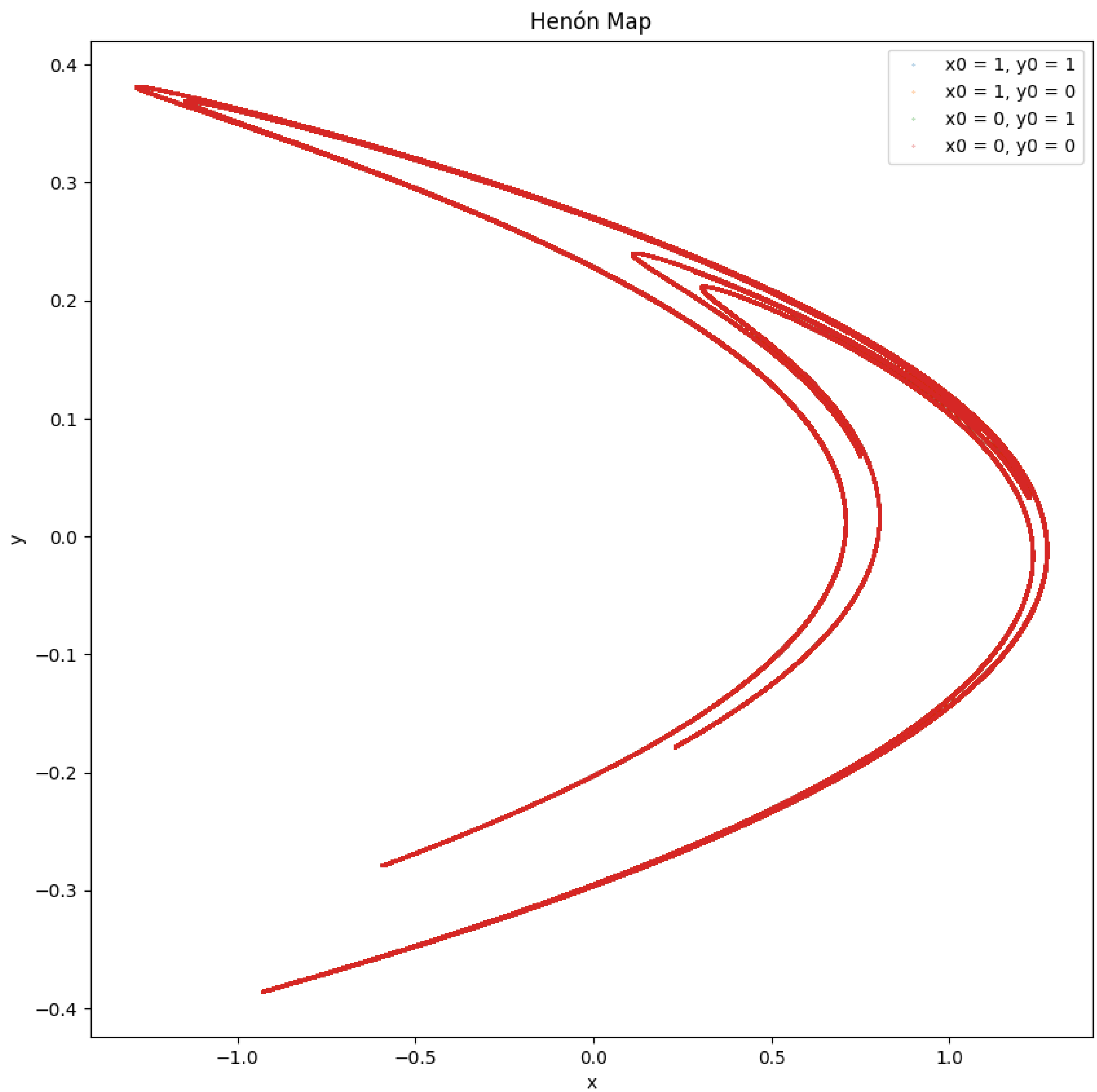
```
import numpy as np
import matplotlib.pyplot as plt
from tqdm.auto import tqdm
```

Part 1

```
a = 1.4
b = 0.3
iterations = 2000000
transient_iterations = 1000

def recursion_function(x,y,a,b):
    x_next = y + 1 - a*x**2
    y_next = b*x
    return x_next, y_next

def plot_Henon_map(x,y, x0,y0):
    plt.plot(x,y, '.', markersize=0.5)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Henón Map')

def run_recursion(x0,y0,a,b,iterations):
    x_list = np.zeros(iterations)
    y_list = np.zeros(iterations)
    x_list[0] = x0
    y_list[0] = y0
    for i in range(1,iterations):
        x_list[i], y_list[i] = recursion_function(x_list[i-
1],y_list[i-1],a,b)

    return x_list, y_list

initial_cond = [[1, 1], [1, 0], [0, 1], [0, 0]]
plt.figure(figsize=(10, 10))

for initial in initial_cond:
    x0 = initial[0]
    y0 = initial[1]
    x_list, y_list = run_recursion(x0, y0, a, b, transient_iterations)
