+1]], 'b',
                 markersize=1)
        plt.plot([x_sim[i], x_sim[i+1]], [x_sim[i+1], x_sim[i+1]], 'b',
                 markersize=1)
    plt.xlabel(r'$x$')
    plt.ylabel(r'$F(x)$')
    plt.title(r'Map dynamics with $a$ = ' + str(a) +
    'and $x_0$ = ' + str(x_0))
    plt.show()

```

```
plot_map(single_simulation(x_0, N, a))
a, x_0 = 2, 0.2
plot_map(single_simulation(x_0, N, a))
a, x_0 = 3.2, 0.4
plot_map(single_simulation(x_0, N, a))
```

The output of the code above is shown in Figure 1.1.

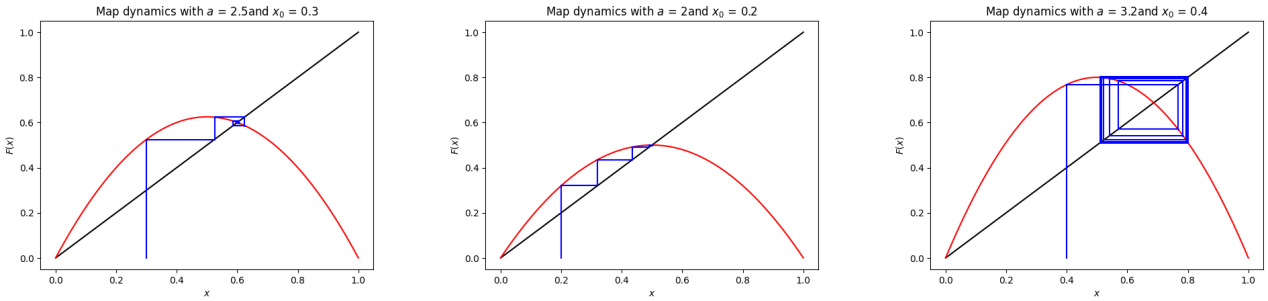

Figure 1.1: Map dynamics with different $a, x_0 = 2.5, 0.3$, $a, x_0 = 2, 0.2$, and $a, x_0 = 3.2, 0.4$ respectively.

(ii) Now, we modify the code such that the program takes NP initial conditions on the x axis i.e. (x,0) with x between xmin and xmax.

```
# Redefining F
def F(x, y, a):
    return x+a*np.sin(x+y), x + y

# Modifying single_simulation to accomodate 2 variables
def single_simulation(x_0, y_0, N, a):
    x = np.zeros(N)
    y = np.zeros(N)
    x[0], y[0] = x_0, y_0
    for i in range(1,N):
        x[i], y[i] = F(x[i-1], y[i-1], a)
    return x, y

def NP_simulation(NP, N, a):
    x_init = np.linspace(0,2*np.pi,NP)
    x = np.zeros(NP*N)
    y = np.zeros(NP*N)
    for j in range(NP):
        x[j*N:(j+1)*N], y[j*N:(j+1)*N] = single_simulation(x_init[j],
                                                           0, N, a)
    # normalizing x and y between -pi and pi
    x = np.mod(x, 2*np.pi)
    y = np.mod(y, 2*np.pi)
    x = x - 2*np.pi*(x>np.pi)
    y = y - 2*np.pi*(y>np.pi)
    return x,y

def plot_NP_simulation(NP, N, a):
```

```
    x, y = NP_simulation(NP, N, a)
    for i in range(NP):
        plt.plot(x[i*N:(i+1)*N], y[i*N:(i+1)*N], '.', markersize=1)
    plt.xlabel('x')
    plt.ylabel('F(x)')
    plt.gcf().set_dpi(200)
    plt.show()

N = 800
a = -0.7
plot_NP_simulation(150, N, a)
```

The output of the code above is shown in Figure 1.2.

1.2 Periodic points

To find the periodic points we will start by roughly finding where it is manually, and then using a newtons method from the Scipy library.

