An Overview: Navigating Linear & Nonlinear Dynamics with Python code

Simon Leung
11 min read · Oct 11, 2024

What Is a Dynamical System?
A dynamical system is a mathematical framework used to describe the time-dependent behavior of complex systems. In essence, a dynamical system comprises variables or quantities that evolve over time based on specific rules. It can be either deterministic or stochastic, depending on whether randomness is involved. Also, it can be classified as linear or nonlinear, depending on the nature of its evolution rules.
Here's a more detailed breakdown:

Deterministic Dynamical Systems

Linear Deterministic Systems

Definition: A linear deterministic dynamical system is one in which the evolution rules are linear functions of the state variables, and the future state is entirely determined by the initial state and the rules governing its evolution.

Key Properties:

1. Superposition Principle: The response to a combination of inputs is the sum of the individual responses.
2. Homogeneity: Scaling the input results in a proportional scaling of the output.
3. Predictability: The future state can be precisely predicted given the initial state and the evolution rules.

Examples:

Continuous-Time Systems:

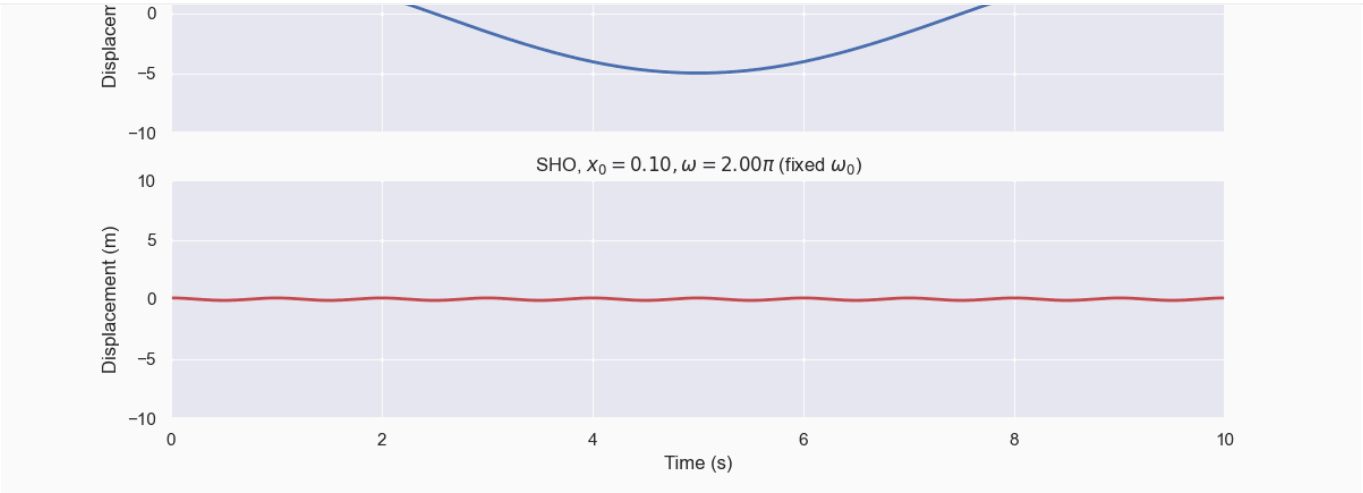
Example: A simple harmonic oscillator: $d^2x/dt^22 + \omega^2x = 0$, where x is the displacement and ω is the angular frequency.

Use SymPy to solve the equation of motion symbolically
(Full source code here)

```
# Define symbols
#t = sp.symbols('t')
omega, t = sp.symbols('omega t', positive=True, real=True)

# Define the displacement function
x = sp.Function('x')
```

```
# Define the differential equation
diffeq = sp.Eq( x(t).diff(t, t) + (omega**2) * x(t), 0)

# Solve the differential equation
solution = sp.dsolve(diffeq, x(t))
print("General Solution:\n")
#sp.pprint(solution)
display(solution)
```

$$x(t) = C_1 sin(\omega t) + C_1 cos(\omega t)$$



Use matplotlib.animation to plot the result

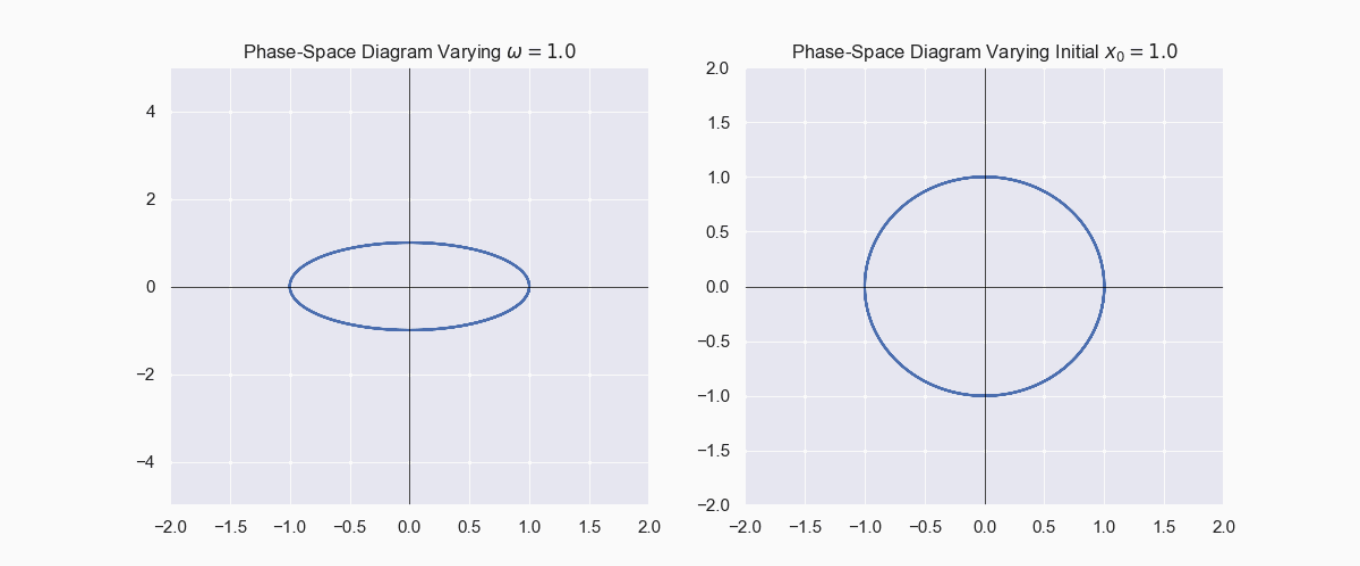
(Full source code here)

Phase-Space Diagram: Gives a complete picture of the system's state over time. Great for understanding the stability of systems and identifying attractors and repellers.

