Exploring the Dynamics of Logistic Equations via Python (Final Draft)

January 3, 2026

1 Introduction:

The Logistic Equation has seen some real world applications especially on the global scene. During the Covid-19 Pandemic the Logistic Growth Model was used to predict the growth of the disease in several countries. The model was dependent on three parameters, the growth rate of the infected cases, the total number of cases in the last phase of the pandemic, as well as the total time of the phases. Models like this are paramount to preventing another global pandemic, and using this information we can suggest improvements to countries Public Health policies. The sigmoid function also known as the Logistic function, is a solution to the Logistic Growth Differential Equation. Represented as $f(x) = \frac{1}{1+e^{-x}}$, this has an important use as an activation function in neural networks. An activation function is a function that determines the output of a neuron in the network. This is important as the activation function adds a form of complexity to the neural network by introducing nonlinearity in its model. Without the nonlinearity, then the neural network would only be able to determine linear models.

2 Methodology:

The Logistic Function can be defined as $f(x) = \frac{1}{1+e^{-x}}$ The Logistic Function is an important equation used in many aspects of math. Also called a Logistic Curve or the Sigmoid, this function is a S-shaped curve when plotted on a graph that ranges from (0,1). It has seen different uses from a representation to population growth to a very important part to neural networks. It is important in neural networks as this function, when used as an activation function in the hidden layer, this allows the neural network to add a degree of non-linearity to the network. Without this function in a neural network, it would only be able to learn linearly separable problems.

The Logistic Growth Differential Equation can be defined as $\frac{dP}{dt} = rP(1 - \frac{P}{M})$. As discussed in the previous section, the Logistic Function is a solution of the Logistic Growth ODE. This is a differential equation represents the rate of change of the population with respect to time. A helpful tool to use when modeling population growth. One thing to note about this equation is that this starts off exponentially, but slows down as it approaches the maximum limit M. This is interesting in the equation as this puts into consideration resource limitations in a populations, with $(1 - \frac{P}{M})$ as a self regulating term. This equation uses k, which is the intrinsic growth rate which represents how fast the population grows also known as the proportionality constant, this number is usually very small and less than one.

Beginning with $\frac{dX}{dt} = \sigma X(1-X)$ and using forward difference, $\frac{dX}{dt}$ can be expressed as

$$\frac{dX}{dt} = \frac{X_{n+1} - X_n}{\Delta t}$$

and substituting we get

$$\frac{X_{n+1} - X_n}{\Delta t} = \sigma X_n(1 - X_n)$$

rearranging to get in the form of the Logistic map

$$Y_{n+1} = \rho Y_n(1 - Y_n)$$

we get

$$X_{n+1} = X_n + \Delta t \sigma X_n(1 - X_n)$$
$$X_{n+1} = X_n(1 + \Delta t \sigma(1 - X_n))$$