    x_list, y_list = run_recursion(x_list[-1], y_list[-1], a, b,
iterations)
    plt.plot(x_list, y_list, '.', markersize=0.5, label=f'x0 = {x0},
y0 = {y0}')

plt.xlabel('x')
plt.ylabel('y')
plt.title('Henón Map')
```

```
plt.legend()  # Call legend outside the loop with the list of labels
plt.show()
```

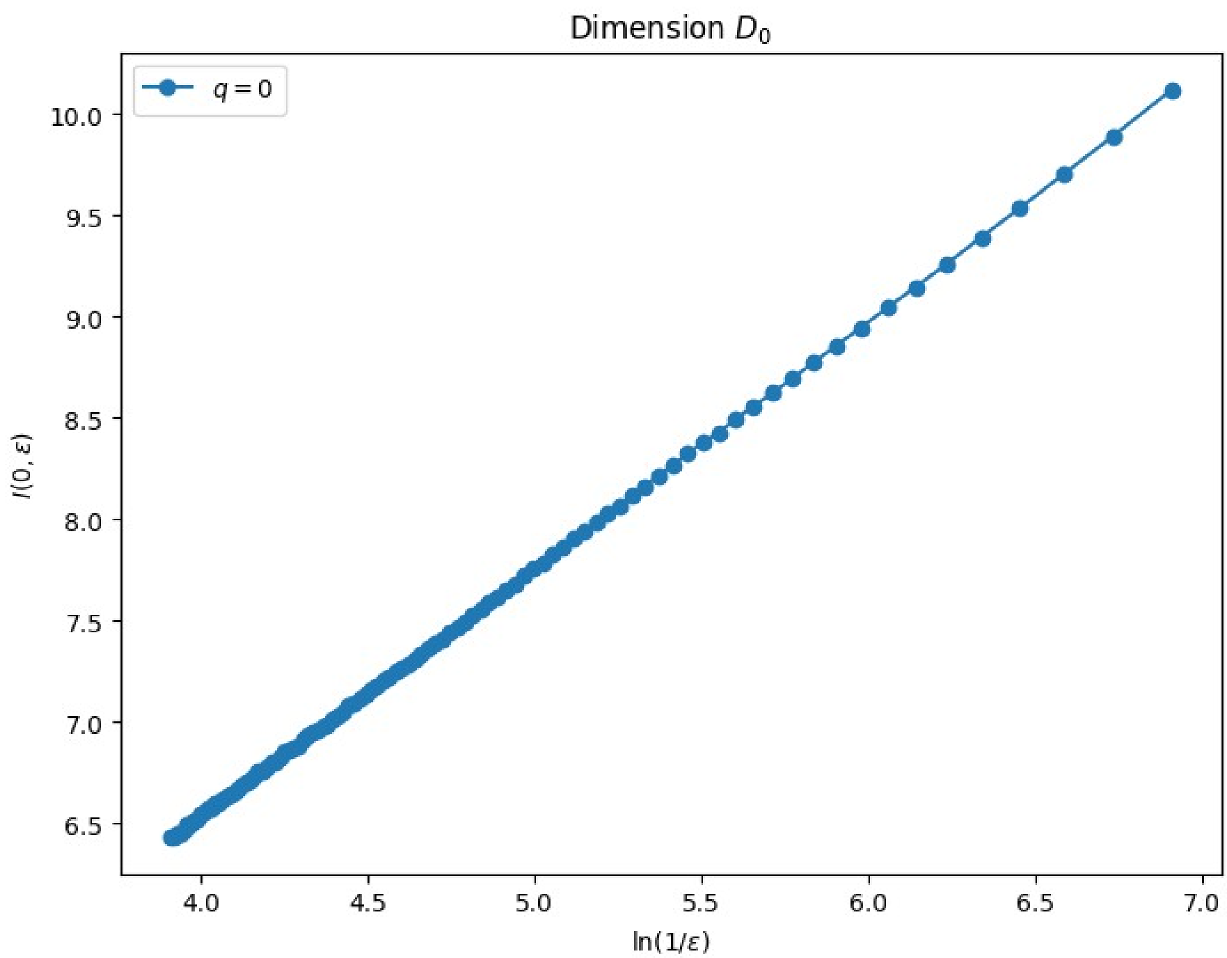
```
<ipython-input-38-dd46af0bd5aa>:2: RuntimeWarning: overflow
encountered in scalar power
  x_next = y + 1 - a*x**2
```

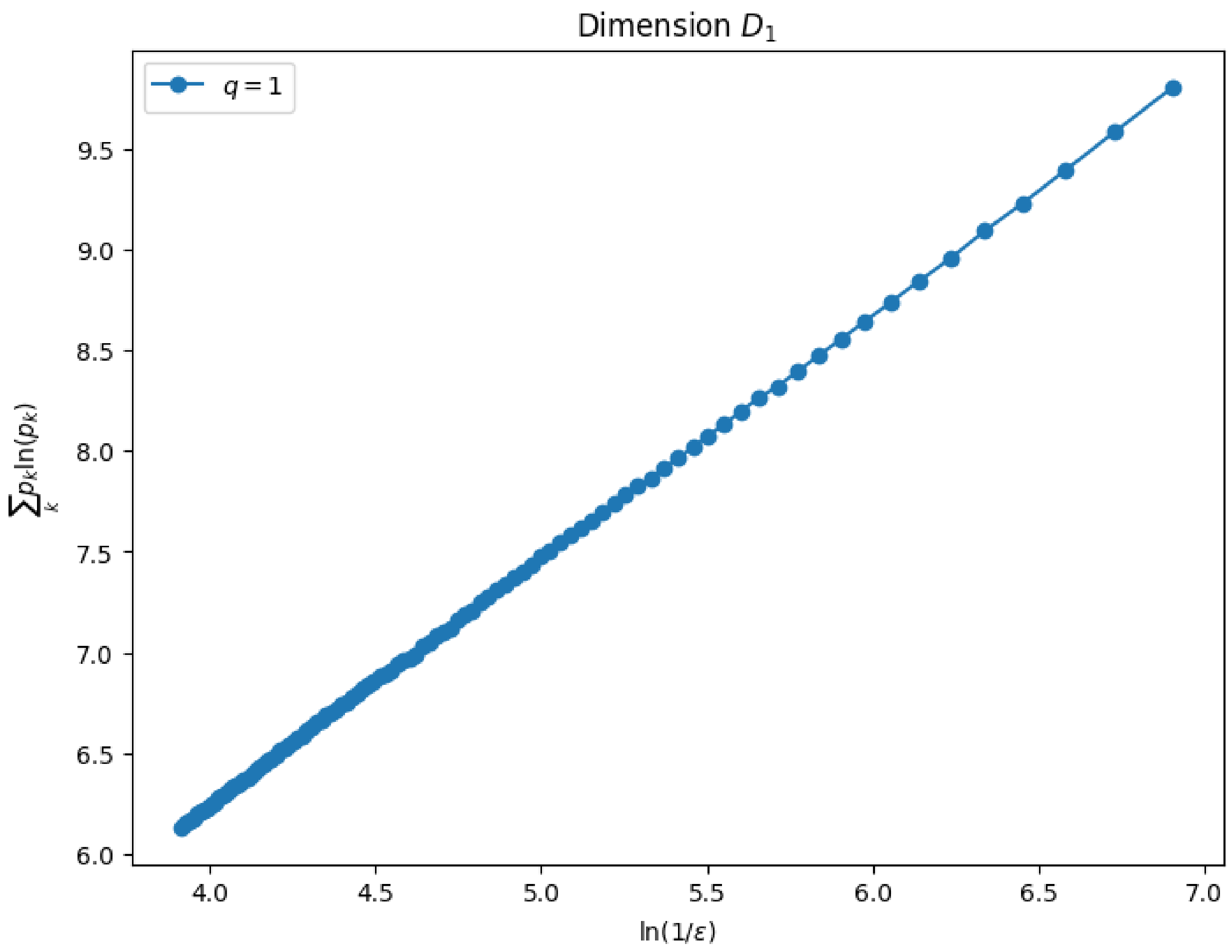


Part 2

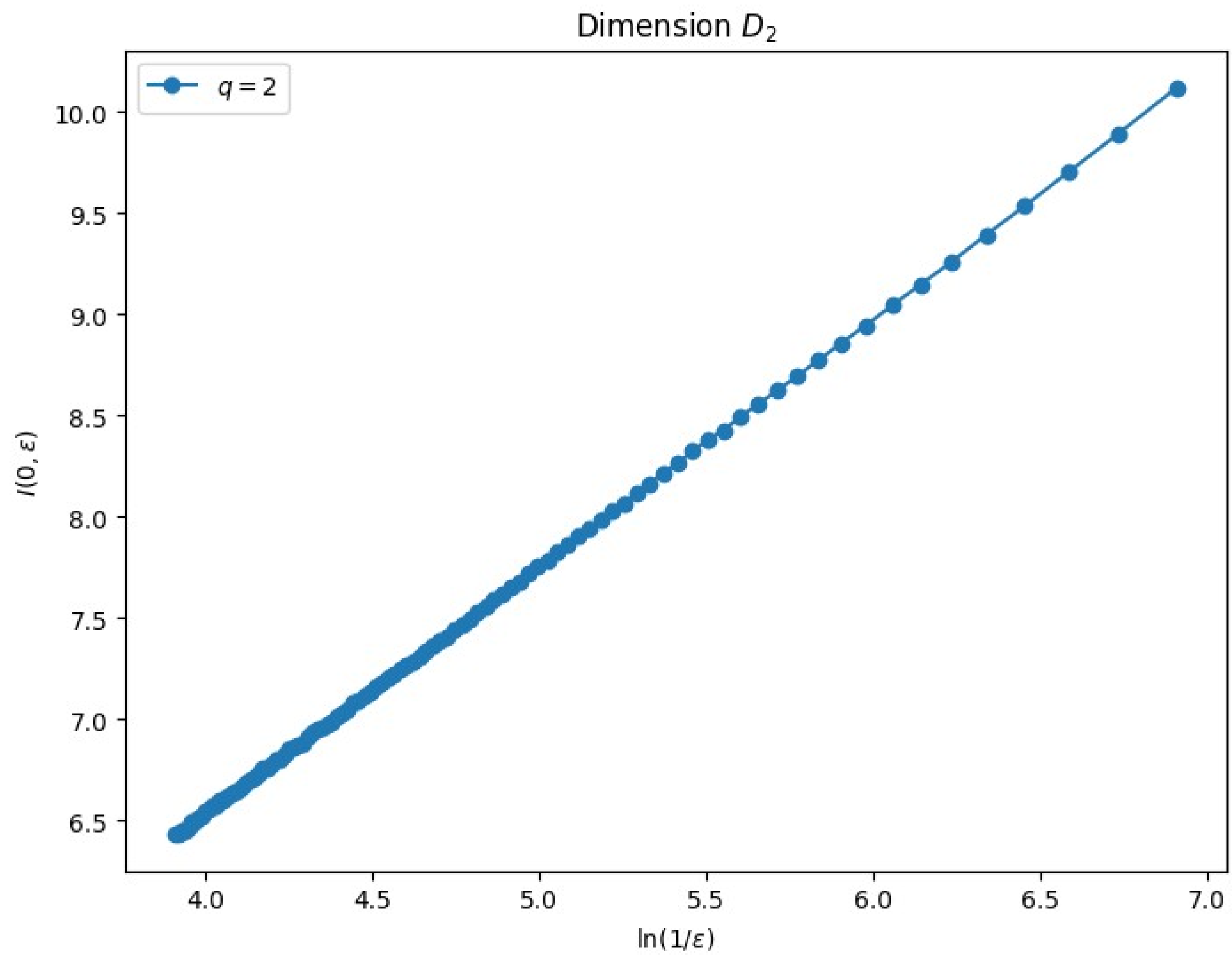
```
epsillon_list = np.linspace(1e-3, 2e-2, 100)
ln_inv_eps = np.log(1 / epsillon_list)

def get_limits(x_list,y_list):
    return np.min(x_list), np.max(x_list), np.min(y_list),
np.max(y_list)

x_list, y_list = run_recursion(0, 0, a, b, transient_iterations)
x_list, y_list = run_recursion(x_list[-1], y_list[-1], a, b,
iterations)
x_min, x_max, y_min, y_max = get_limits(x_list,y_list)