In dynamical systems, attractors and repellers are points or sets that

describe the behavior of the system over time. Attractors are states where the system tends to move toward and stay, like a ball rolling into a valley. Repellers, on the other hand, are states where the system tends to move away from, like a ball balancing on the top of a hill.

Attractors can be fixed points, cycles, or more complex structures like strange attractors. They represent stability in the system. Repellers indicate instability, pushing the system away toward different states.



Phase-space of SHM

In SHM, the phase-space diagram will show elliptical (or circular) trajectories centered around the origin. For SHM specifically:

- **Attractors**: Not typically present in ideal SHM as it's a conservative system (no damping).
- **Repellers**: Also not present in ideal SHM for the same reason.

Discrete-Time Systems:

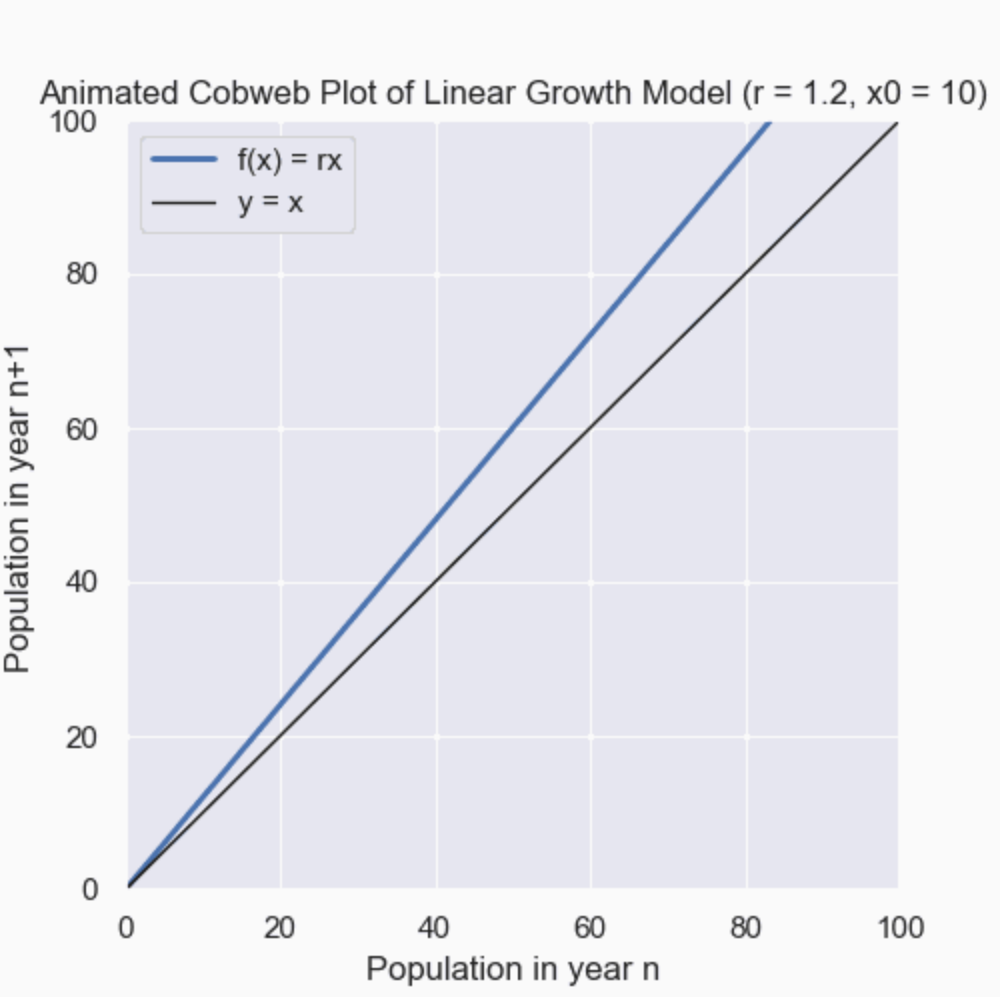
- Example: A first-order linear difference equation:

$$x_{n+1} = ax_n + b$$

where *a* and *b* are constants.

A cobweb plot, also known as a web diagram or stair-step diagram, is a graphical tool used to visualize the behavior of discrete dynamical systems, particularly iterated functions. It's especially useful for understanding the long-term behavior of these systems, including whether they converge to a fixed point, oscillate, or exhibit chaotic behavior. (Full source code here)

Note: A cobweb plot, known also as Lémeray Diagram or Verhulst diagram.



Nonlinear Deterministic Systems

Definition: A nonlinear deterministic dynamical system is one in which the evolution rules are nonlinear functions of the state variables, and the future state is entirely determined by the initial state and the rules governing its evolution.

Key Properties:

1. Non-Superposition: The response to a combination of inputs is not simply the sum of the individual responses.
2. Non-Homogeneity: Scaling the input does not result in a proportional scaling of the output.
3. Predictability: The future state can be precisely predicted given the initial state and the evolution rules, but the behavior can be complex and sensitive to initial conditions.

Continuous-Time Systems:

Example: The Lorenz system:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y - x),$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = x(\rho - z) - y,$$

