Exercise_13

December 9, 2025

0.0.1 Course: BIO-341 *Dynamical systems in biology*

Professor: *Julian Shillcock & Felix Naef*

SSV, BA5, 2025

Note that this document is primarily aimed at being consulted as a Jupyter notebook, the PDF rendering being not optimal.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact
from sklearn.linear_model import LinearRegression
```

1 1.0 Fractal dimension of the Koch curve (only this question is a model exam question)

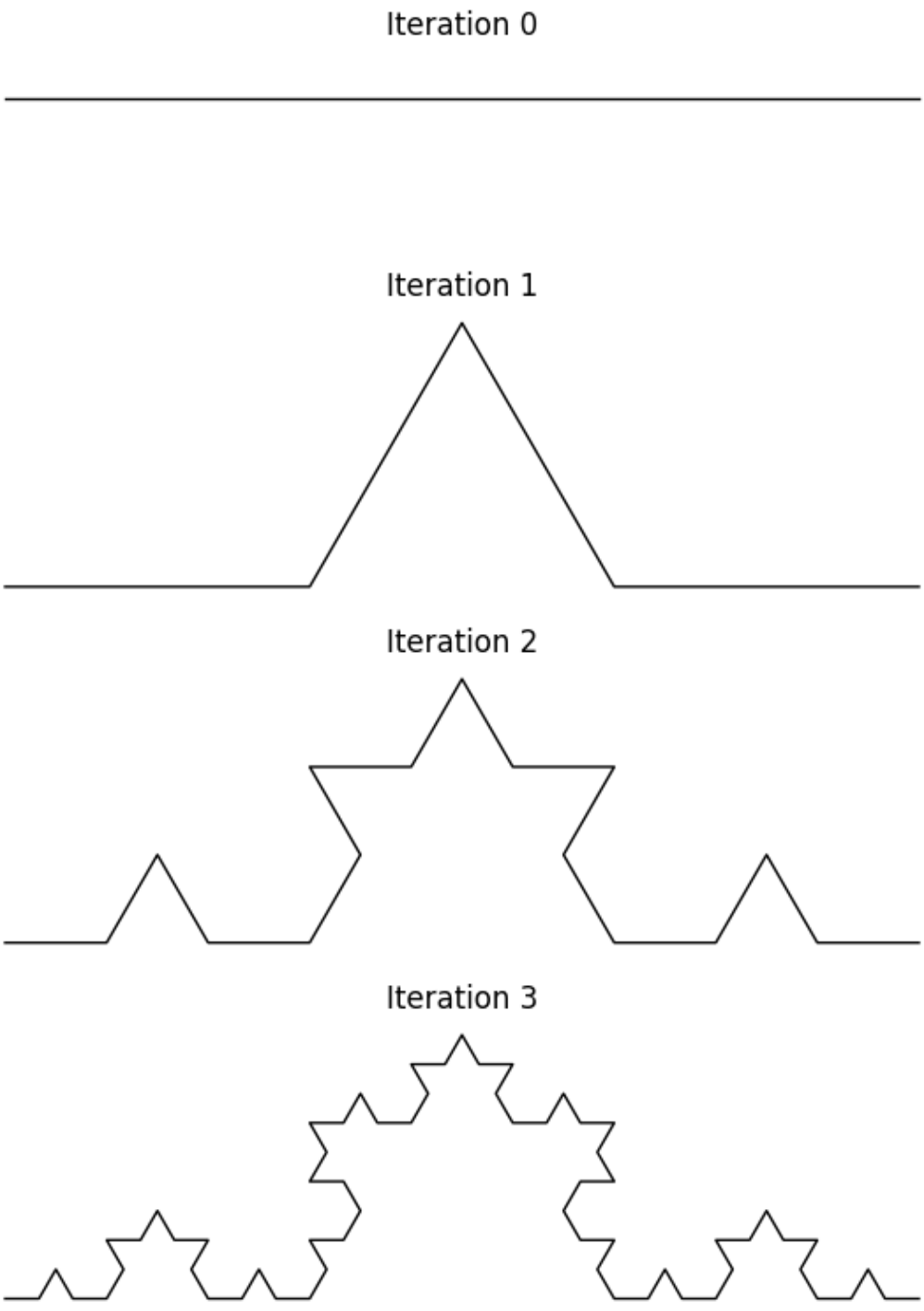
The Koch curve is a continuous, non-differentiable curve defined in the unit interval (0,1). It is constructed using the following steps:

1) Draw a unit interval
2) Remove the middle third segment, and replace it by two sides of a triangle with side length 1/3.
3) For each straight segment of the curve, remove the middle third, and replace it by the two sides of a triangle whose lengths are equal to one third of the length of the original segment.
4) Repeat step 3 infinitely many times.

Using the box counting method of calculating the Fractal Dimension find the dimension of the Koch curve (see figure below):

[10]:
```
import numpy as np
import matplotlib.pyplot as plt

def koch_iter(points):
    new_points = []
    for i in range(len(points) - 1):
        p1 = points[i]
        p2 = points[i+1]
        v = p2 - p1
        s = p1 + v/3.0
```

```
        t = p1 + 2.0*v/3.0
        # rotate by +60 degrees around s to get the peak
        rot = np.array([[np.cos(np.pi/3), -np.sin(np.pi/3)],
                        [np.sin(np.pi/3),  np.cos(np.pi/3)]])
        peak = s + rot.dot((t - s))
        new_points += [p1, s, peak, t]
    new_points.append(points[-1])
    return np.array(new_points)

def koch_curve(n):
    pts = np.array([[0.0, 0.0], [1.0, 0.0]])
    for _ in range(n):
        pts = koch_iter(pts)
    return pts

fig, axes = plt.subplots(4, 1, figsize=(6, 8))
for idx, n in enumerate([0, 1, 2, 3]):
    pts = koch_curve(n)
    axes[idx].plot(pts[:,0], pts[:,1], color="black", linewidth=1)
    axes[idx].set_title(f"Iteration {n}")
    axes[idx].set_aspect('equal', adjustable='box')
    axes[idx].axis('off')
plt.tight_layout()
plt.show()
```



1.0.1 Box-counting dimension of the Koch curve

The box-counting dimension of a set $F \subset \mathbb{R}^2$ is defined as

$$d = \lim_{\varepsilon \to 0} \frac{\ln N(\varepsilon)}{\ln(1/\varepsilon)},$$

where $N(\varepsilon)$ is the minimal number of squares of side length ε needed to cover F.

2 Ex 2

The sine map is defined by the iteration scheme:

$$X_{n+1} = r \sin(\pi x_n) \tag{1}$$

For a constant parameter $ 0 < r < 1 $, and an initial value that satisfies $ 0 < x_0 < 1 $.

Write a Python code that iterates the above map, given the parameters: $ r $, the number of iterations $ N $, and an initial point $ x_0 $. The code should either write the iterates to a file for later plotting, or plot them directly. Use 2000 iterations and discard the first 1000 points for plotting or calculations to remove the initial transients in the map.

In order to find the fixed point we want to find the point x_{FP} such that:

$$x_{FP} = rsin(\pi x_{FP})$$

One solution of the equation is obviously 0, for the other ones (if any), we need to use numerical approaches

[2]:
```python
from scipy.optimize import fsolve
import numpy as np

def sin_map(x, r):
    '''
    Sinusoidal map
    '''
    return r * np.sin(np.pi * x)

def sin_map_prime(x, r):
    '''
    Derivative of the sinusoidal map
    '''
    return r * np.pi * np.cos(np.pi * x)