here we can see

$$X_n = Y_n$$

and

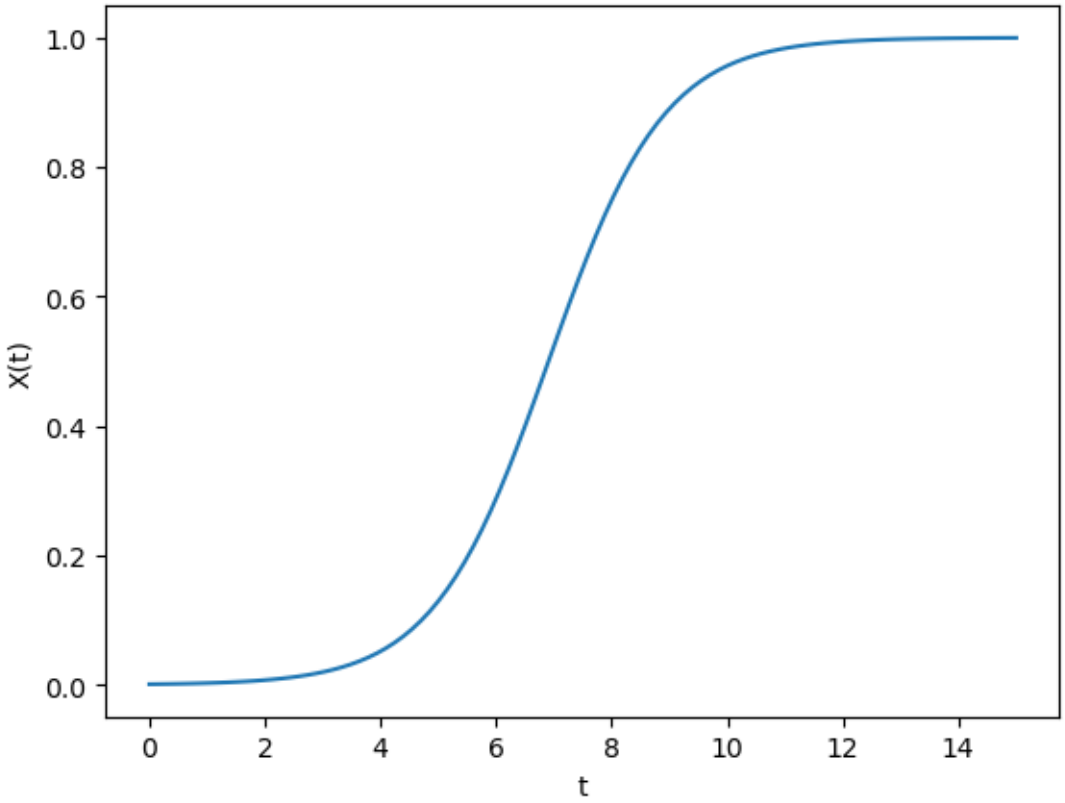
$$\rho = 1 + \Delta t \sigma$$

The continuous logistic equation describes how a population grows over time, considering both the natural growth rate and the carrying capacity of the environment. This equation, referred to as the logistic ODE (ordinary differential equation), can be discretized to form the logistic map. Using Euler's forward method, we approximate the population at the next time step by using the current population and the rate of change. The logistic map is a discrete-time counterpart to the logistic equation, showing how the population evolves in discrete time steps instead of continuously. The differences between the logistic function and logistic map can be seen on a graph. The continuous logistic equation will be a smooth S shaped curve that will model gradual changes in population. The discretized logistic map however will plot data points, with periodic changes or chaotic behavior depending on the parameters set. Consider the following code to represent the logistic equation:

[28]:
```
from sympy import symbols, lambdify
from sympy import dsolve, Function, Eq, Derivative
import matplotlib.pyplot as plt
import numpy as np

t,sigma = symbols('t,sigma')
X = Function('X')

dX_dt = sigma * X(t) * (1 - X(t))

# Define the equation
logistic_eq = Eq(Derivative(X(t), t), dX_dt)

# Solve the equation symbolically with the initial condition X(0) = 0.001
solution = dsolve(logistic_eq, X(t), ics={X(0): 0.001})
```

```
display(solution)

lamX = lambdify((t,sigma),solution.rhs,modules=['numpy'])

sigma_val = 1
t_val = np.linspace(0, 15, 100)
X_val = lamX(t_val, sigma_val)

plt.plot(t_val, X_val)
plt.xlabel('t')
plt.ylabel('X(t)')
plt.show()
```