x_min -= 0.01
x_max += 0.01
y_min -= 0.01
y_max += 0.01

ln_I = np.zeros((len(epsillon_list),2))
D1_values = np.zeros((len(epsillon_list),1))
for i, epsillon in enumerate(epsillon_list):

    nx = int(np.ceil((x_max - x_min) / epsillon))
    ny = int(np.ceil((y_max - y_min) / epsillon))

    counts, _, _ = np.histogram2d(x_list, y_list, bins=[nx, ny],
                                  range=[[x_min,x_max],[y_min, y_max]])
    counts = counts.flatten()
    N_total = counts.sum()

    p = counts / N_total
    p_nonzero = p[p > 0]
    for q in [0,1,2]:
        if q == 0:
            val = np.log(len(p_nonzero))
            ln_I[i,0] = val
        if q == 1:
            sum_p_logp = np.sum(p_nonzero * np.log(1/p_nonzero))
            D1_values[i,0] = sum_p_logp

        if q == 2:
            sum_pq = np.sum(p_nonzero**q)
            val = (1/(1-q))*np.log(sum_pq)
            ln_I[i,1] = val

plt.figure(figsize=(8,6))
plt.plot(ln_inv_eps, ln_I[:,0], 'o-', label='$q=0$')
plt.xlabel(r'$\ln(1/\epsilon)$')
plt.ylabel(r'$I(0, \epsilon)$')
```

```python
plt.title(r'Dimension $D_0$')
plt.legend()
plt.show()

plt.figure(figsize=(8,6))
plt.plot(ln_inv_eps, D1_values[:,0], 'o-', label='$q=1$')
plt.xlabel(r'$\ln(1/\epsilon)$')
plt.ylabel(r'$\sum_k p_k \ln(p_k)$')
plt.title(r'Dimension $D_1$')
plt.legend()
plt.show()

plt.figure(figsize=(8,6))
plt.plot(ln_inv_eps, ln_I[:,0], 'o-', label='$q=2$')
plt.xlabel(r'$\ln(1/\epsilon)$')
plt.ylabel(r'$I(0, \epsilon)$')
plt.title(r'Dimension $D_2$')
plt.legend()
plt.show()
```





Dimension D_1
$q = 1$
$\sum_k p_k \ln(p_k)$
$\ln(1/\varepsilon)$
9.5
9.0
8.5
8.0
7.5
7.0
6.5
6.0
4.0
4.5
5.0
5.5
6.0
6.5
7.0




```python
from scipy.stats import linregress

D0_estimate, _,_,_,_ = linregress(ln_inv_eps, ln_I[:,0])

D1_estimate, _,_,_,_ = linregress(ln_inv_eps, D1_values[:,0])

D2_estimate, _,_,_,_ = linregress(ln_inv_eps, ln_I[:,1])