def equation(x, r):
    return sin_map(x,r) - x

def find_fixed_point(r):
    initial_guess = 0.7
    return fsolve(equation, initial_guess, args=(r,))
```

2.1 exercise 2.1

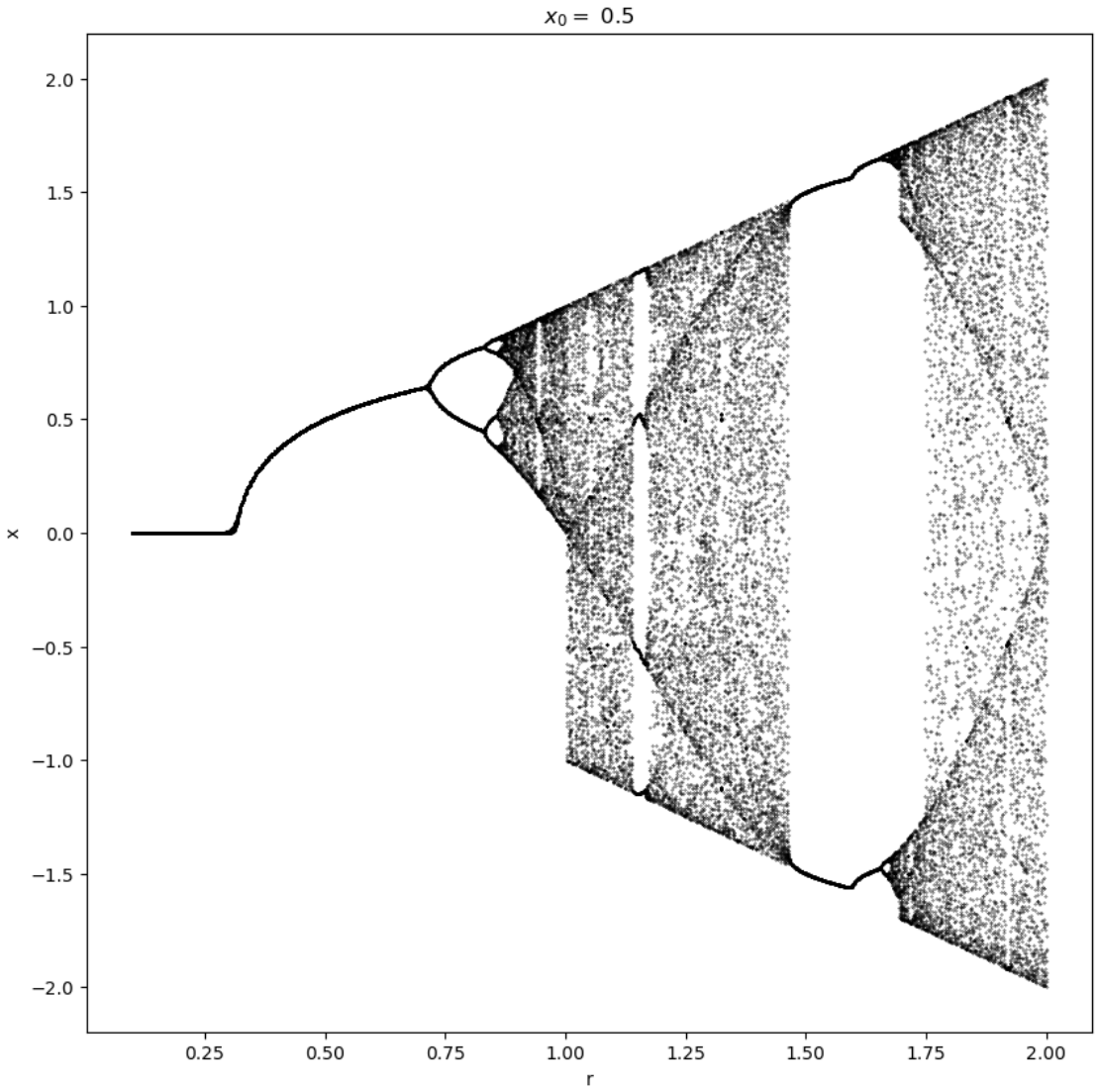
Wrap the iteration function in a loop over a user-defined range of r values, and plot all the iterates for each r value in the same graph. It should start to look like the figure below, i.e., for each value of r, all the iterates produced from the initial value x0 should be plotted vertically. If you want, you can change the value of x0 for each value of r, but because the map is "chaotic", it doesn't really matter. NB. Don't use x0 = 0, or 1!

[3]:
```
def trajectory(r, N, x0):
    '''
    Compute the trajectory of the sinusoidal map
    '''
    x = np.zeros(N)
    x[0] = x0
    for i in range(1, N):
        x[i] = sin_map(x[i-1], r)
    return x
```

For this exercise we just need to run the trajectory function for different values of r.
Notice how we run the system for 100 iterations, but than we plot only the last 50: we basically make sure that the system has time to evolve from x0 to the asymptotic solution.

[54]:
```
x0 = 0.5
n_last = 50

plt.figure(figsize=(10, 10))
for r in np.linspace(0.1, 2.0, 1000):

    x = np.ones(n_last)*r
    tr = trajectory(r, 100, x0)
    y = tr[-n_last:]

    plt.scatter(x, y, s=0.1, c = 'black')

plt.title('$x_0 = $ ' + str(x0))
plt.xlabel('r')
plt.ylabel('x')
```

[54]:
```
Text(0, 0.5, 'x')
```

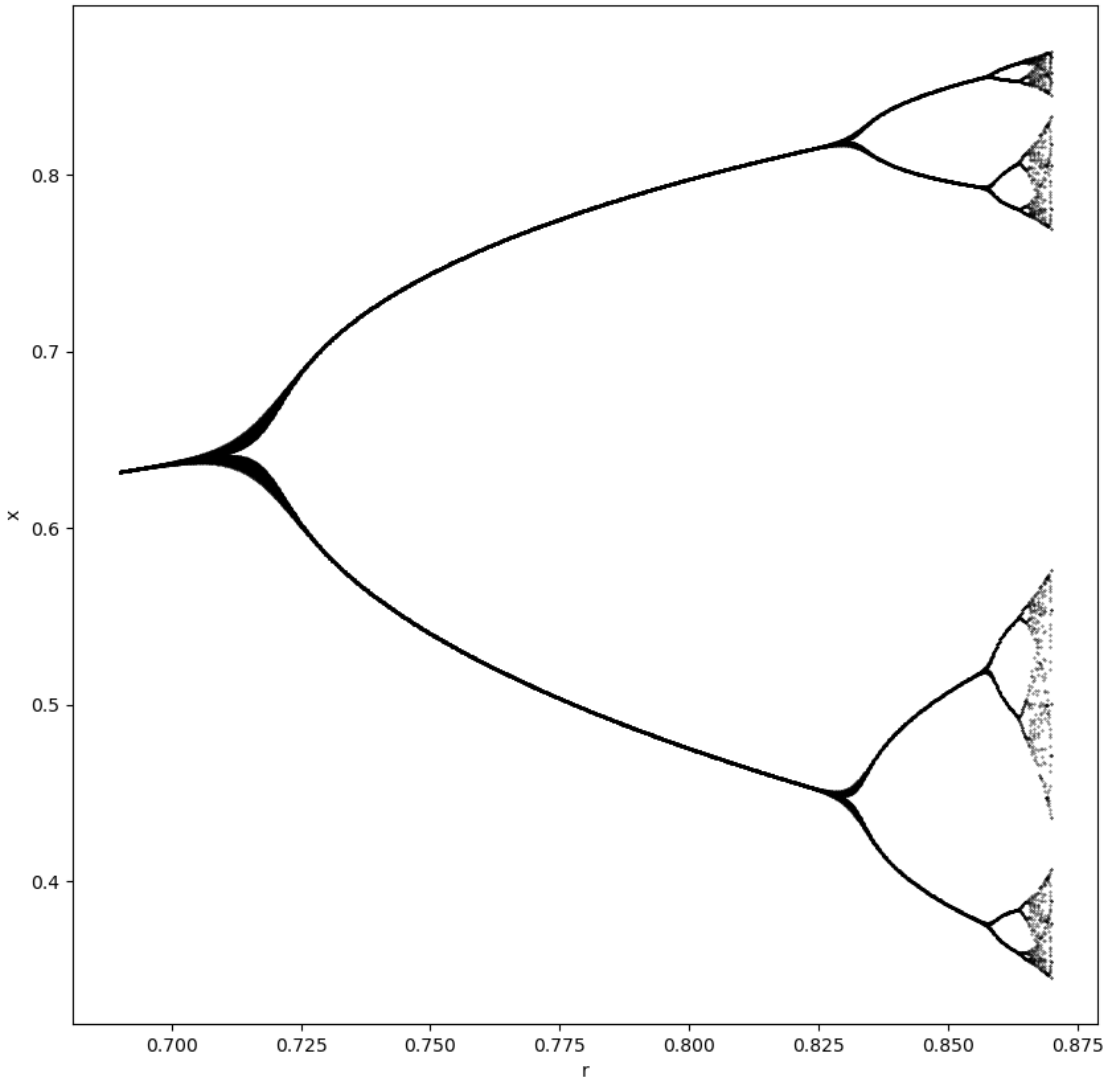


Zooming in to a more narrow range of r

[ ]:
```
n_last = 50

plt.figure(figsize=(10, 10))

xx = []
yy = []

r_span = np.linspace(0.69, 0.87, 1000)

for r in r_span:
```

```
    x = np.ones(n_last)*r
    xx.append(x)

    tr = trajectory(r, 100, x0)
    y = tr[-n_last:]
    yy.append(y)
    plt.scatter(x, y, s=0.1, c = 'black')