$$X(t) = \frac{1}{1 + 999.0e^{-\sigma t}}$$



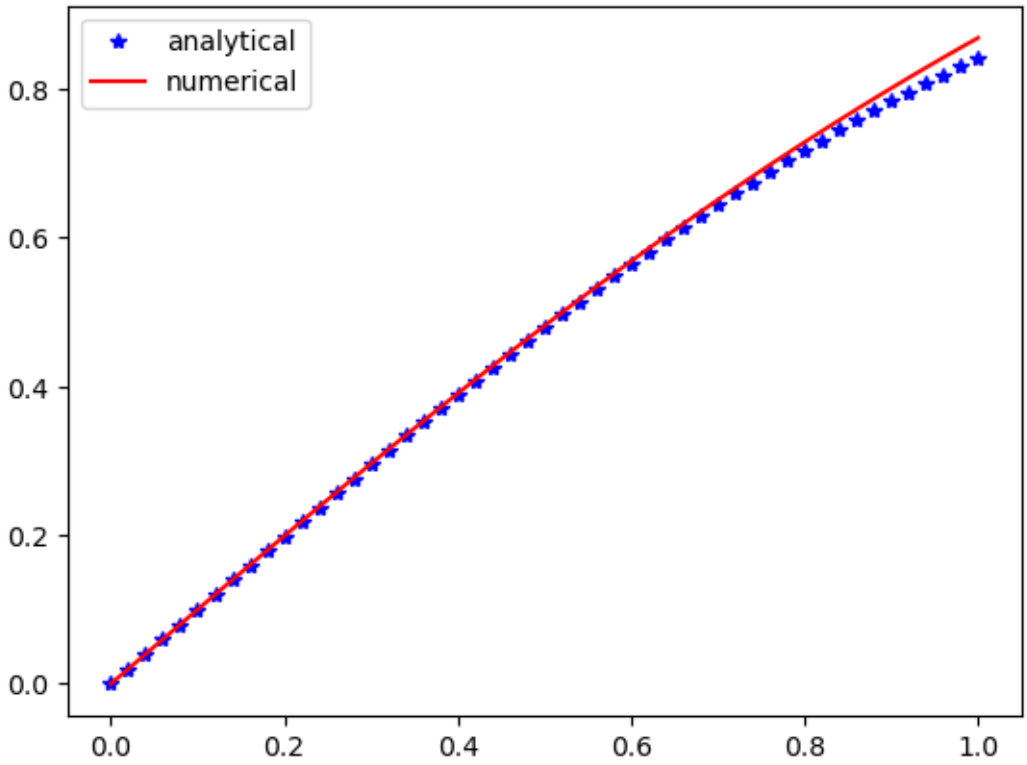
From above we can see the general shape of the logistic equation, as well as the code to generate the plot of the logistic equation. In order to create the graph of the logistic equation, we use python programming packages to assist in generating a graph. SymPy for example is used in the above code to define the logistic differential equation, solve it symbolically, and convert the solution to a form that can be evaluated numerically. Lambdify is used to convert our symbolic solution into a numerical solution, which we can then evaluate at specific values for t and sigma. NumPy is

also used here for numerical computation, we use it to create an array of time values over which the solution is evaluated. Lastly from the above code, in order to graph and plot our results we use Matplotlib. Here, Matplotlib is used for graphical representation of the solution to the logistic differential equation. Each of these programing libraries work together, Sympy defines the logistic equation and solves it symbolically. Numpy generates time values and uses the numerical function to compute the population at each time point. Matplotlib visualizes the results to see the population's behavior over time.

3 Python-Driven Analysis of Key Characteristics

When comparing the differences of symbolic, graphical, and numerical analysis of functions and plots there are a few key differences to note. As previously discussed, symbolic analysis manipulates expressions in their symbolic form and is useful to obtain general solutions or understand the relationship between the variables in the equation. In doing such, we can get accurate and exact results by substituting and value for variables used. An example of this can be seen in the equation used above.Numerical analysis can approximate solutions for a problem when finding an exact solution is difficult. This can typically be used when we have non linear equations that cannot be solved symbolically. An example of this is Euler's forward propagation method:

[32]:
```
### Euler method (forward)
dt = 0.02
y0 = 0

# t vals
t=np.arange(0,1+dt,dt)

y = np.zeros(len(t))
y[0] = y0

for i in range(0,len(t)-1):
    y[i+1] = y[i] + dt*np.cos(y[i])

analytical = np.sin(t)

#plot
plt.plot(t,analytical,'b*')
plt.plot(t,y,'r-')
plt.legend(['analytical','numerical'])
plt.show()
```