```
# Now modifying the code to zoom in on the plot
def NP_magnify_simulation(NP, N, a, start, end):
    x_init = np.linspace(start, end, NP)
    x = np.zeros(NP*N)
    y = np.zeros(NP*N)
    for j in range(NP):
        x[j*N:(j+1)*N], y[j*N:(j+1)*N] = single_simulation(x_init[j],
                                                           0, N, a)
    x = np.mod(x, 2*np.pi)
    y = np.mod(y, 2*np.pi)
    # normalize x and y between -pi and pi
    x = x - 2*np.pi*(x>np.pi)
    y = y - 2*np.pi*(y>np.pi)
    return x,y

def plot_NP_magnify_simulation(NP, N, a, start, end, magnify=True):
    x, y = NP_magnify_simulation(NP, N, a, start, end)
    for i in range(NP):
        plt.plot(x[i*N:(i+1)*N], y[i*N:(i+1)*N], '.', markersize=1)
    if not magnify:
        # show the plot from -pi to pi
        plt.xlim(-np.pi, np.pi)
        plt.ylim(-np.pi, np.pi)
    plt.xlabel('x')
    plt.ylabel('F(x)')
    plt.gcf().set_dpi(70)
    plt.show()

N = 500
a = -0.7
start, end = 2.1, 2.4
```

```
plot_NP_magnify_simulation(40, N, a, start, end)
```

The output of the code above is shown in Figure 1.3.

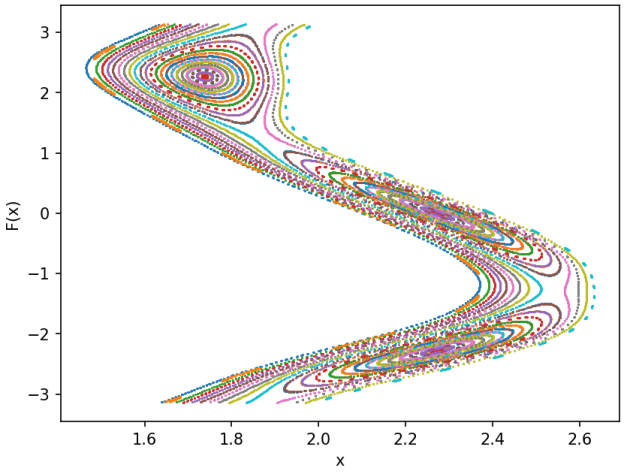

Figure 1.2: From the plot we can achieve a good initial guess for the newtons method

Now, we need to implement the newtons method

```python
#from scipy.optimize import newton_krylov
import scipy.optimize as opt
# Now using the Newton-Krylov method to find the fixed point.
# The Newton-Krylov method is a generalization of Newton's method
# It is a more efficient method for solving systems of nonlinear equations

# Redefining F to be able to use fsolve
def norm_F(p, a):
    x, y = p
    x, y = x+a*np.sin(x+y), x + y

    # Normalizing between -pi and pi

    x = np.mod(x, 2*np.pi)
    y = np.mod(y, 2*np.pi)
    x = x - 2*np.pi*(x>np.pi)
    y = y - 2*np.pi*(y>np.pi)

    return x, y

x0, y0 = 2.1, 0

# Using scipy.optimize to find the fixed point
x, y = opt.newton_krylov(lambda p: norm_F(norm_F(norm_F(p,a), a), a) - p,
                         (x0, y0), f_tol=1e-14)

x = np.mod(x, 2*np.pi)
y = np.mod(y, 2*np.pi)

x = x - 2*np.pi*(x>np.pi)
y = y - 2*np.pi*(y>np.pi)

x_t, y_t = norm_F(norm_F(norm_F((x,y), a), a), a)

print(f"Fixed point: ({x}, {y})")
print(f"Testing if it is a fixed point: {x_t-x} and {y_t-y}")

start, end = x, x
plot_NP_magnify_simulation(70, N, a, start, end, magnify=False)


start, end = x-.1, x+.1
N = 10000
plot_NP_magnify_simulation(1, N, a, start, end, magnify=False)
```

The output of the code is