# plt.xlim(0.856, 0.875)
plt.xlabel('r')
plt.ylabel('x')
```

```
Text(0, 0.5, 'x')
```



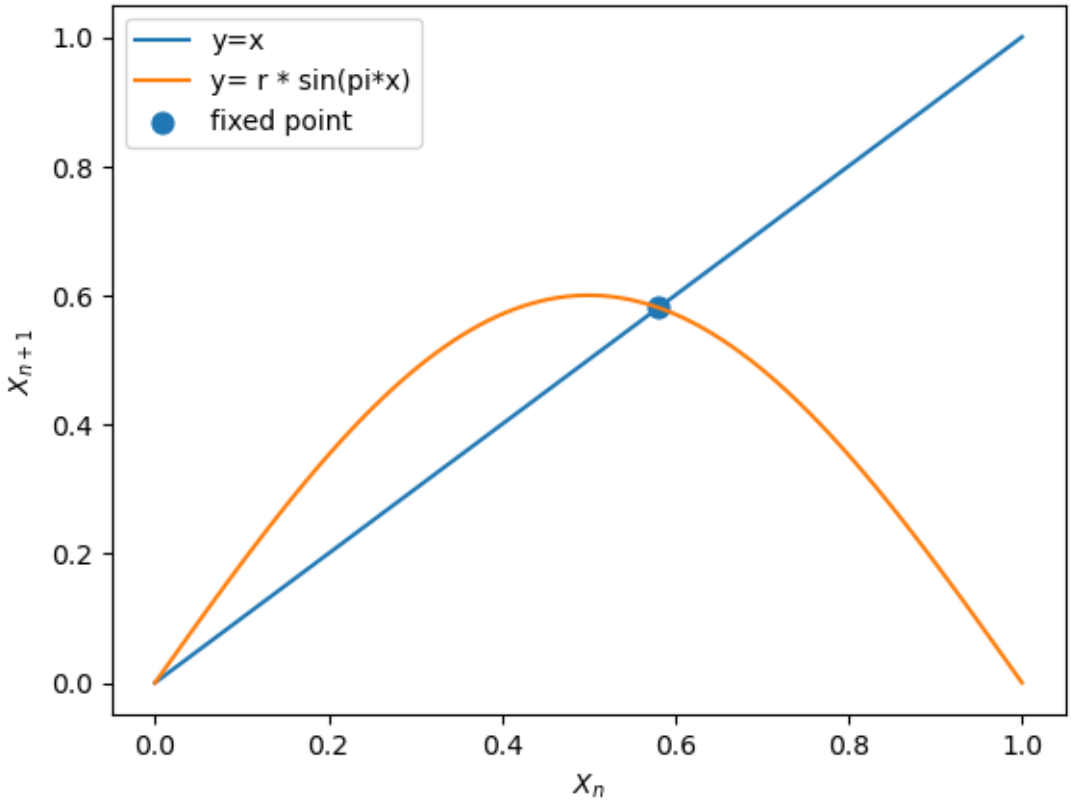
2.2 Exercise 2.2

1.2 For r = 0.6, iterate the map for at least 5 initial values randomly chosen between (0, 1). What do you observe? Then repeat for r = 0.72 and 0.75. What do you observe in your plot of xn against increasing r values?

2.2.1 r = 0.6

[4]:
```
x = np.linspace(0, 1, 100)
r = 0.6
solution = find_fixed_point(r)
plt.plot(x, x, label='y=x')
plt.plot(x, sin_map(x , r), label='y= r * sin(pi*x)')
plt.scatter(solution, solution, s=60 , label='fixed point')

plt.xlabel('$X_n$')
plt.ylabel('$X_{n+1}$')

plt.legend()
```

[4]:
```
<matplotlib.legend.Legend at 0x143bfe700>
```



Let's study the stability of the FP

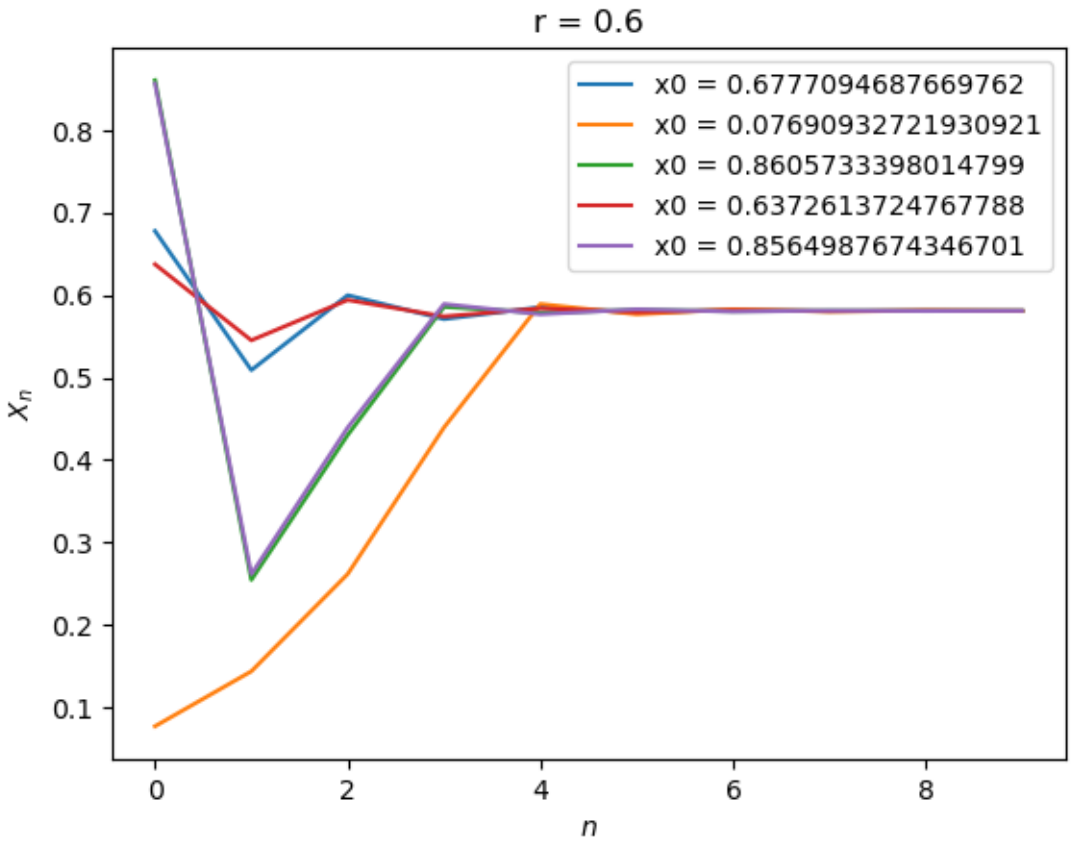
[5]:
```
print(sin_map_prime(0.,r) ,sin_map_prime(solution, r))
```

```
1.8849555921538759 [-0.47325162]
```

therefore zero is unstable (the abs value of the multiplier is larger than 1) while the other FP is stable

[6]:
```
# randomly choose 5 values of x0 beween 0 and 1
x0 = np.random.random(5)
trajs = []

for s in x0:
    trajs.append( trajectory(r, 10, s))

for traj in trajs:
    plt.plot(traj, label = 'x0 = ' + str(traj[0]))
plt.legend()
plt.title('r = ' + str(r))
plt.xlabel('$n$')
plt.ylabel('$X_{n}$')
```

[6]:
```
Text(0, 0.5, '$X_{n}$')
```



All trajectories converge to value x_final

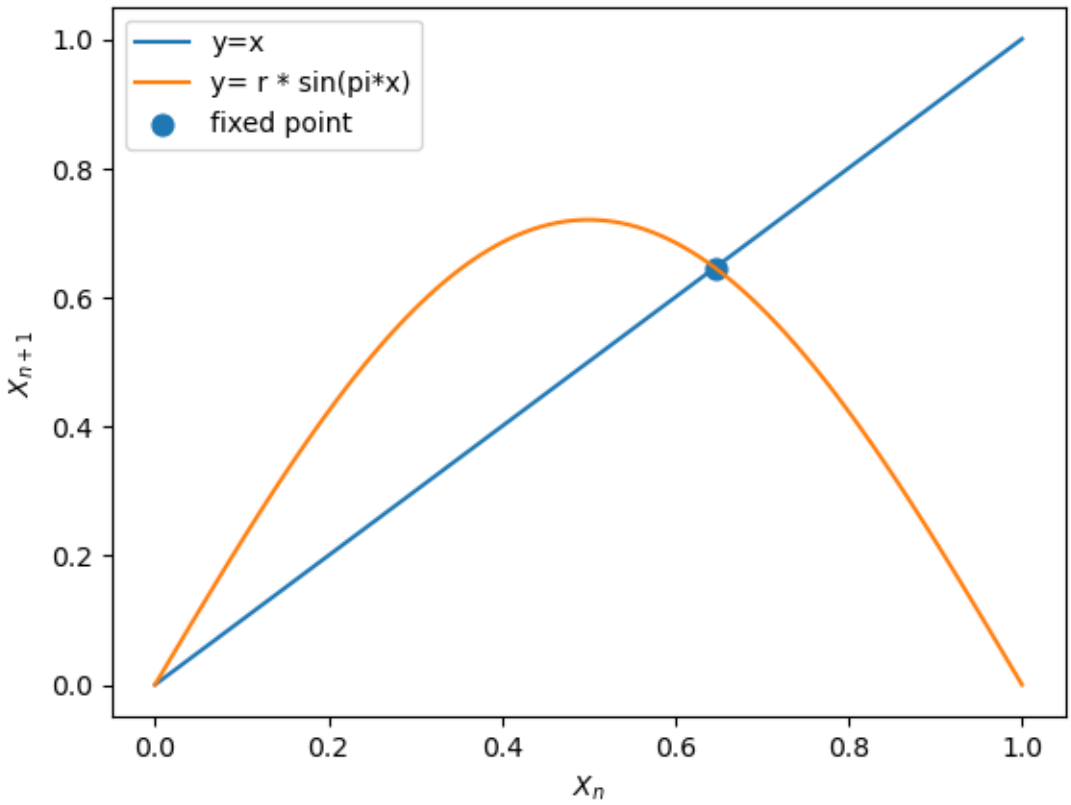
[7]:
```
print(trajs[0][-1])
```