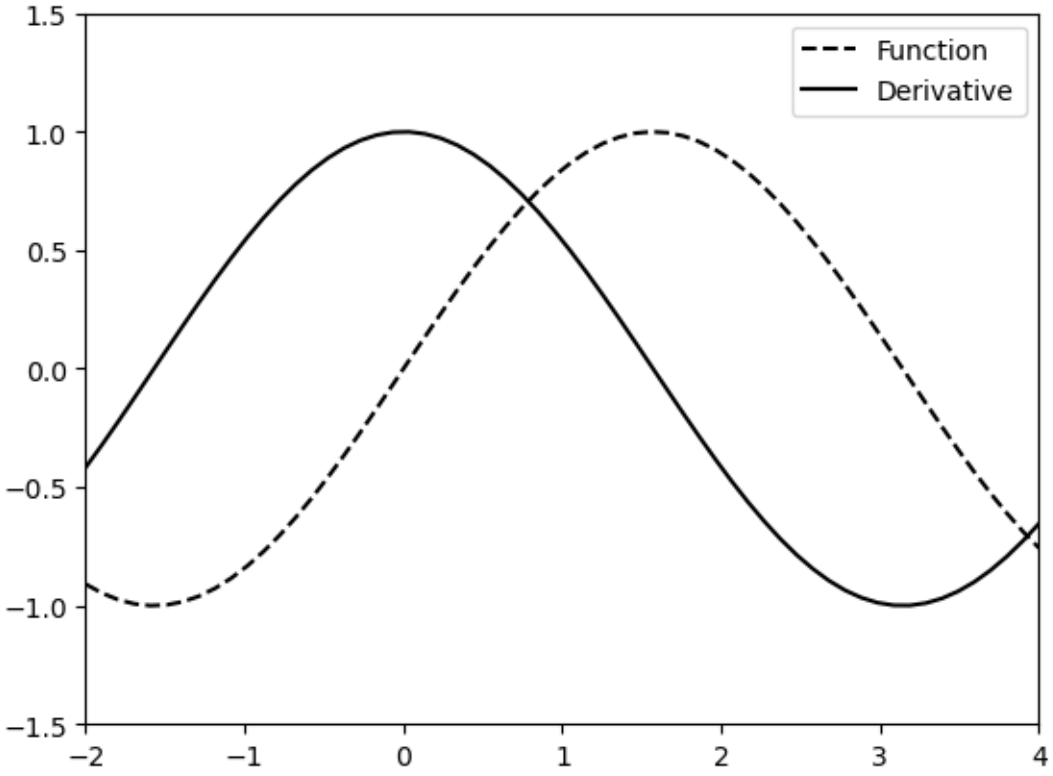


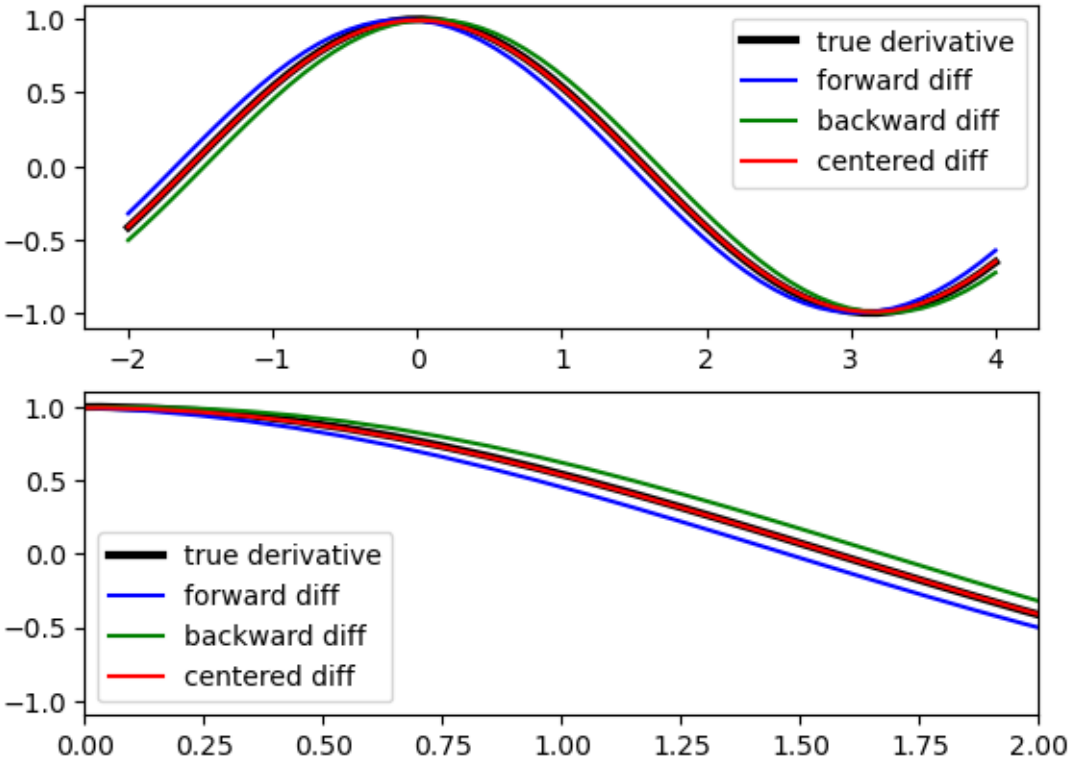
Graphical analysis, using Matplotlib involves visualizing the solutions in the forms of plots and graphs. This type of analysis is useful when observing behaviors of graphs such as its stability or long term trends. This type of analysis is useful for detecting growth and decay, and with the logistic map chaotic patterns as well. While it does provide a visual representation it must be noted that graphical analysis cannot provide accurate or exact values very well.