```
Fixed point: (2.272588585208659, -4.440892098500626e-15)
```

```
Testing if it is a fixed point: 0.0 and 8.881784197001252e-16
```

The plots from the code above can be found bellow.

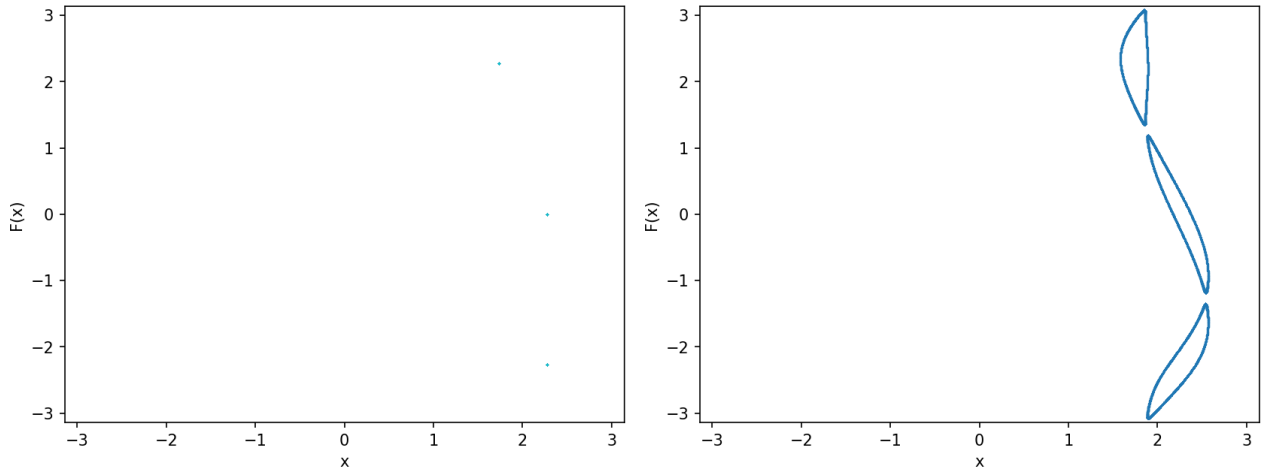

Figure 1.3: Plots of the three 3-periodic points, as well as an orbit around them

Now for the 4 periodic points, doing the same as the 3 periodic ones.

```python
x0, y0 = 1.9, 0

x, y = opt.newton_krylov(lambda p: norm_F(norm_F(norm_F(norm_F(
                        p,a), a), a), a) - p, (x0, y0), f_tol=1e-14)

x = np.mod(x, 2*np.pi)
y = np.mod(y, 2*np.pi)
x = x - 2*np.pi*(x>np.pi)
y = y - 2*np.pi*(y>np.pi)

x_t, y_t = norm_F(norm_F(norm_F(norm_F((x,y), a), a), a), a)

print(f"Fixed point: ({x}, {y})")
print(f"Testing if it is a fixed point: {x_t-x} and {y_t-y}")

start, end = x, x
plot_NP_magnify_simulation(70, N, a, start, end, magnify=False)

start, end = x-.015, x+.015
N = 10000
plot_NP_magnify_simulation(1, N, a, start, end, magnify=False)
```

The output of the code is

```
Fixed point: (1.901797439828015, -4.529709940470639e-14)
Testing if it is a fixed point: 1.7763568394002505e-15 and 8.881784197001252e-16
```

The plots from the code above can be found bellow.

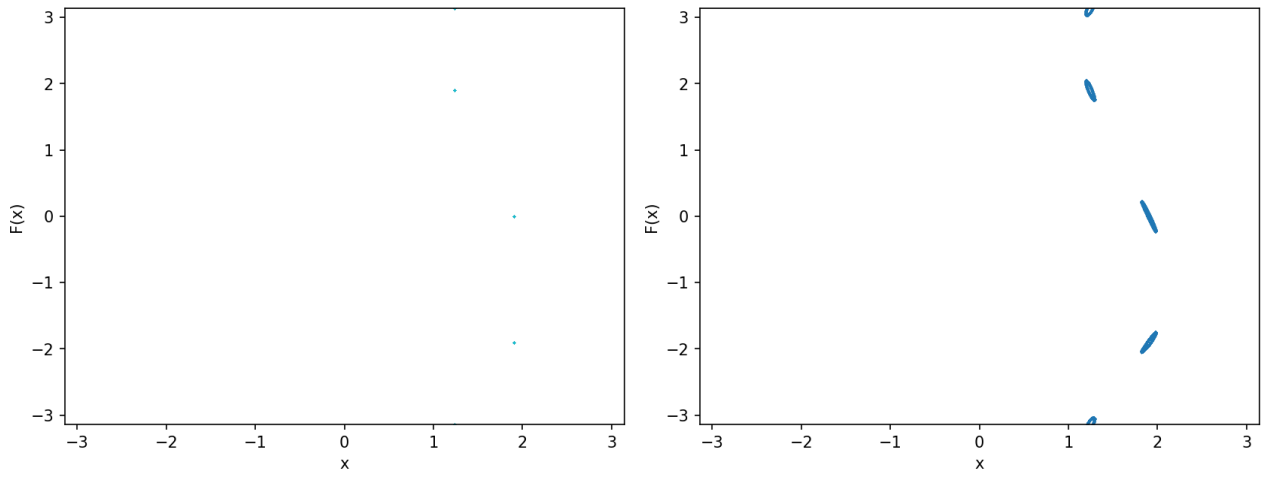

Figure 1.4: Plots of the four 4-periodic points, as well as an orbit around them

1.3 Orbits

We'll visualize a vertical invariant curve and another around the origin, then discuss the dynamics revealed in these plots, enhancing our comprehension of the system's behaviors and underlying patterns.

Plotting a vertical invariant curve, and another one around the origin.