```
0.580674158924557
```

2.2.2 r = 0.72

[8]:
```
x = np.linspace(0, 1, 100)
r2 = 0.72
plt.plot(x, x, label='y=x')
plt.plot(x, sin_map(x , r2), label='y= r * sin(pi*x)')
solution2 = find_fixed_point(r2)
plt.scatter(solution2, solution2, s=60 , label='fixed point')

plt.xlabel('$X_n$')
plt.ylabel('$X_{n+1}$')

plt.legend()
```

[8]:
```
<matplotlib.legend.Legend at 0x143f70b50>
```



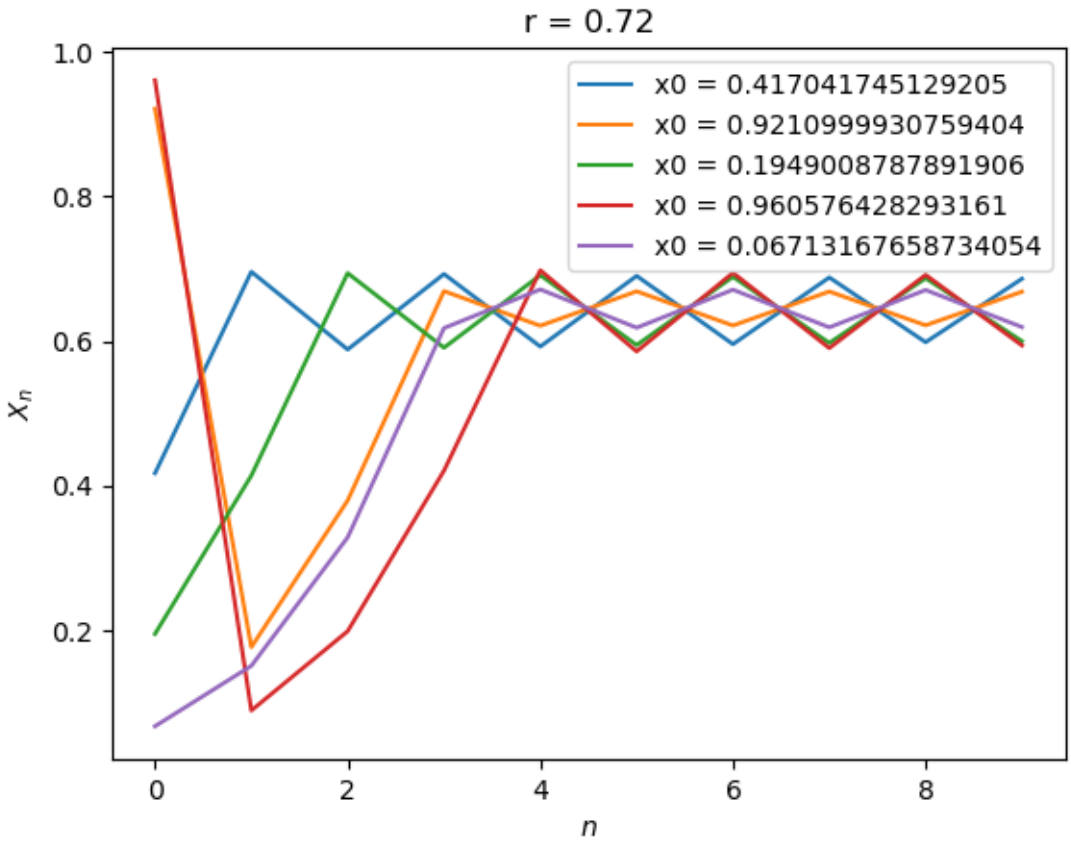
Both the fixed points are unstable!

[9]:
```
print(sin_map_prime(0.,r2) ,sin_map_prime(solution2, r2))
```

```
2.261946710584651 [-1.00016274]
```

[10]:
```
x0 = np.random.random(5)
trajs = []

for s in x0:
    trajs.append( trajectory(r2, 10, s))

for traj in trajs:
    plt.plot(traj, label = 'x0 = ' + str(traj[0]))
plt.legend()
plt.title('r = ' + str(r2))

plt.xlabel('$n$')
plt.ylabel('$X_{n}$')
```

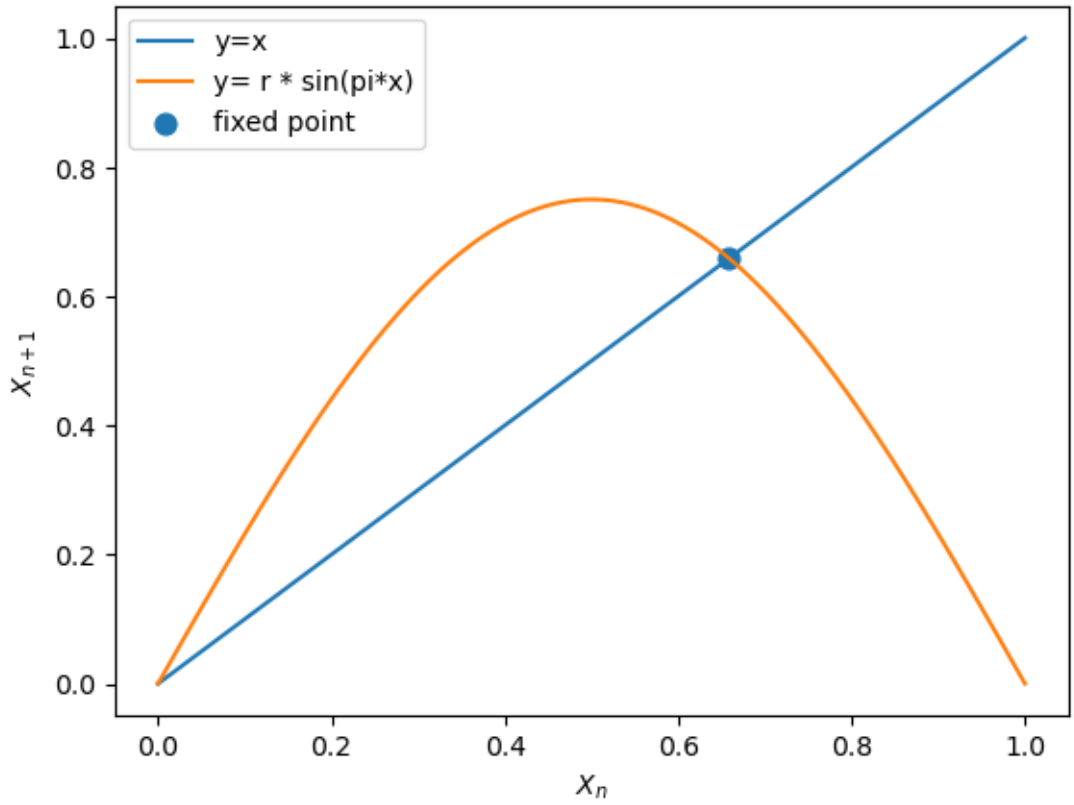
[10]:
```
Text(0, 0.5, '$X_{n}$')
```



2.2.3 r = 0.75

[11]:
```
x = np.linspace(0, 1, 100)
r3 = 0.75
plt.plot(x, x, label='y=x')
plt.plot(x, sin_map(x , r3), label='y= r * sin(pi*x)')
solution3 = find_fixed_point(r3)
plt.scatter(solution3, solution3, s=60 , label='fixed point')

plt.xlabel('$X_n$')
plt.ylabel('$X_{n+1}$')