These methods are useful in the use of finite difference schemes when computing linear and non linear growth rates. Finite difference schemes are used to approximate solutions to differential equations. In growth models like the logistic equation, finite difference schemes allow us to calculate growth rates at certain time steps for real world applications. Linear growth rates are constant over time, where population increases at a constant while non linear growth rates, like the logistic equation, are non constant. We can use the forward difference method to approximate solutions for both linear and non linear growth, but we can also use center and backwards when approximating solutions as well. Consider the following:

[34]:
```
## time step
dt = 0.2

# create a t array
t = np.linspace(-2,4,60)

# our function
f = np.sin(t)
```

```
# exact or true derivative
dfdt = np.cos(t)

#plot
plt.plot(t,f,'k--') # Function
plt.plot(t,dfdt,'k') # Derivative
plt.legend(['Function','Derivative'])
plt.xlim(-2,4)
plt.ylim(-1.5,1.5)
plt.show()


# forward difference
dfdtF = (np.sin(t+dt)-np.sin(t))/dt

# backward difference
dfdtB = (np.sin(t)-np.sin(t-dt))/dt

#cd
dfdtC = (np.sin(t+dt)-np.sin(t-dt))/(2*dt)

plt.subplot(2,1,1)
plt.plot(t,dfdt,'k',linewidth=3) #true derivative cos
plt.plot(t,dfdtF,'b') #FD
plt.plot(t,dfdtB,'g') #BD
plt.plot(t,dfdtC,'r') #CD
plt.legend(['true derivative','forward diff','backward diff','centered diff'])

plt.subplot(2,1,2)
plt.plot(t,dfdt,'k',linewidth=3) #true derivative cos
plt.plot(t,dfdtF,'b') #FD
plt.plot(t,dfdtB,'g') #BD
plt.plot(t,dfdtC,'r') #CD
plt.legend(['true derivative','forward diff','backward diff','centered diff'])
plt.xlim([0,2])
plt.show()
```