$$\frac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z.$$

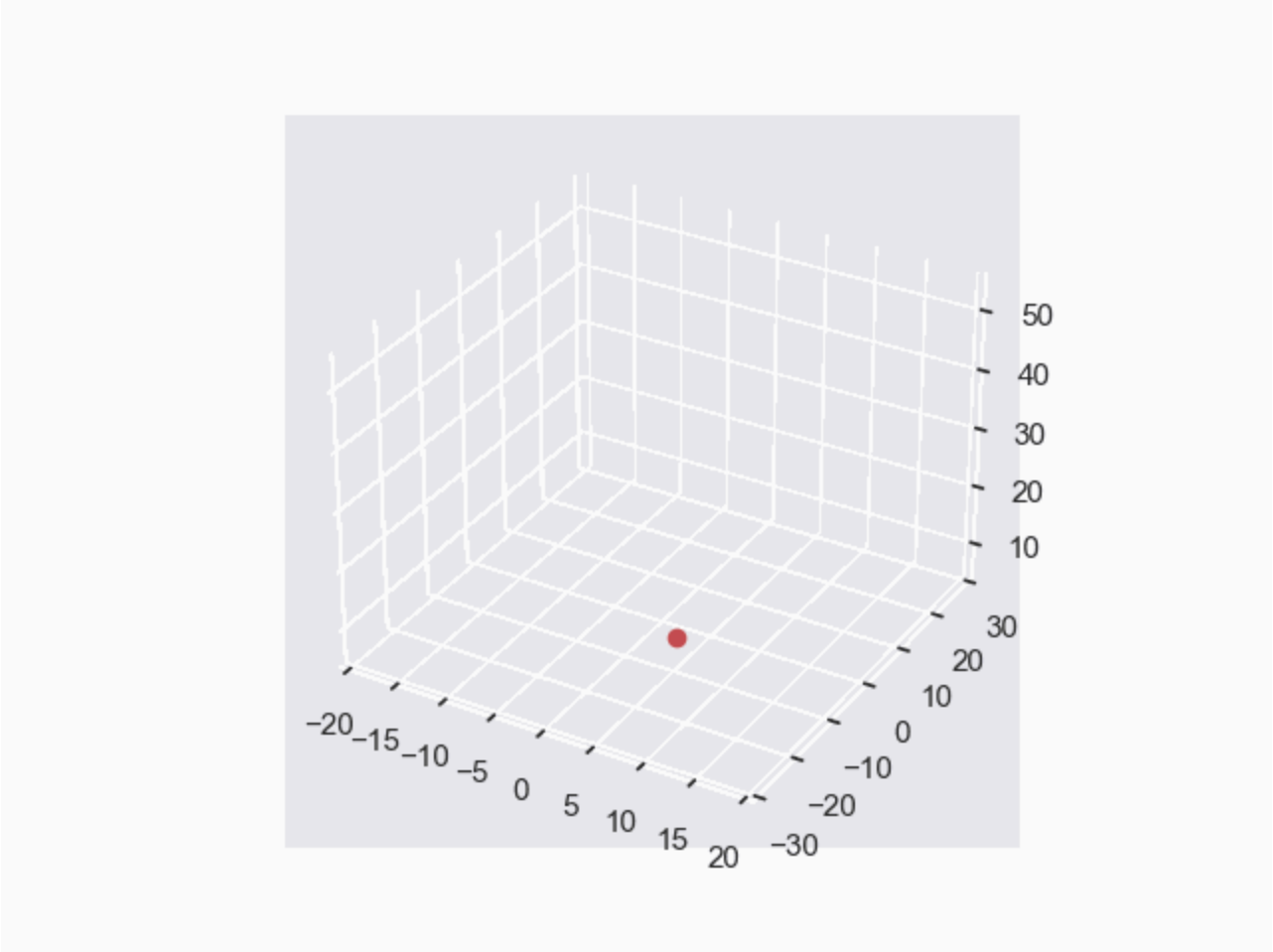
Image from Wikipedia

where σ, ρ, and β are parameters.

Solve these equations numerically using scipy: (Full source code here)

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Parameters
sigma = 10.0
rho = 28.0
beta = 8.0 / 3.0

# Lorenz system equations
def lorenz(X, t, sigma, rho, beta):
    x, y, z = X
    dxdt = sigma * (y - x)
    dydt = x * (rho - z) - y
    dzdt = x * y - beta * z
    return np.array([dxdt, dydt, dzdt])

# Time array
dt = 0.01
t = np.arange(0, 40, dt)

# Initial conditions
X0 = [1.0, 1.0, 1.0]

# Integrate the Lorenz equations
def integrate_lorenz(X0, t, sigma, rho, beta):
    X = np.empty((len(t), 3))
    X[0] = X0
    for i in range(1, len(t)):
        X[i] = X[i - 1] + lorenz(X[i - 1], t[i - 1], sigma, rho, beta) * dt
```

```
    return X

X = integrate_lorenz(X0, t, sigma, rho, beta)