plt.legend()
```

[11]:
```
<matplotlib.legend.Legend at 0x143f43f10>
```



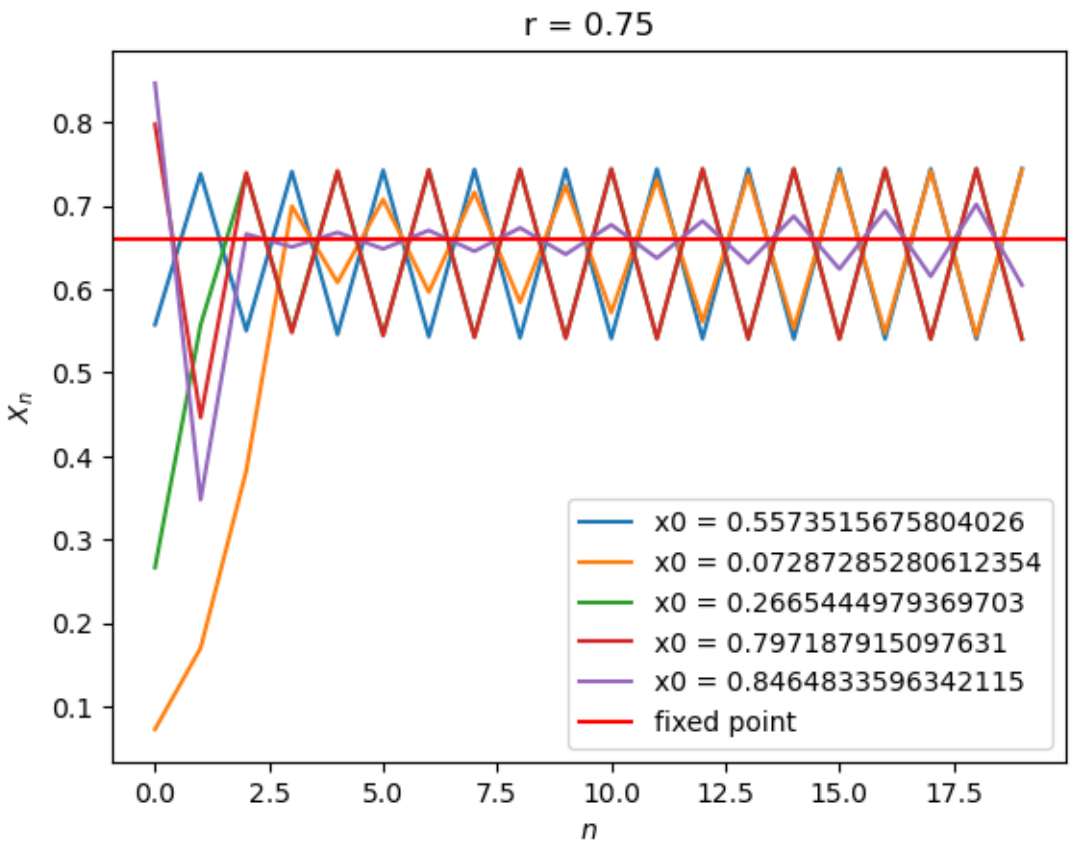
Both FP are unstable!

[12]:
```
print(sin_map_prime(0.,r3) ,sin_map_prime(solution2, r3))
```

```
2.356194490192345 [-1.04183619]
```

[13]:
```
x0 = np.random.random(5)
trajs = []

for s in x0:
    trajs.append( trajectory(r3, 20, s))

for traj in trajs:
    plt.plot(traj[:], label = 'x0 = ' + str(traj[0]))
#plot orizontal line
plt.axhline(y=solution3, color='r', linestyle='-', label='fixed point')

plt.legend()
plt.title('r = ' + str(r3))

plt.xlabel('$n$')
plt.ylabel('$X_{n}$')
```

[13]:
```
Text(0, 0.5, '$X_{n}$')
```



[14]:
```
# taking the 2 asymptotic values
v1 = trajs[0][-1]
v2 = trajs[0][-2]
```

2.2.4 Understanding 2-periodic solutions with second-iterate maps

We can understand this period 2 solution looking at the second-iterate map: this is a map that instead of sending to the future x_n by one time unit, it sends it by 2 time units:

basically the original map f reads:

$$x_{n+1} = f(x_n)$$

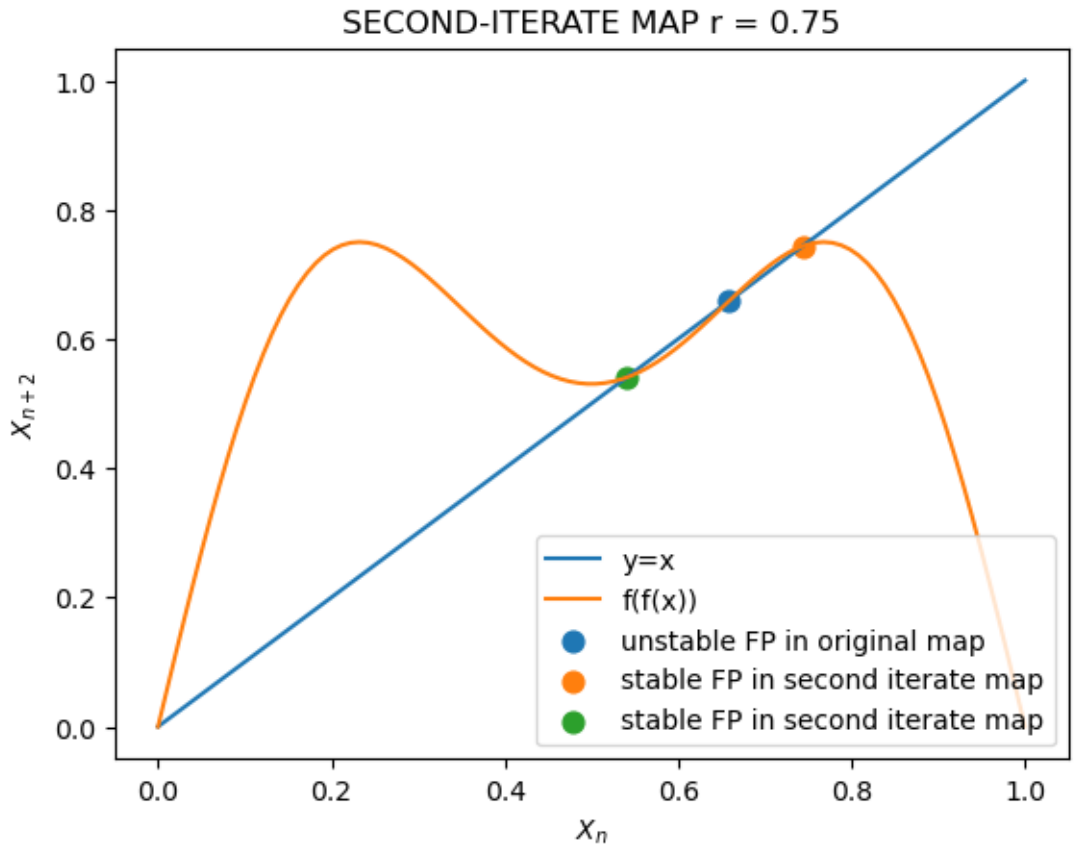
while if we re-applying the map we get the second-iterate map g:

$$x_{n+2} = f(x_{n+1}) = f(f(x_n)) = g(x_n)$$

Studying this map, we have 3 fixed point, where the one we found earlier is unstable, while the other two are stable (from the plot we can see that the derivative at the intersection with y = x is close to zero)

[15]:
```
# plot second degree map

x = np.linspace(0, 1, 100)
r3 = 0.75
plt.plot(x, x, label='y=x')
plt.plot(x, sin_map(sin_map(x , r3), r3), label='f(f(x))')
solution3 = find_fixed_point(r3)
plt.scatter(solution3, solution3, s=60 , label='unstable FP in original map')
plt.scatter(v1, v1, s=60 , label='stable FP in second iterate map')
plt.scatter(v2, v2, s=60 , label='stable FP in second iterate map')

plt.title('SECOND-ITERATE MAP r = ' + str(r3))

plt.xlabel('$X_n$')
plt.ylabel('$X_{n+2}$')

plt.legend()
```

[15]:
```
<matplotlib.legend.Legend at 0x144dd9670>
```



2.3 Ex. 2.3

Keep increasing r, and plot the iterates produced. At what r value does "chaos" set in for the sine map? i.e., when does a (nearly) infinite number of fixed points appears (cp. Logistic map shown in the figure below, for which lambda ~ 3.6 is the transition to chaos. Note that lambda in the logistic map plays the role of r for the sine map.) Do you observe any "stability windows" in which the number of fixed points is small? (cp. Logistic map for lambda ~ 3.82, where only 3 fixed points appear.)

Chaos emerges aroud r = 0.86

theoretical values of $r_{\inf} = r_0 + \Delta_0 \Sigma_{n=0} \frac{1}{\delta^n}$

There are stability windows around r = 1.1 and 1.6

2.4 Interactive plots

[18]:
```
# now plot the map for with a number of times


def plot_map(r):
```

```
    x = np.linspace(-1, 1, 1000)

    plt.figure(figsize=(15, 5))
    plt.suptitle('maps')

    plt.subplot(1, 2, 1)
    plt.plot(x, x, label='y=x')
    plt.plot(x, sin_map(x , r), label='f(x)')
    plt.title('number of FP changes with r = ' + str(r))
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(x, x, label='y=x')
    plt.plot(x, sin_map(sin_map(x , r),r), label='f(f(x))')
    plt.title('SECOND-ITERATE MAP r = ' + str(r))
    plt.legend()


interact(plot_map, r=(0.1, 1.0, 0.01))
```

```
interactive(children=(FloatSlider(value=0.55, description='r', max=1.0, min=0.1,␣
 ↪step=0.01), Output()), _dom_c…
```

[18]:
```
<function __main__.plot_map(r)>
```

Here we plots two rajectories, with very similar but not idenitcal initial conditions: on the onset of chaos this trajectories start to diverge fro each other

[19]:
```
trajectory(r, 10, s)
x0 = 0.5
x02 = 0.51

def plot_map(r):


    plt.figure(figsize=(15, 5))
    plt.suptitle('maps')
    tr = trajectory(r, 1000, x0)
    tr2 = trajectory(r, 1000, x02)
    # plt.subplot(1, 2, 1)

    plt.plot(tr[-20:], label='x0 = ' + str(tr[0]))
    plt.plot(tr[-20:], 'o')


    plt.plot(tr2[-20:], label='x0 = ' + str(tr2[0]))
    plt.plot(tr2[-20:], 'o')
```