```
start , end = 1.85, 1.85
plot_NP_magnify_simulation(1, N, a, start , end, magnify=False)

start , end = 1,1
plot_NP_magnify_simulation(1, N, a, start , end, magnify=False)
```

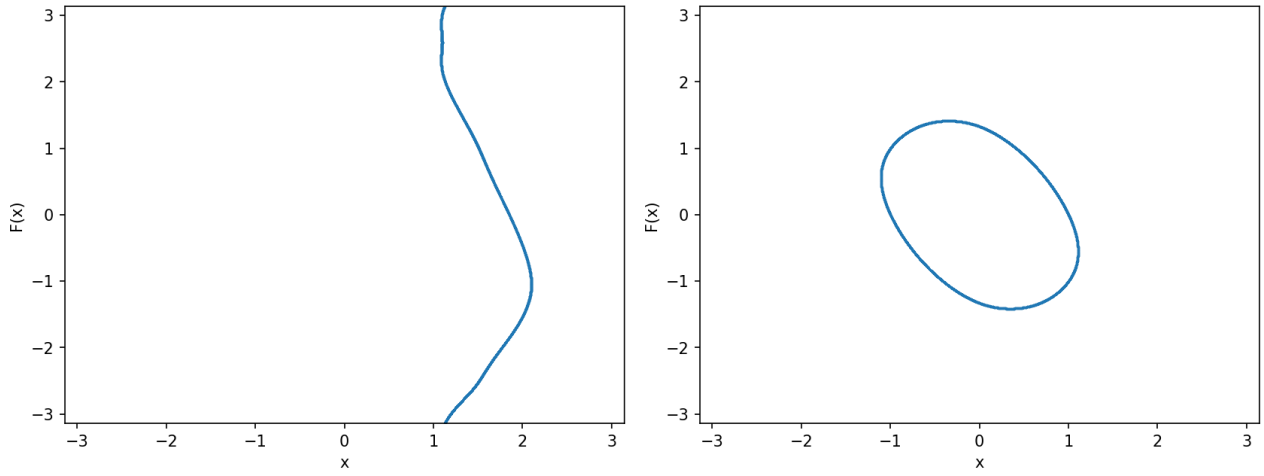

Figure 1.5: Plot of a vertical invariant curve, and a curve around the origin.

1.4 Reflections

There are many interesting insights gained from this function. Some of the more noteworthy are:

1. **Density Variations**: The varying density of points in different regions of the plot indicates

the nature of the dynamics. Sparse regions or gaps might hint at unstable regions or regions where trajectories tend to escape.

2. **Invariant Curves**: The presence of curves or regions with densely packed points suggests invariant curves. These curves represent quasi-periodic behavior in the system, where trajectories remain confined to these curves without diverging.
3. **Chaotic Regions**: The scattered and seemingly random distribution of points in certain areas of the plot indicates chaotic behavior. In these regions, trajectories do not settle down to a periodic orbit or an invariant curve.
4. **Impact of Parameter** a: Since $a = -0.7$ for this plot, the dynamics captured might be specific to this value. The standard map's behavior can vary significantly with changes in a. Later in this text, comparing this plot with others (for different a values) will reveal how this parameter influences the map's dynamics.
5. **Sensitive Dependence on Initial Conditions**: The wide variety of patterns seen in the plot for different initial conditions underscores the standard map's sensitive dependence on initial conditions. This is a hallmark of chaotic systems, where tiny differences in starting conditions can lead to vastly different outcomes.
6. **Central Dynamics**: The behavior around the origin (or center) of the plot might be especially interesting. If there are distinct patterns or invariant curves close to the origin, it could provide insights into the dynamics near this critical point.
7. **Overall Behavior**: Given the mix of chaotic regions and possible invariant curves, the dynamics for $a = -0.7$ seem to offer both regular and irregular behaviors.

To conclude this section, the seemingly simple function we have analyzed has given rise to great complexity - even for a single value of a.

1.5 Exploration of the variable a

In this task, we delve into the exploration of the standard map's dynamics by varying the parameter a, which plays an important role in dictating the system's behavior.

The parameter a is varied from -0.1 to -2.1, capturing a wide range of behaviors. The code and plot is given below