# Plotting the Lorenz attractor
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.plot(X[:, 0], X[:, 1], X[:, 2], lw=0.5)
ax.set_title("Lorenz Attractor")
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
plt.show()
```



matplotlib.animation

For the Lorenz system, the fixed points (x,y,z) can be found by setting the derivatives to zero. Setting

$$\dot{x} = \dot{y} \quad = \dot{z} = 0$$

we get:
1. (0, 0, 0)
2.

$$(\pm\sqrt{\beta(\rho-1)}, \pm\sqrt{\beta(\rho-1)}, \rho-1)$$

Limit Cycles: The Lorenz system typically does not exhibit stable limit cycles. Instead, it shows chaotic behavior with strange attractors. The system's trajectories are highly sensitive to initial conditions and can exhibit a butterfly-shaped structure, but not traditional limit cycles.

Chaotic systems like the Lorenz attractor are mesmerizing because they defy predictability yet follow deterministic rules.

Additional articles on numerically solving the Lorenz System with Python:
Applications of Numerical Integration | Part 1 — Solving ODEs in Python
Higher Order Numeric Differential Equations(Python)
Lorenz-63 System Integration Using 4th Order Runge-Kutta Methods in Python
Python and Physics: Lorenz and Rossler Systems
Python, Complex Systems, Chaos and Lorenz Attractor

Another example: Van der Pol oscillator

$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x = 0$$

or

$$\begin{cases} \dot{x} = y \\ \dot{y} = \mu(1 - x^2)y - x \end{cases}$$

It is a non-conservative, oscillating system with non-linear damping. It evolves in time according to the second-order nonlinear differential equation. And μ is a scalar parameter indicating the nonlinearity and the strength of the damping.

The Van der Pol oscillator is a type of non-linear oscillator with some pretty interesting characteristics:

1. Damping: It has non-linear damping, meaning it can either lose or gain energy. When the amplitude of oscillation is small, the system adds energy, leading to an increase in amplitude. Conversely, at larger amplitudes, it dissipates energy, causing the amplitude to decrease. This balance keeps the oscillations from blowing up or dying down.

2. Self-Sustained Oscillations: It's known for producing self-sustained oscillations, meaning the system can maintain periodic oscillations without

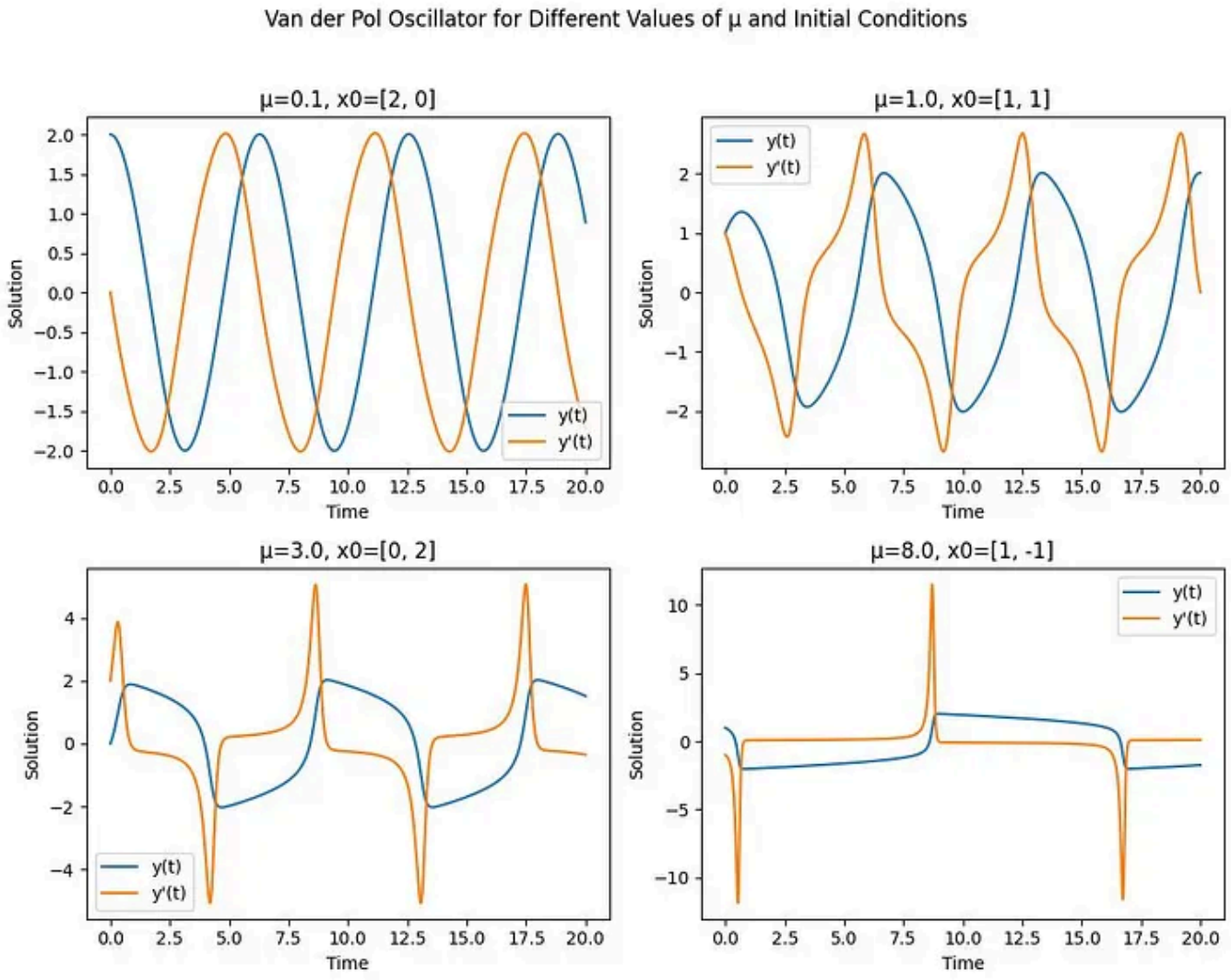
any external periodic force. This property is crucial for modeling biological rhythms and electronic circuits.

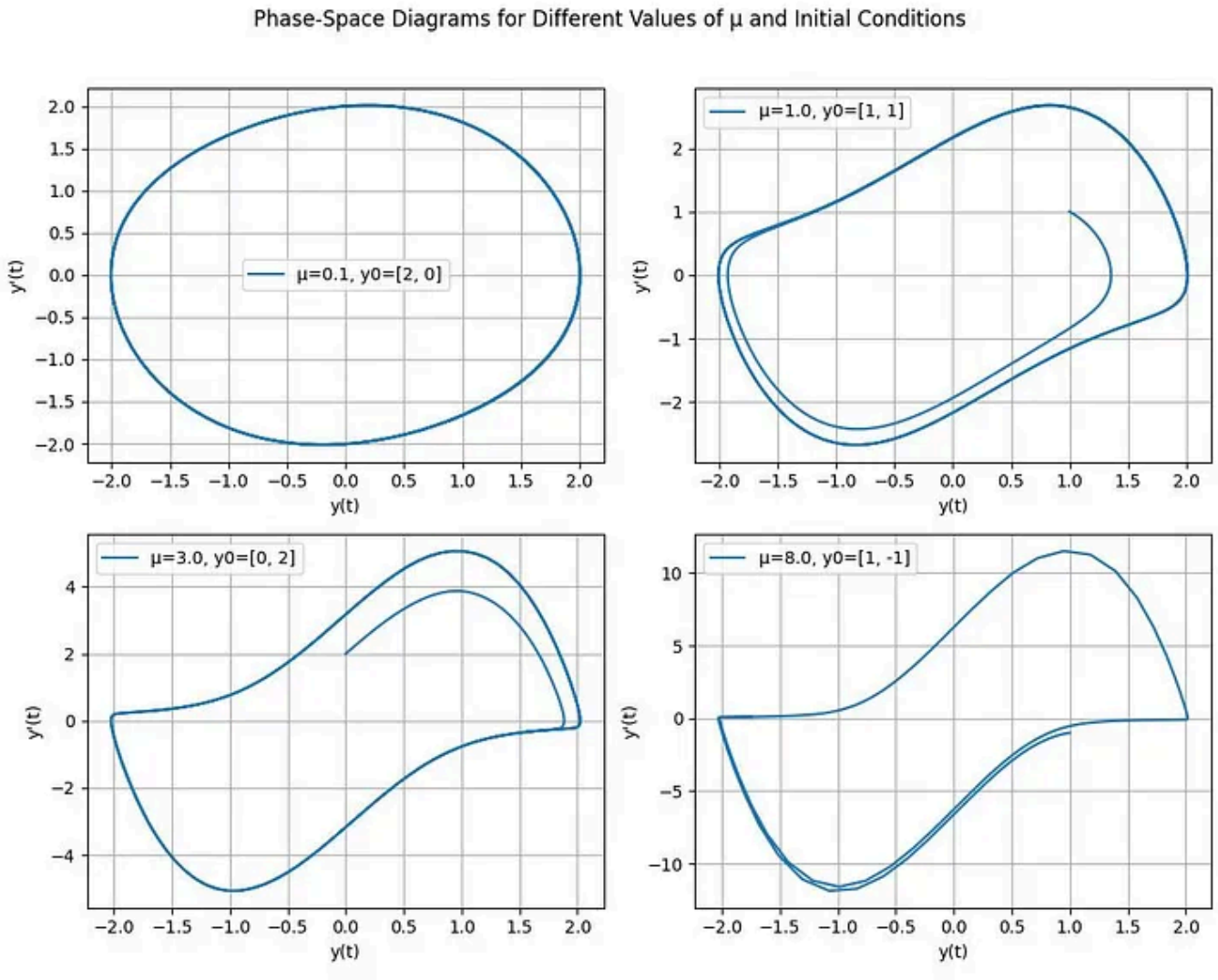
3. Limit Cycle: The oscillator exhibits what's called a limit cycle behavior. Regardless of the initial conditions, the system will eventually reach a stable, closed trajectory in its phase space. In simpler terms, after some transient period, the oscillations settle into a steady state.

4. Dependence on Parameter μ: The behavior of the Van der Pol oscillator is heavily influenced by the parameter μ. For small values of μ, the system behaves almost like a linear harmonic oscillator. As μ increases, the non-linear effects become more pronounced, leading to the distinctive characteristics mentioned above.
(Full source code here)

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

def van_der_pol(t, y, mu):
    dydt = [y[1], mu * (1 - y[0]**2) * y[1] - y[0]]
    return dydt

# Define different values for mu and initial conditions
mus = [0.1, 1.0, 3.0, 8.0]
initial_conditions = [[2, 0], [1, 1], [0, 2], [1, -1]]

t_span = [0, 20]
t_eval = np.linspace(t_span[0], t_span[1], 1000)

fig, axs = plt.subplots(2, 2, figsize=(10, 8))
fig.suptitle('Van der Pol Oscillator for Different Values of μ and Initial Condi

for i, (mu, y0) in enumerate(zip(mus, initial_conditions)):
    sol = solve_ivp(van_der_pol, t_span, y0, args=(mu,), t_eval=t_eval)
    ax = axs[i // 2, i % 2]
    ax.plot(sol.t, sol.y[0], label='y(t)')
    ax.plot(sol.t, sol.y[1], label="y'(t)")
```