```
    plt.xlabel('n')
    plt.ylabel('$X_n$')
    plt.legend()



interact(plot_map, r=(0.1, 2.0, 0.01))
```

```
interactive(children=(FloatSlider(value=1.05, description='r', max=2.0, min=0.1,␣
 ↪step=0.01), Output()), _dom_c…
```

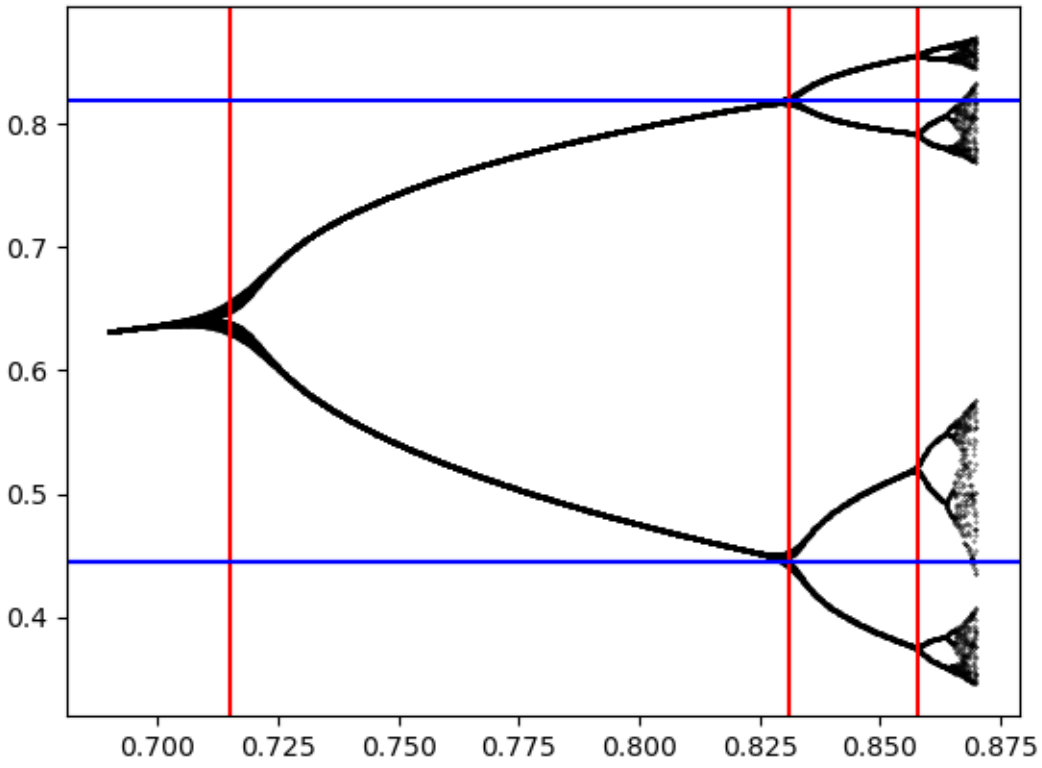
[19]:
```
<function __main__.plot_map(r)>
```

3 Ex 3

Extend your code to output the iterates from the sine map for a sequence of r values between 0.1 and 1, taking at least 20 values, and randomly setting the initial point for each one. Plot the fixed point(s) on the Y axis against the r value on the X axis, so you get a plot similar to the one above. Then repeat this accurately enough that you can estimate the successive r values at which the number of fixed points doubles (you may need more than 20 values of r, and you may need to zoom in on small portions of the r-axis to get sufficient accuracy. So you'll need to examine many little graphs across the r axis to locate the values of r where the period doubling occurs.

[21]:
```
for i,r in enumerate(r_span):
    plt.scatter(xx[i], yy[i], s =0.1 , c = 'black')

# vertical lineb

b1= 0.715
b2= 0.831
b3= 0.858

plt.axvline(x=b1, color='r', linestyle='-')
plt.axvline(x=b2, color='r', linestyle='-')
plt.axvline(x=b3, color='r', linestyle='-')

# horizontal line
plt.axhline(y=0.82, color='r', linestyle='-', c = 'blue')
plt.axhline(y=0.445, color='r', linestyle='-', c = 'blue')

plt.axhline(y=0.445, color='r', linestyle='-', c = 'blue')
```

[21]:
```
<matplotlib.lines.Line2D at 0x14a0c3370>
```



Estimating Feigenbaum number

[22]:
```
delta = (0.831 - 0.715) / (0.858 - 0.831)
delta
```

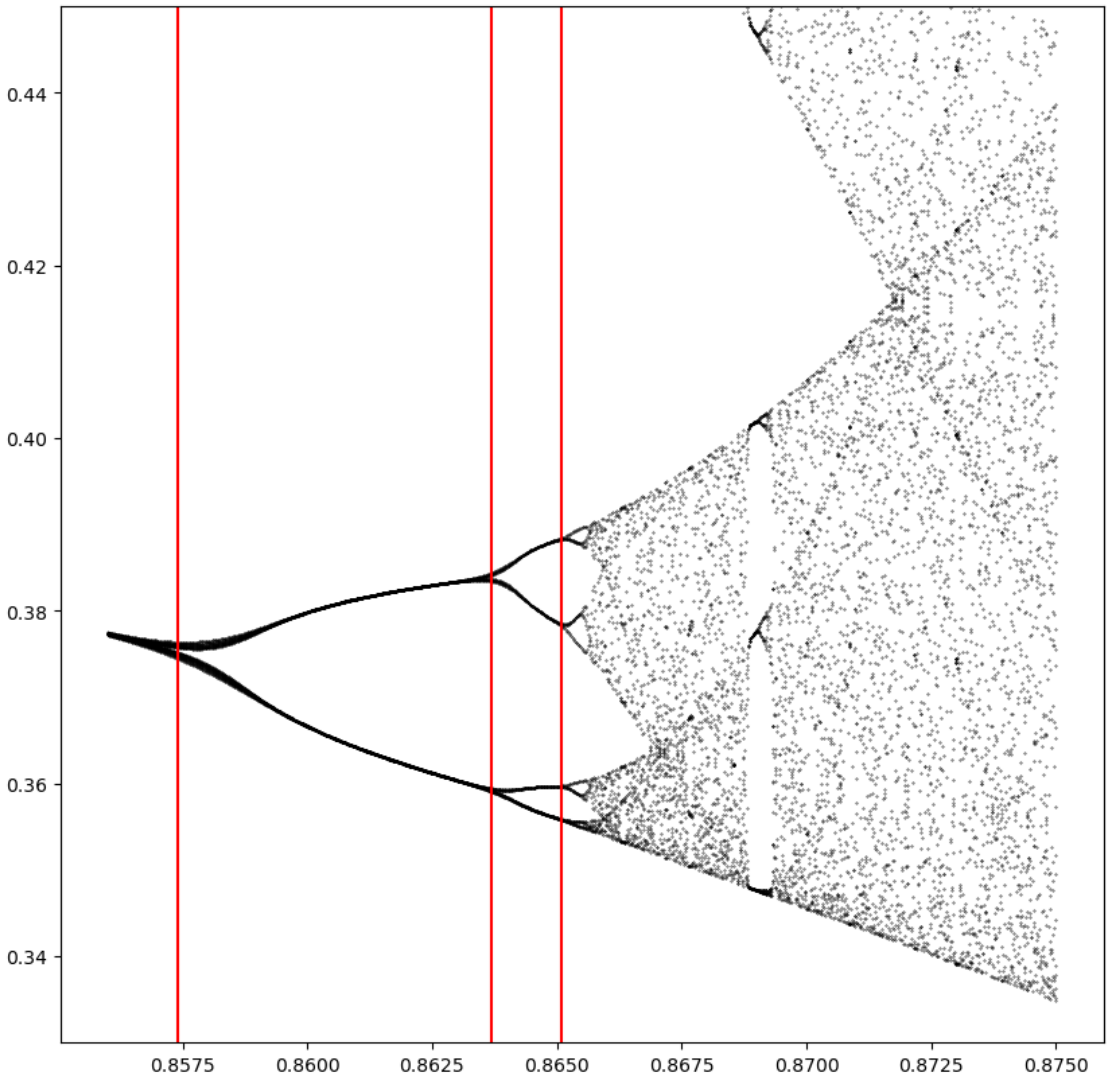
[22]:
```
4.296296296296292
```

Zooming in mre to get a more accurate esitamtion of the F. numbers

[23]:
```
plt.figure(figsize=(10, 10))

for r in np.linspace(0.856, 0.875, 1000):

    x = np.ones(n_last)*r
    tr = trajectory(r, 100, x0)
    y = tr[-n_last:]

    plt.scatter(x, y, s=0.1, c = 'black')

# vertical line
plt.axvline(x=0.8574, color='r', linestyle='-')
plt.axvline(x=0.8637, color='r', linestyle='-')
plt.axvline(x=0.8651, color='r', linestyle='-')
```