```
fig, ((ax1, ax2, ax3, ax4),
      (ax5, ax6, ax7, ax8),
      (ax9, ax10, ax11, ax12)) = plt.subplots(3, 4, figsize = (12,9),
                                              dpi=300)

figures = [ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8, ax9, ax10, ax11, ax12]
j = 0
N= 500

for x in figures:
    x.set_xlabel('x')
    x.set_ylabel('y')
    a = -0.1 - 2.1*j/(len(figures)-1)
    x.set_title(f"Map dynamics with a = {round(a, 2)}")

    x_data, y_data = NP_simulation(NP, N, a)
```

```
    for i in range(NP):
        x.plot(x_data[i*N:(i+1)*N], y_data[i*N:(i+1)*N], '.',
               markersize=1)
    j+=1


fig.tight_layout()
plt.show()
```

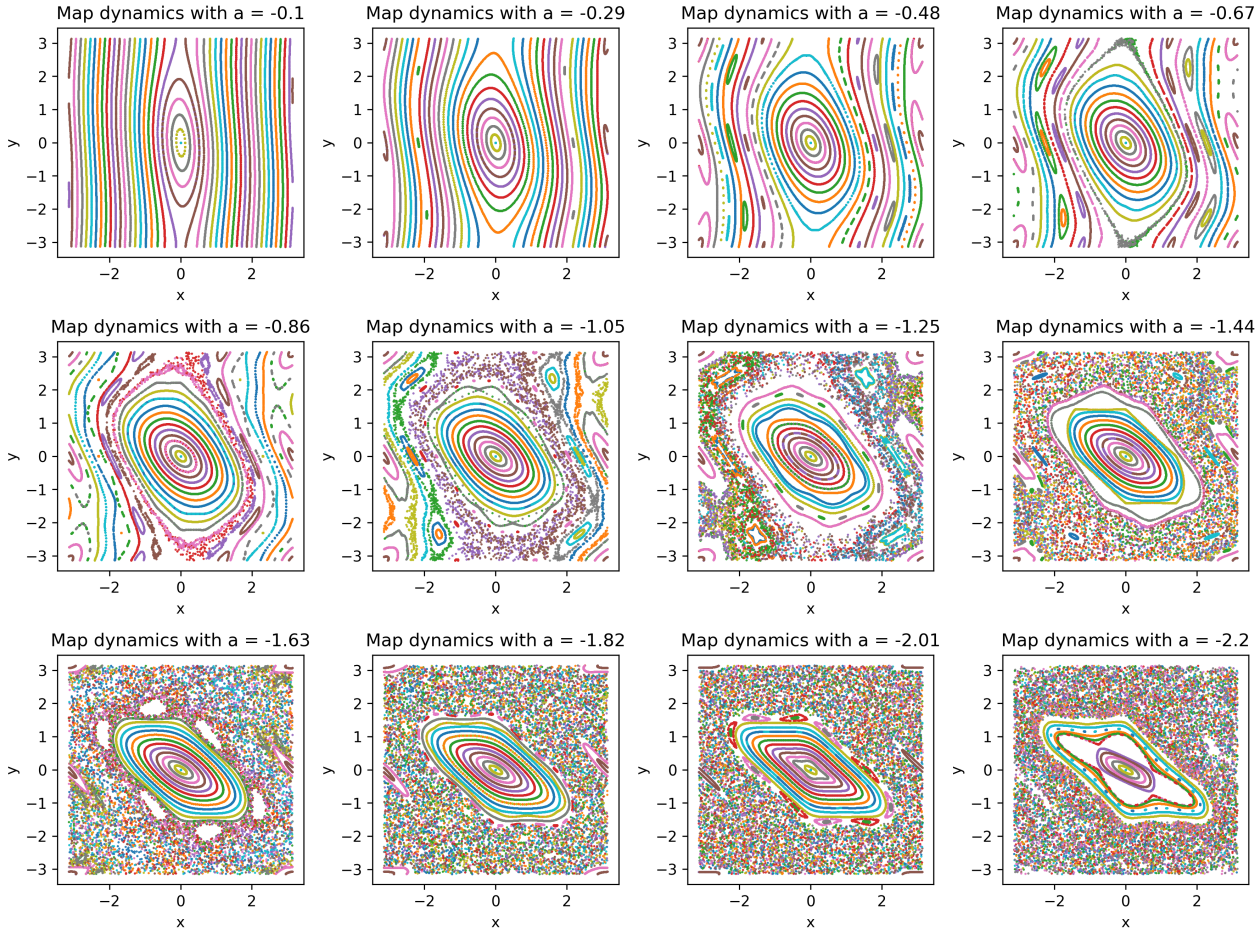

Figure 1.6: The figure shows the evolution of varying a.

1.6 Observations for specific values of a

For $a = -0.1$:

- The dynamics appear to be basically not chaotic at all, with trajectories possibly converging to only the origin in the phase space.
- There are clear invariant curves indicating quasi-periodic behavior.

For $a = -2.1$:

- The dynamics appear more chaotic and scattered, with an island of periodicity around the origin.
- The invariant curves evident for smaller values of a are possibly broken and less distinct, indicating increased chaotic behavior.

1.7 Evolution of dynamics when varying a

As a varies from -0.1 to -2.2:

1. **Initial Transition (from -0.1 to around -0.5 or -0.7)**: The dynamics might transition from relatively simple or quasi-periodic patterns to more intricate structures. The number of invariant curves decreases, and chaotic regions emerge.