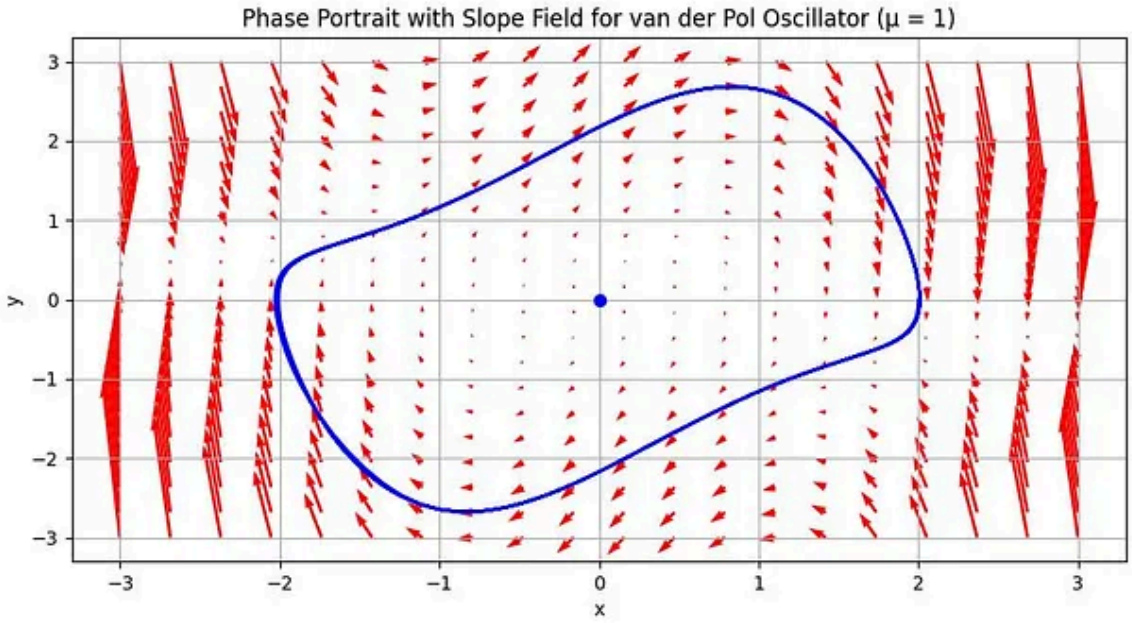
```
        ax.set_title(f'μ={mu}, x0={y0}')
        ax.set_xlabel('Time')
        ax.set_ylabel('Solution')
        ax.legend()

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    #plt.show()

    fig, axs = plt.subplots(2, 2, figsize=(10, 8))
    fig.suptitle('Phase-Space Diagrams for Different Values of μ and Initial Conditi

    for i, (mu, y0) in enumerate(zip(mus, initial_conditions)):
        sol = solve_ivp(van_der_pol, t_span, y0, args=(mu,), t_eval=t_eval)
        ax = axs[i // 2, i % 2]
        ax.plot(sol.y[0], sol.y[1], label=f'μ={mu}, y0={y0}')
        ax.set_xlabel('y(t)')
        ax.set_ylabel("y'(t)")
        ax.legend()
        ax.grid(True)

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()
```

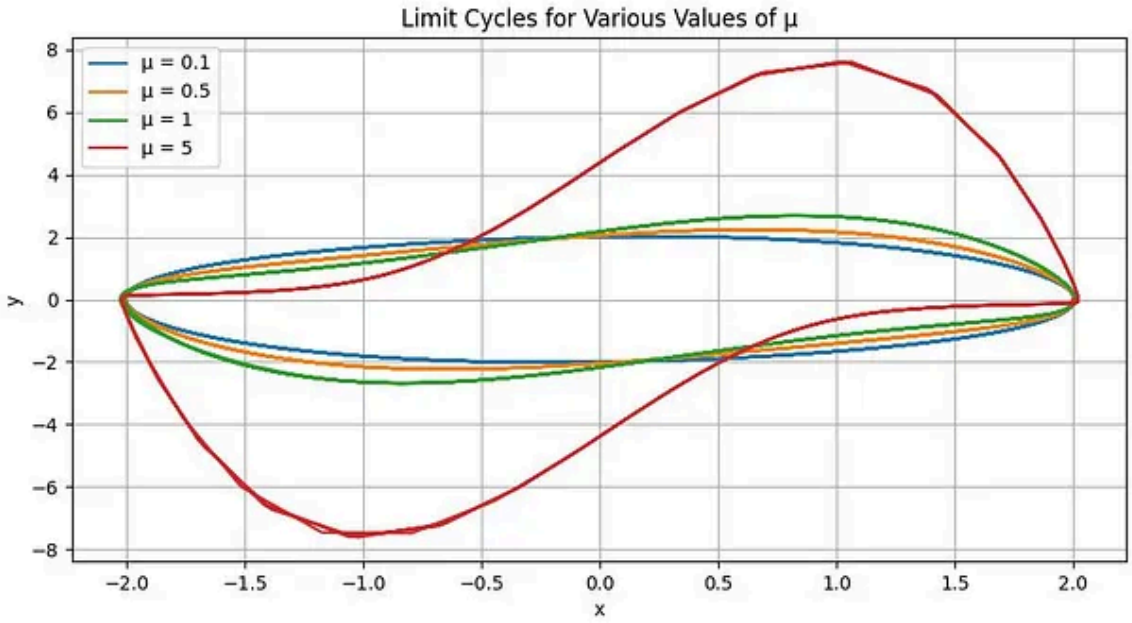

Van der Pol Oscillator for Different Values of μ and Initial Conditions
μ=0.1, x0=[2, 0]
μ=1.0, x0=[1, 1]
μ=3.0, x0=[0, 2]
μ=8.0, x0=[1, -1]
Solution
Time
y(t)
y'(t)