Function
Derivative
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2
3
4

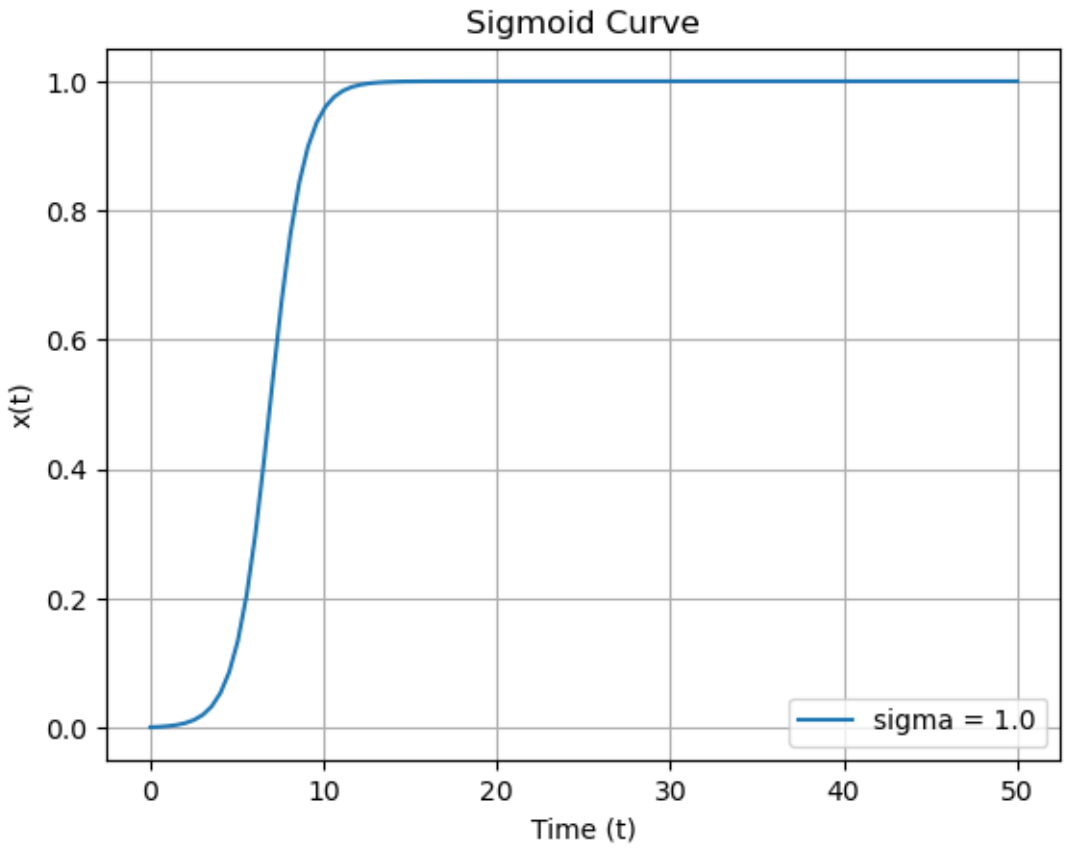

true derivative
forward diff
backward diff
centered diff
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2
3
4
true derivative
forward diff
backward diff
centered diff
1.0
0.5
0.0
−0.5
−1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Using these approximation methods we can see the center difference scheme being the most accurate of each of the 3 different types. Using both the previous and next point in time, it has a smaller error than both forward and backward differences. The use of these methods provide additional information when analyzing growth.

Knowing the Sigmoid curve which is represented as such, $f(x) = \frac{1}{1+e^{-x}}$ is a solution to the Logistic Growth Differential Equation. We can see that this is an orderly growth curve. This is a smooth curve and is also a stable system, meaning there is no chaotic pattern shown in this curve. A benefit of this is that the solutions on this curve are predictable and deterministic. This is good as we will see in the next section that predictable and deterministic will go out the window as we examine how taking the r value and increasing it can change the behavior of the model in a discretized version of the Logistic Map.

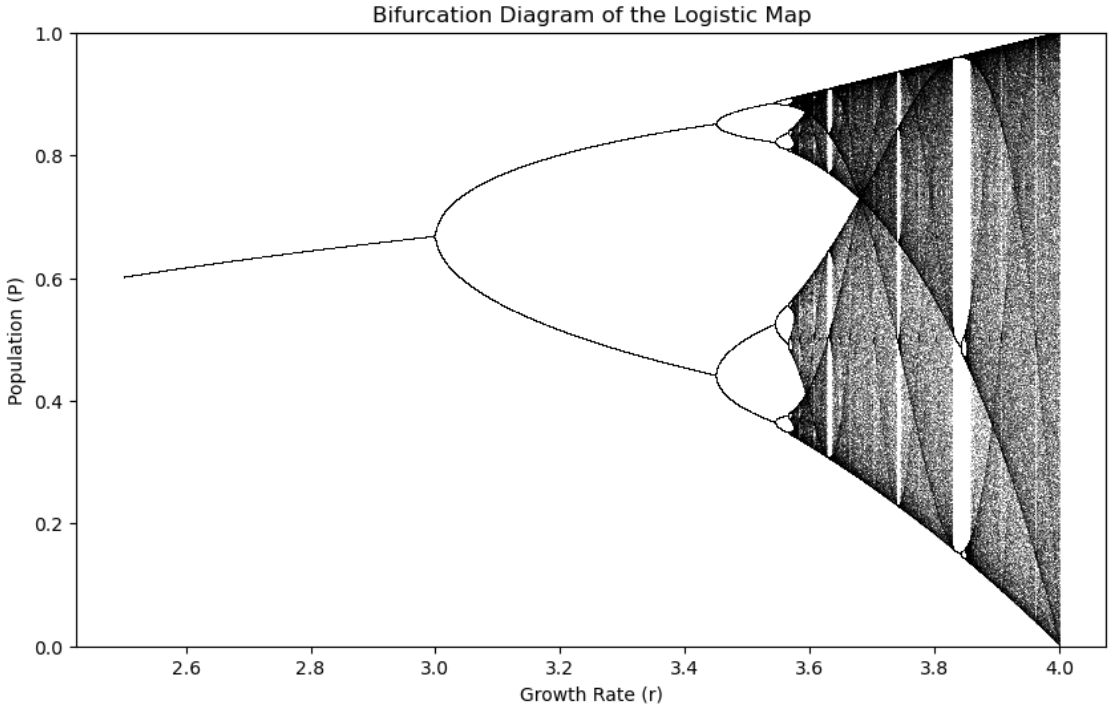
[38]:
```
import sympy as sp
t, sigma = symbols('t sigma')
x = Function('x')(t)
x0 = 0.001

logistic = sp.Eq(Derivative(x,t), sigma * x * (1-x))

solution = sp.dsolve(logistic, x, ics={x.subs(t, 0): x0})
display(solution)

# (B)

solx = lambdify((t, sigma), solution.rhs, 'numpy')

sigma_val = 1.0
time = np.linspace(0,50,100)
xvals = solx(time, sigma_val)

plt.plot(time, xvals, label = f'sigma = {sigma_val}')
plt.title('Sigmoid Curve')
plt.xlabel('Time (t)')
plt.ylabel('x(t)')
plt.legend()
plt.grid(True)
plt.show()
```