2. **Mid-Range Values (from around -0.7 to -1.5)**: The plots show a mix of regular (invariant curves) and chaotic behaviors. Using the following term very loosely; there seems to be the most "information" in these plots. They are extremely complex without succumbing too much to "random", chaotic noise. As a increases in magnitude, the chaotic regions expand.
3. **Later Values (from -1.5 to -2.1)**: The dynamics become even more chaotic, with fewer clear patterns or invariant curves. The sensitivity to initial conditions increases for values further from the origin, leading to a more scattered distribution of trajectories in the phase space.

In summary, as a becomes more negative, the dynamics of the standard map seem to transition from quasi-periodic or regular behaviors to more chaotic ones. This gradual change underscores the complex interplay between regularity and chaos in the standard map and how sensitive it is to parameter variations.

2 Bonus

Here is some code to create a nice animation of the development of the plot varying a. The gif produced is in the e-mail.

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

fig, ax = plt.subplots()

frame_NO = 100
NP = 50

def update(i):
    ax.clear()
    a = -5 + 10*i/frame_NO
    x, y = NP_simulation(NP, N, a)
    for i in range(NP):
        plt.plot(x[i*N:(i+1)*N], y[i*N:(i+1)*N], '.', markersize=1)
    plt.xlabel('x')
    plt.ylabel('norm_norm_F(x)')
    plt.gcf().set_dpi(40)
    ax.set_title(f"Map dynamics with a = {a}")

ani = FuncAnimation(fig, update, frames=np.arange(0, frame_NO),
                    interval=frame_NO)
ani.save('animation.gif', writer='pillow', fps=20)
HTML(ani.to_jshtml())
```