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

# Define the van der Pol oscillator differential equation
def van_der_pol(t, state, mu):
    x, y = state
    dxdt = y
    dydt = mu * (1 - x**2) * y - x
    return [dxdt, dydt]

# Generate the phase portrait with slope field using quiver
from scipy.integrate import solve_ivp

mu = 1.0
x = np.linspace(-3, 3, 20)
y = np.linspace(-3, 3, 20)
X, Y = np.meshgrid(x, y)
```

```
U, V = np.zeros(X.shape), np.zeros(Y.shape)

for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        U[i, j], V[i, j] = van_der_pol(0, [X[i, j], Y[i, j]], mu)

plt.figure(figsize=(10, 5))
plt.quiver(X, Y, U, V, color='r')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Phase Portrait with Slope Field for van der Pol Oscillator (μ = 1)')

# Highlight the limit cycle and the fixed point at (0, 0)
sol = solve_ivp(van_der_pol, [0, 50], [2, 0], args=(mu,), dense_output=True)
t = np.linspace(0, 50, 1000)
x, y = sol.sol(t)
plt.plot(x, y, 'b')
plt.plot(0, 0, 'bo')  # Fixed point
plt.grid()
plt.show()

# Plot the limit cycles for various values of μ
mu_values = [0.1, 0.5, 1, 5]
plt.figure(figsize=(10, 5))

for mu in mu_values:
    sol = solve_ivp(van_der_pol, [0, 50], [2, 0], args=(mu,), dense_output=True)
    x, y = sol.sol(t)
    plt.plot(x, y, label=f'μ = {mu}')

plt.xlabel('x')
plt.ylabel('y')
plt.title('Limit Cycles for Various Values of μ')
plt.legend()
plt.grid()
plt.show()
```





Further readings on this topic:

1. Change the period/frequency of a Van der Pol Oscillator
2. Stabilizing the Van der Pol Nonlinear Oscillator using Reinforcement Learning (RL) - (Part 1) (Part 2)(Part 3)(Part 4)

Discrete-Time Systems:

Example: The logistic map: $x\_n+1=ax\_n(1-x\_n)$, where r is a growth rate parameter.
A real life example is relation to cardiac arrhythmias [A Bifurcation Diagram for the Heart].
Application on logistic map: COVID-19 projection using data-science — A case for modeling epidemics using logistic map.

(Full source code here)

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

amin = 2.8
amax = 4.0
itmax = 200    # number of iterations for which we will calculate values
ivor = 1000    # number of iterations to ignore initially (to let the system set
tiny = 1e-6    # small initial value for x
da = (amax - amin) / 1000.0     # increment step for a

a_values = []
x_values = []
xs_values =[]

for a in range(int(amin * 1000), int(amax * 1000), int(da * 1000)):
    a /= 1000.0                             # scales a back to the correct range
    x = tiny                                # initializes x for each a
    for it in range(1, itmax + ivor + 1):
        x = a * x * (1.0 - x)
        if it > ivor:                       # store the values after ivor iterat
            a_values.append(a)
            x_values.append(x)
            xs_values.append(1 - 1 / a)

# Steady solution x_{s}(a) in bold
plt.plot(a_values, xs_values, 'b-', lw=2)
plt.axvline(x = 3, color = 'r', linestyle='-.')
```

```
plt.scatter(a_values, x_values, s=0.1)
plt.xlabel('a')
plt.ylabel('x', rotation=180)
plt.suptitle('Logistic Map')
plt.title("The stationary solution $x_s$ (blue) becomes unstable for a > 3", y=1
plt.show()
```

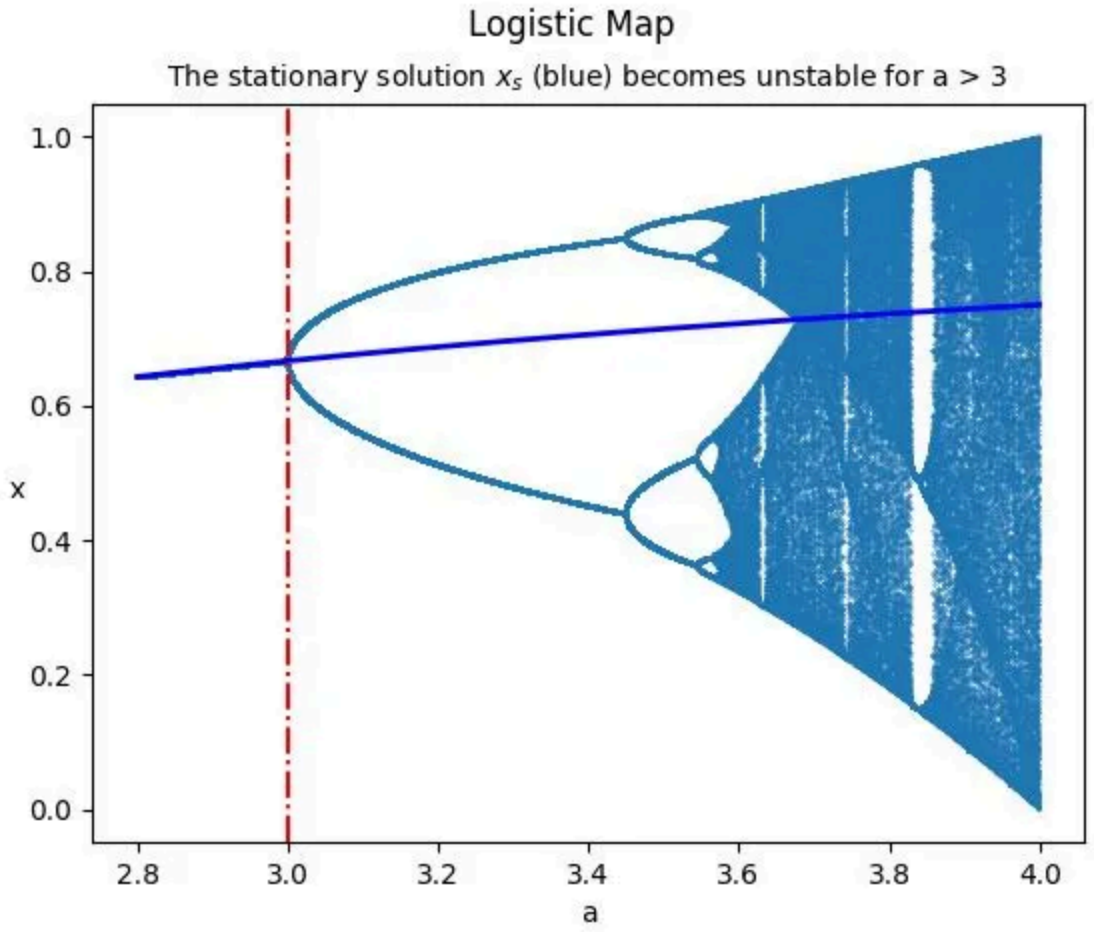
In the context of the logistic map, a period-2 cycle (or Period Doubling) means the system oscillates between two distinct values every iteration (when a>3, this is called *bifurcation*).