```
vv1 = 0.8574
vv2 = 0.8637
vv3 = 0.8651

plt.ylim(0.33, 0.45)
```

[23]: (0.33, 0.45)



Zooming in, we get a bit closer to the real values of $\delta = 0.6692$

[95]:
```
delta_true = 0.6692
delta = (vv2 - vv1) / (vv3 - vv2)
delta
```

[95]: 4.500000000000119

Theretical value for r_{inf}

[97]:
```
diff = vv2 - vv1
r_inf = vv1 + diff * 1 / (1 - delta_true)
r_inf
```

[97]: 0.8764447400241837

3.1 Ex 3.2

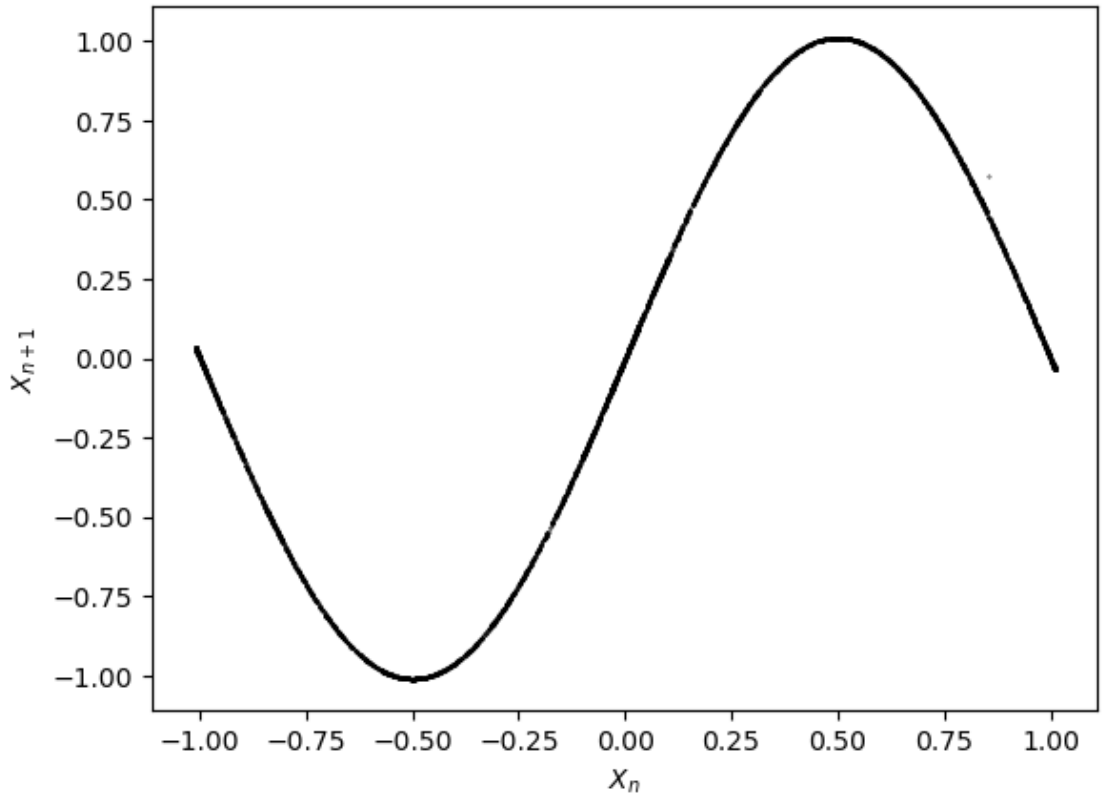
4 Ex 4: Fractal dimension of the sine map

4.1 Ex 4.1

Find a value of r such that you get a lot of fixed points (i.e., you are in a region for the sine map corresponding to the region r ~ 3.95 for the logistic map). Generate 10,000 points from a randomly-chosen initial point x0, and write them to a file. Discard the first 5,000. Duplicate the data into a second column shifting each value by one. So, for each row in the file, column one contains xn, and column two contains xn+1. Plot xn+1 against xn.

[ ]:
```
x0 = 0.5
r_chaos = 1.01
# r_chaos = 0.75

x0 = 0.5
x0_2 = 0.5001
tr = trajectory(r_chaos, 10000, x0)


# discard first half of the trajectory
tr = tr[5000:]
tr_shifted = np.roll(tr, -1)

plt.scatter(tr, tr_shifted, s= 0.1, c = 'black')
# plt.scatter(tr2, tr2_shifted, s= 0.1, c = 'red')

plt.xlabel('$X_n$')
plt.ylabel('$X_{n+1}$')
# plt.plot(tr, tr_shifted)
```

[ ]: Text(0, 0.5, 'X_{n+1}')



4.2 Ex 4.2

For the 50,000 points from section 3.1, and for a series of values of $\epsilon = 1, 0.5, 0.25, \frac{1}{2^n}$ ..., create a histogram of the number of points within a distance ϵ of x_i for a sequence of values of x_i ranging from 0 to 1. Then average the number of points in all the bins for each x_i value, to get an average $\langle N(\epsilon) \rangle$ for the number of points that lie in bins of size ϵ across the set of points. Repeat this for each ϵ.

Plot $\ln(\langle N(\epsilon) \rangle)$ against $\ln(\epsilon)$ and measure the slope to obtain the "fractal dimension" of the fixed points of the sine map for a single r value.

What is the distribution of fixed points for this value of r?

[60]:
```
# seq = np.linspace(0, r_chaos, 100)
seq = tr
eps = [1/2**n for n in range(1, 15)]

# now i want a function that counts the number of point of trajectory that are␣
 ↪within eps radius from each point of seq

def count_points(tr, eps):
    '''
```

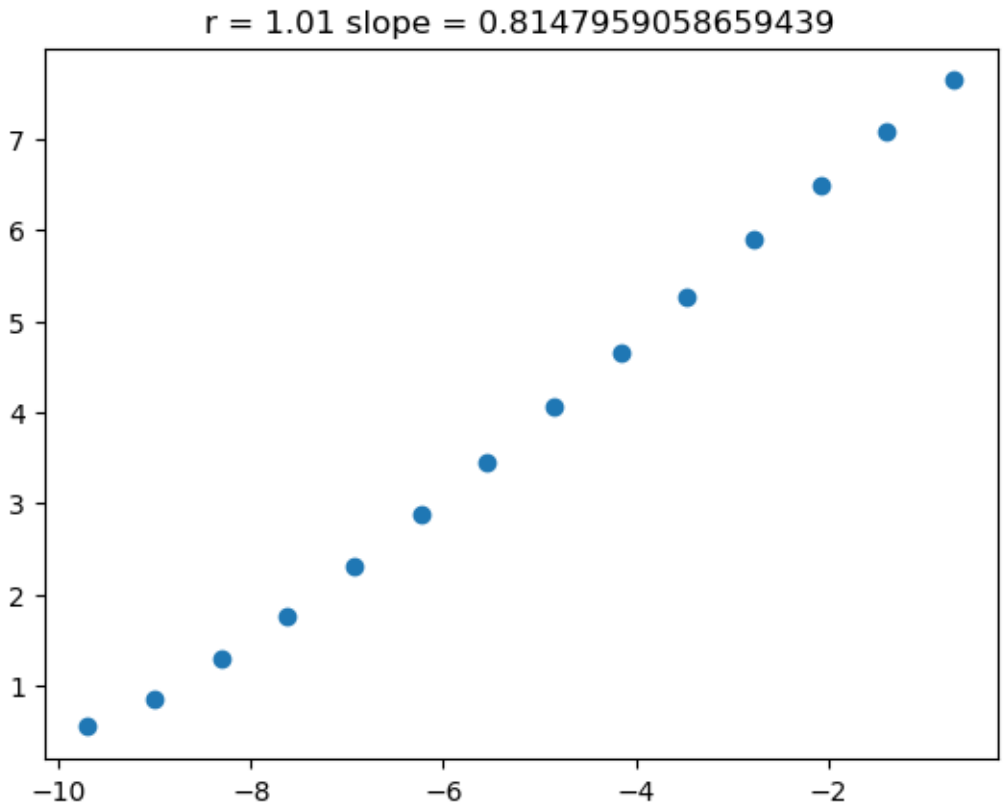
```
    Count the number of points of tr that are within eps radius from each point␣
 ↪of tr
    '''
    # compute the distance between each point of tr, getting a matrix
    dist = np.abs(tr[:, None] - tr[None, :])
    # count how many points are within eps radius from each point of seq
    count = np.sum(dist < eps, axis=0)

    return count.mean()

count_points(tr, eps[13])
N_eps = [count_points(tr, eps[i]) for i in range(len(eps))]
N_eps
```

[60]:
```
[2077.764,
 1188.142,
 662.8468,
 361.1936,
 192.8592,
 105.1792,
 57.4848,
 31.7644,
 17.818,
 10.0636,
 5.8544,
 3.6228,
 2.3732,
 1.7312]
```

[46]:
```
# linear regression

reg = LinearRegression().fit(np.log(eps).reshape(-1, 1), np.log(N_eps).
 ↪reshape(-1, 1))
# plot slope and intercept
# reg.coef_, reg.intercept_
slope = reg.coef_[0][0]
print('slope = ', reg.coef_[0][0])

plt.plot(np.log(eps), np.log(N_eps), 'o')
plt.title('r = ' + str(r_chaos) + ' slope = ' + str(slope))
```