print(f"D0 estimate: {D0_estimate:.2f}")
print(f"D1 estimate: {D1_estimate:.2f}")
print(f"D2 estimate: {D2_estimate:.2f}")
```

```
D0 estimate: 1.22
D1 estimate: 1.22
D2 estimate: 1.18
```

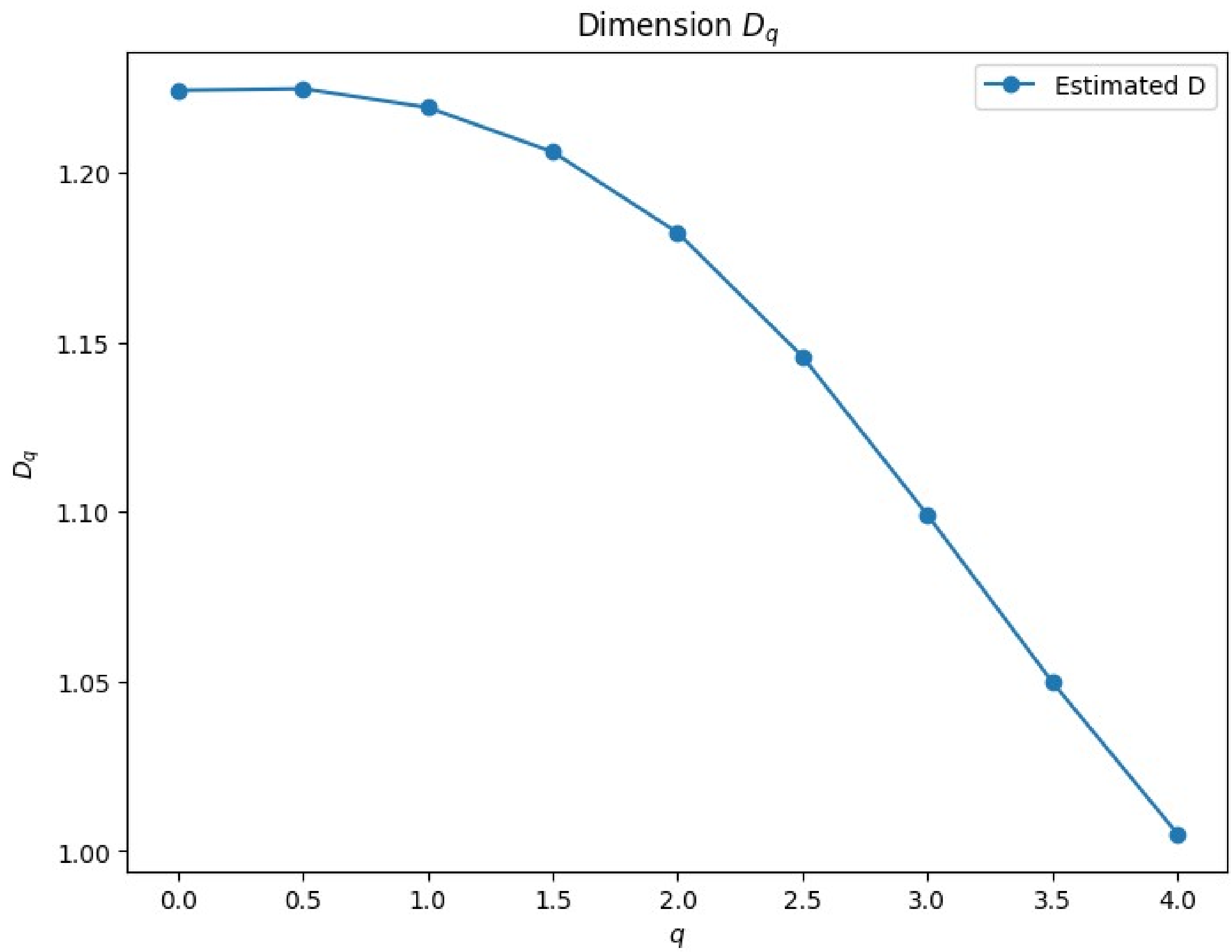
part d

```
epsillon_list = np.linspace(1e-3, 2e-2, 100)
ln_inv_eps = np.log(1 / epsillon_list)

x_list, y_list = run_recursion(0, 0, a, b, transient_iterations)
x_list, y_list = run_recursion(x_list[-1], y_list[-1], a, b,
iterations)
x_min, x_max, y_min, y_max = get_limits(x_list,y_list)

x_min -= 0.01
x_max += 0.01
y_min -= 0.01
y_max += 0.01

q_list = np.linspace(0, 4, 9)
Dq_values = np.zeros((len(q_list),len(epsillon_list)))

for i, epsillon in tqdm(enumerate(epsillon_list)):

    nx = int(np.ceil((x_max - x_min) / epsillon))
    ny = int(np.ceil((y_max - y_min) / epsillon))

    counts, _, _ = np.histogram2d(x_list, y_list, bins=[nx, ny],
                                range=[[x_min,x_max],[y_min, y_max]])