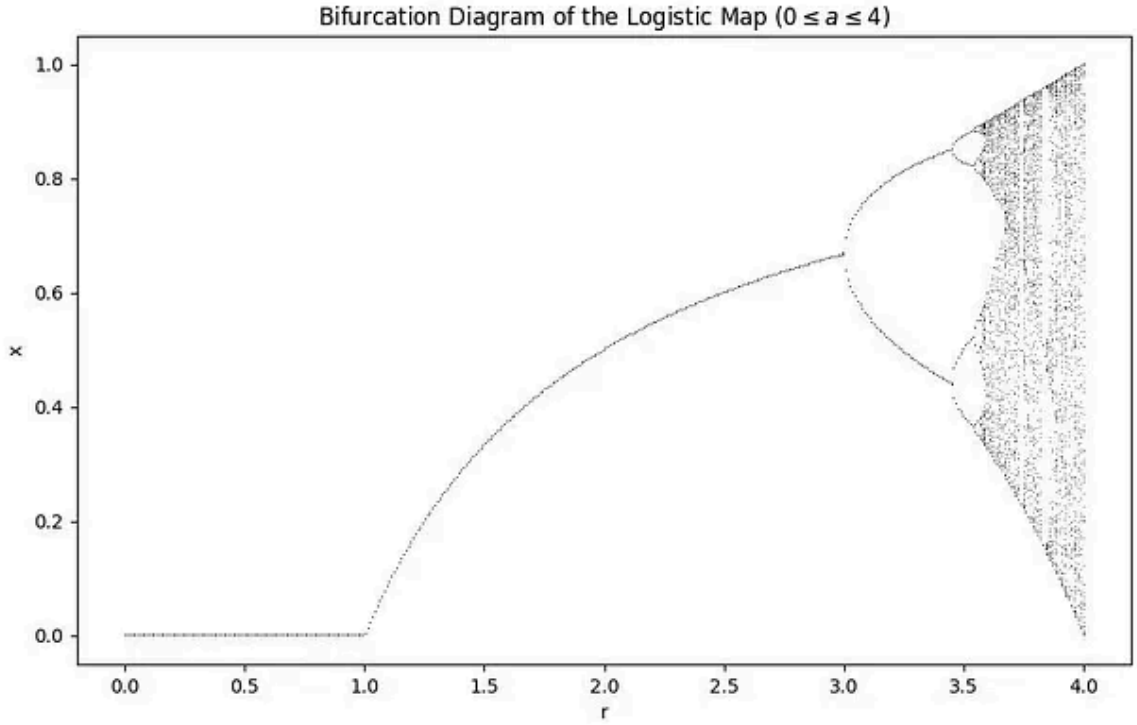
> *The period doubling continues with periods 2^N occurring at more and more closely spaced values of* a. *When* a > *3.569946, for many values of* a *the behavior is aperiodic, and the values of xj never form a repeating sequence. [4]*

If taking any two consecutive lengths of each bifurcation of the graph, the ratio of the consecutive lengths converges to the number is called the **Feigenbaum constant δ ~4.669….**

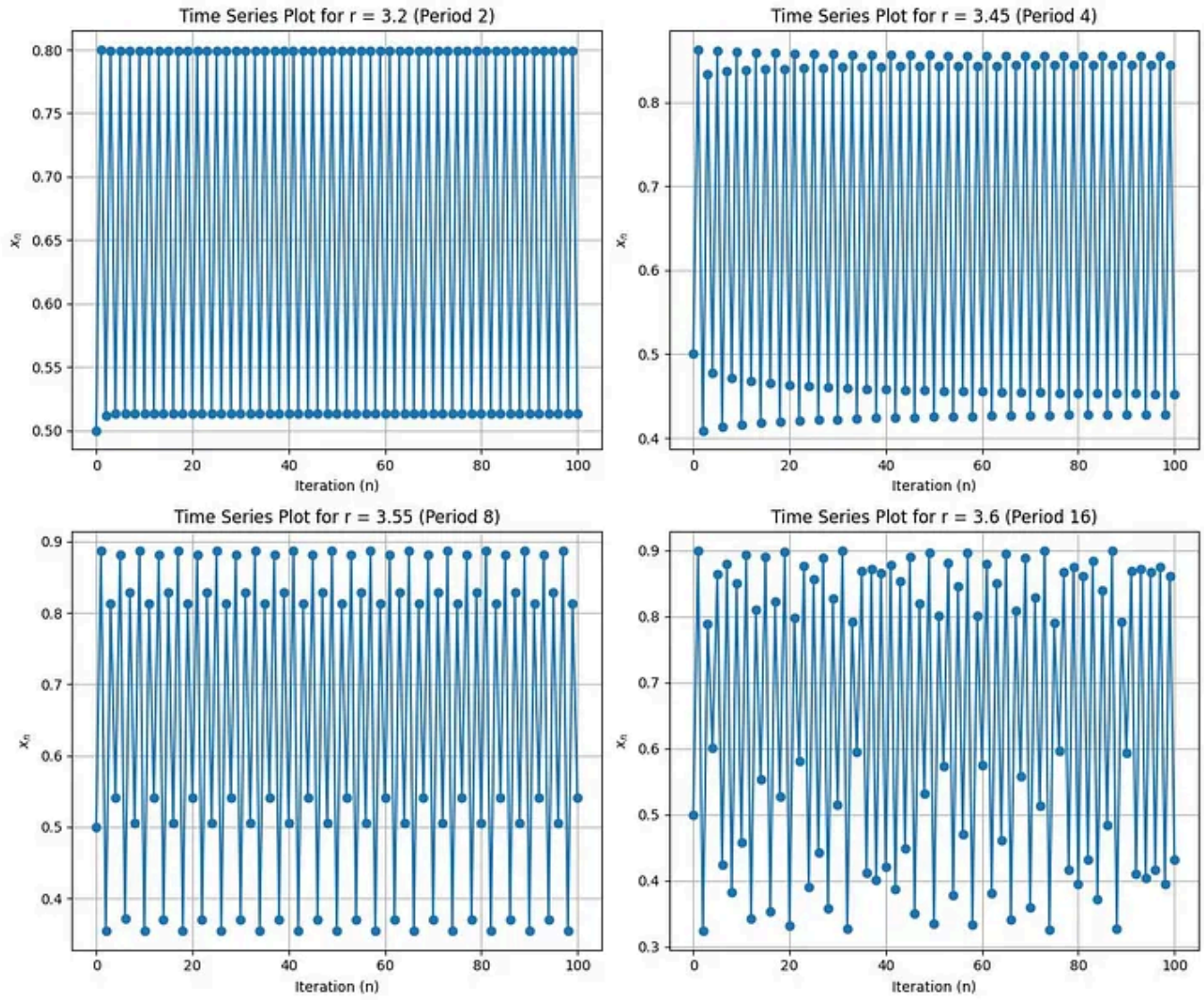
Further readings on this topic:

1. "Logistic Equation" with Python & Feigenbaum Constant
2. The Coolest Mathematical Constant You've Never Heard Of
3. Feigenbaum Constant
4. Period Three Implies Chaos
5. Chaos Theory and the Logistic Map
6. The Logistic Map: a Simple Model with Rich Dynamics

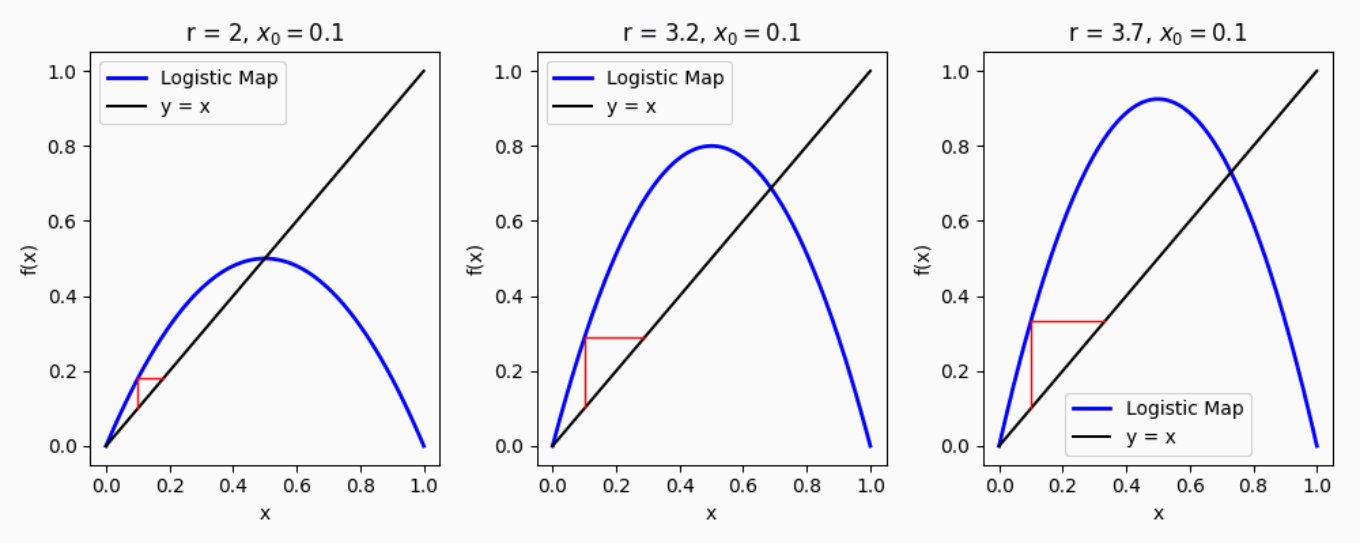

Logistic Map
The stationary solution x_s (blue) becomes unstable for a > 3
1.0
0.8
0.6
0.4
0.2
0.0
x
2.8
3.0
3.2
3.4
3.6
3.8
4.0
a


Bifurcation Diagram of the Logistic Map (0 ≤ a ≤ 4)
1.0
0.8
0.6
0.4
0.2
0.0
x
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
r

```
import matplotlib.pyplot as plt

def logistic_map(r, x0, n_iter):
    x = [x0]
    for _ in range(n_iter):
        x_next = r * x[-1] * (1 - x[-1])
        x.append(x_next)
    return x

# Parameters
r_values = [3.2, 3.45, 3.55, 3.6]
x0 = 0.5  # initial value
n_iter = 100  # number of iterations

# Create subplots
fig, axs = plt.subplots(2, 2, figsize=(12, 10))

for idx, r in enumerate(r_values):
    x = logistic_map(r, x0, n_iter)
    ax = axs[idx // 2, idx % 2]
    ax.plot(range(n_iter + 1), x, marker='o', linestyle='-')
    ax.set_title(f'Time Series Plot for r = {r} (Period {2**(idx+1)})')
    ax.set_xlabel('Iteration (n)')
    ax.set_ylabel('$x_n$')
    ax.grid(True)

# Adjust layout
plt.tight_layout()
plt.show()
```



In a cobweb plot for the logistic map, the parabola corresponds to the function f(x) = rx(1 — x), which represents the logistic map equation. The line y = x is also included in the plot as it helps to visualize the points where the function f(x) intersects with y = x. These points are known as fixed points, where the value remains the same in subsequent iterations.



1. *r=2* (*x=1/2* is an attractive fixed point)
2. *r=3.2* (*x=22/32* has a 2-periodic orbit)
3. *r=3.7* (*x=27/37* is a repulsive fixed point)

> *An attractive fixed point draws the values of successive iterations of the map closer to the fixed point, while a repulsive fixed point pushes those values away from the fixed point. There is also the possibility that the fixed point will neither attract nor repel successive iterations and instead produce a periodic orbit.*

quote from http://sites.saintmarys.edu/~sbroad/example-logistic-cobweb.html

Other articles on Logistic Map:
From Logistic Function to Logistic Map, to Chaos
Bifurcation and chaos

Thank you for reading,

To Be Continued……. Stochastic Dynamical Systems ……

Recommended books:

IPython Interactive Computing and Visualization Cookbook 2nd Ed, Cyrille Rossant, (2018 Packt Publishing)

Computational Physics 4th Ed : Problem Solving with Python, Rubin H Landau, Manuel J Paez & Cristian Bordeianu (2024, Wiley-VCH)

Nonlinear Dynamics and Chaos 3rd Ed, Steven H. Strogatz (2024, Chapman and Hall/CRC)

Gentle Introduction To Chaotic Dynamical Systems, Vincent Granville, (2023, www.MLTechniques.com)

Computational Physics, Mark Newman (2013, CreateSpace)

Chaotic Dynamics — An Introduction Based on Classical Mechanics, Tamás Tél, Márton Gruiz (2006, Cambridge)

Chaos and Time-Series Analysis, J. C. Sprott (2001, Oxford University Press)

Intermediate Physics for Medicine and Biology 5th Ed, Russell K. Hobbie, Bradley J. Roth (2015, Springer)

S

Written by Simon Leung

129 Followers · 4 Following

A fervent enthusiast, fueled by an insatiable curiosity for STEM (Science, Technology, Engineering, and Mathematics) disciplines.