```
slope =  0.8147959058659439
```

[46]:
```
Text(0.5, 1.0, 'r = 1.01 slope = 0.8147959058659439')
```

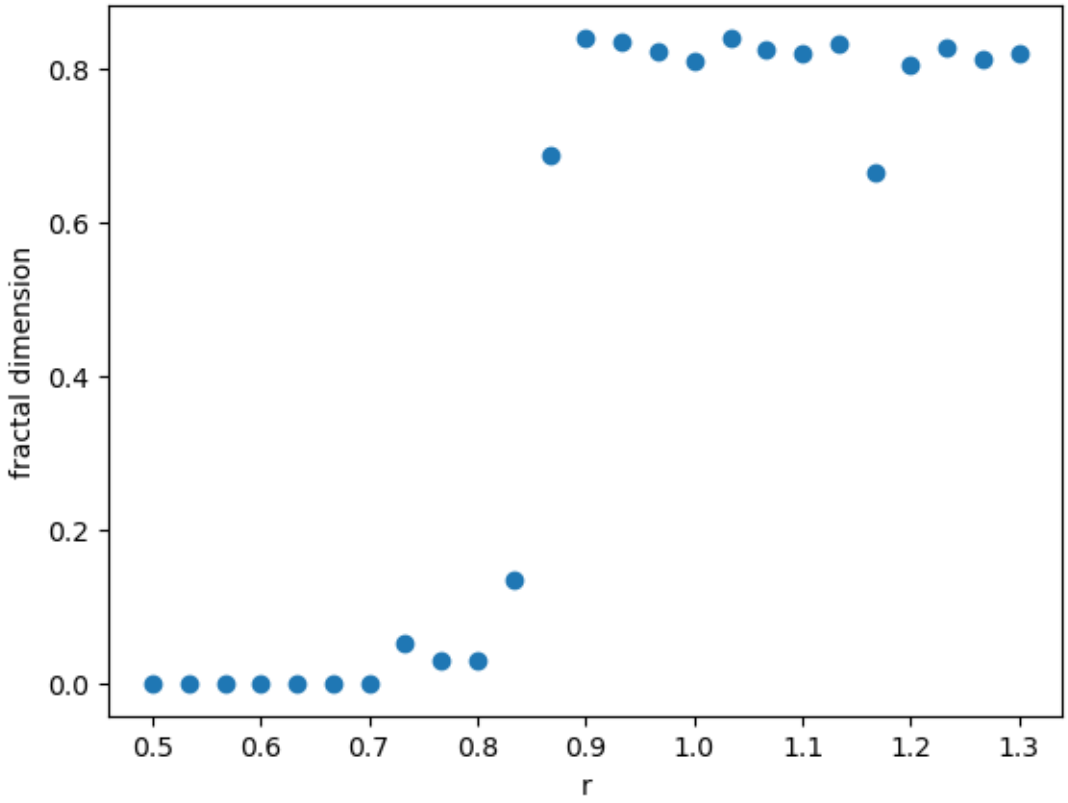


[48]:
```
def fractal_dimension(tr, eps):
    N_eps = [count_points(tr, eps[i]) for i in range(len(eps))]
    reg = LinearRegression().fit(np.log(eps).reshape(-1, 1), np.log(N_eps).
 ↪reshape(-1, 1))
    return reg.coef_[0][0]
```

[50]:
```
fr_dim = []

rangee = np.linspace(0.5, 1.3, 25)
for r in rangee:
    #print(r)
    seq = np.linspace(0, r, 20)
    tr = trajectory(r, 10000, x0)
    tr = tr[5000:]
    fd = fractal_dimension(tr, eps)
    fr_dim.append(fd)
```

[56]:
```
plt.plot(rangee, fr_dim, 'o')
plt.xlabel('r')
plt.ylabel('fractal dimension')
```

[56]: Text(0, 0.5, 'fractal dimension')



5 Ex 5: Universality in chaos

Add a new function to your code from Section 1.0, that iterates the logistic map:

$$x_{n+1} = \lambda x_n(1-x_n)$$

where is in the range(0,4),and x_0 is in the range(0,1).Note that I have used rand because they have different allowed ranges.

5.1 ex 5.1

Choose values of r and λ near the beginning of their range (but not zero), and plot the first iterate obtained from both maps on the same graph, i.e., if $f(x)$ is the logistic map and $g(x)$ is the sine map, plot $f(x_0)$ and $g(x_0)$ for many x_0 between 0 and 1. What do you observe? Can you find values of r, λ so that the curves nearly overlap?

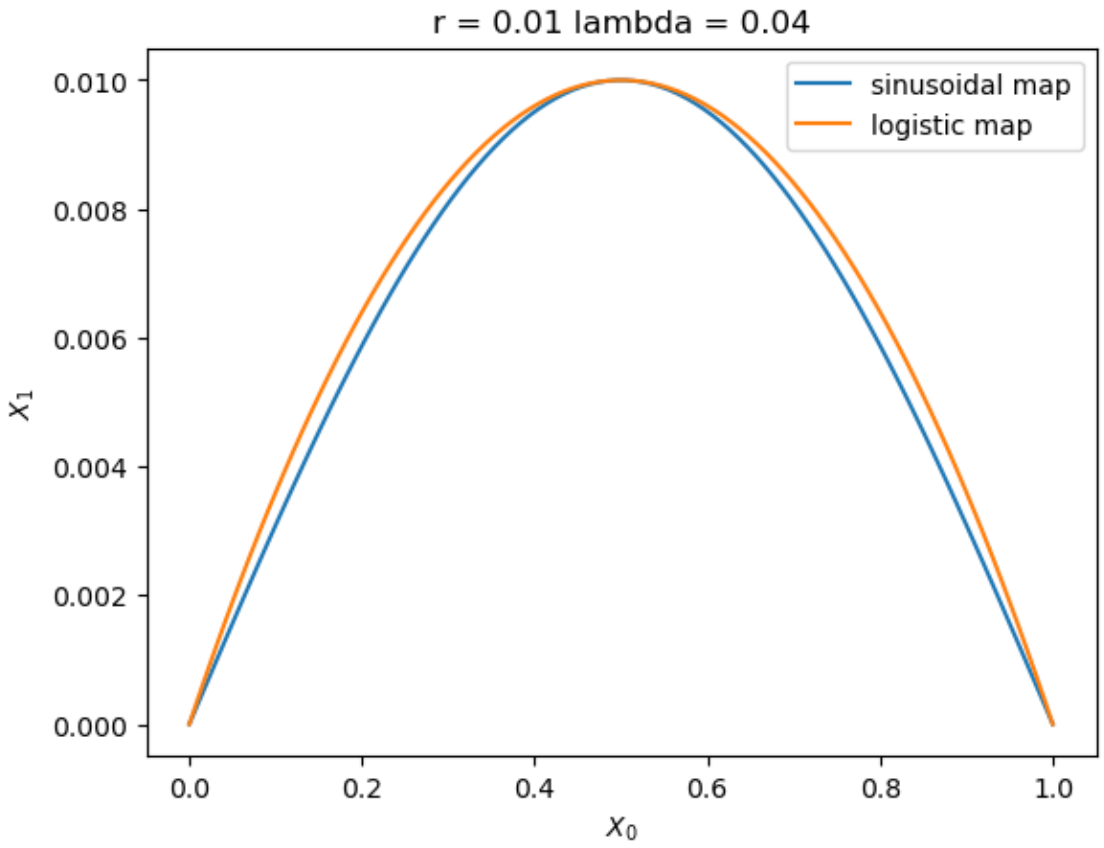
[83]:
```
from scipy.optimize import fsolve
import numpy as np
```

```
def logistic_map(x, lamb):
    '''
    Logistic map
    '''
    return lamb * x * (1 - x)
```

[84]:
```
def trajectory_log(lamb, N, x0):
    '''
    Compute the trajectory of the sinusoidal map
    '''
    x = np.zeros(N)
    x[0] = x0
    for i in range(1, N):
        x[i] = logistic_map_map(x[i-1], r)
    return x
```

[93]:
```
r = 0.01
lambdaa = 0.04

x0s = np.linspace(0, 1, 100)

x1s_sin =sin_map(x0s, r)
x1s_log = logistic_map(x0s, lambdaa)

plt.plot(x0s, x1s_sin, label='sinusoidal map')
plt.plot(x0s, x1s_log, label='logistic map')

plt.xlabel('$X_0$')
plt.ylabel('$X_1$')

plt.title('r = ' + str(r) + ' lambda = ' + str(lambdaa))

plt.legend()
```

[93]:
```
<matplotlib.legend.Legend at 0x1641426a0>
```



What property of the map functions do you think is necessary for two discrete maps to have similar long-time behaviour (referred to in the literature as being in the same universality class)?

[ ]:

5.2 Ex 5.2

Comment on how your bifurcation curve for the sine map compares to that obtained from the logistic map shown above.

5.3 Ex 5.3

What implications does this have for using simple maps like the logistic map for making predictions about complex natural system such as turbulent flow of fluids?