    counts = counts.flatten()
    N_total = counts.sum()

    p = counts / N_total
    p_nonzero = p[p > 0]

    for j, q in enumerate(q_list):
        if q == 0:
            val = np.log(len(p_nonzero))
            Dq_values[j,i] = val
        if q == 1:
            sum_p_logp = np.sum(p_nonzero * np.log(1/p_nonzero))
            Dq_values[j,i] = sum_p_logp
        else:
            sum_pq = np.sum(p_nonzero**q)
            val = (1/(1-q))*np.log(sum_pq)
            Dq_values[j,i] = val
```

{"model_id":"f6f5b540bc764947ade022016b558ee7","version_major":2,"version_minor":0}

```
D_estimate = np.zeros(len(q_list))
for i, q in enumerate(q_list):
    D_estimate[i] = linregress(ln_inv_eps, Dq_values[i,:])[0]
```

```
plt.figure(figsize=(8,6))
plt.plot(q_list, D_estimate, 'o-', label='Estimated D')
plt.xlabel(r'$q$')
plt.ylabel(r'$D_q$')
plt.title(r'Dimension $D_q$')
plt.legend()
plt.show()
```



Part e

```
def jacobian(x,a,b):
    dF1dx = -2*x*a
    dF1dy = 1

    dF2dx = b
    dF2dy = 0
    return np.array([[dF1dx,dF1dy],[dF2dx,dF2dy]])
```

```python
M = np.array([[1, 0], [0, 1]])
exponents_list = np.zeros((iterations,2))

for i in tqdm(range(iterations)):
    x = x_list[i]
    J = jacobian(x,a,b)

    M = np.matmul(J,M)
    Q, R = np.linalg.qr(M)
    eigenvalues = np.diag(R)
    exponents_list[i,:] = np.log(np.abs(eigenvalues))
    M = Q

stability_exponents = np.mean(exponents_list, axis=0)
stability_exponents = np.sort(stability_exponents)[::-1]

print("Computed Stability Exponents:")
print(fr"$\lambda_1$ = {stability_exponents[0]:.2f}, $\lambda_1$ =
{stability_exponents[1]:.2f}")
```

{"model_id":"1d2f359716fd4191a4119d54308055e9","version_major":2,"version_minor":0}

```
Computed Stability Exponents:
$\lambda_1$ = 0.42, $\lambda_1$ = -1.62
```

```python
1 + stability_exponents[0]/np.abs(stability_exponents[1])
```

```
1.2583692640896018
```