$$x(t) = \frac{1}{1 + 999.0e^{-\sigma t}}$$



[1]:
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters for the logistic map
r_values = np.linspace(2.5, 4.0, 10000)  # Growth rates
iterations = 1000  # Number of iterations per r value
last = 100  # Number of points to plot after transient behavior

# Initialize the population and storage for results
P = 0.5  # Initial population
populations = np.empty((len(r_values), last))

# Iterate over each r value
for i, r in enumerate(r_values):
    P_temp = P  # Reset initial population
    # Iterate for the logistic map
    for j in range(iterations - last):
        P_temp = r * P_temp * (1 - P_temp)  # Update population (discard␣
 ↪transient)
    for k in range(last):
```

```
        P_temp = r * P_temp * (1 - P_temp)  # Update population (record final values)
        populations[i, k] = P_temp

# Plot the bifurcation diagram
plt.figure(figsize=(10, 6))
for i in range(last):
    plt.plot(r_values, populations[:, i], ',k', alpha=0.25)  # Add scatter points for each r value

plt.title("Bifurcation Diagram of the Logistic Map")
plt.xlabel("Growth Rate (r)")
plt.ylabel("Population (P)")
plt.ylim(0, 1)
plt.show()
```



Now look at the discrete form of the logistic map represented by, $P_{n+1} = rP_n(1-P_n)$ P_n represents the population at the nth time, and r is the growth parameter. When r is from 0 to 3 the model is stable and does not diverge. Once r>3 we see the first bifurcation and when r $>$ 3.45 we can see the chaotic behavior of the system. This is interesting as it shows how orderly this system begins but as the growth rate increases the system becomes unreliable due to the chaotic behavior. Good thing for purposes of estimating population growth the r variable is a constant so making sure not to have any r value over 3.45 would still produce a reliable model.

4 Conclusions

From our analysis we can conclude that the logistic equation is a powerful mathematical tool that helps explain how populations and systems grow. We've shown how this equation models growth that starts rapidly but slows as it approaches a maximum capacity. This is shown by the sigmoid function, which forms an S-shaped curve. An important aspect of the logistic equation is how it accounts for real-world constraints. Growth happens quickly because there are few limitations. As the population increases, competition for resources slows the growth rate. We also explored the logistic map, a discretized version of the logistic equation. It shows how systems can shift from stable behavior to cycles and chaotic patterns as conditions change. This shows the sensitivity of systems, where small changes can make a difference, and thus making the logistic equation useful for studying dynamic processes. Our analysis involved mathematical solutions, computer-based calculations, and graphical representations. Each method provided unique insights: exact answers from math, approximations through computation, and clear visual patterns from graphs. Together these allow us to see how these systems change. In conclusion, the logistic equation is extremely useful as a mathematical formula. It reveals how systems grow and adapt. Its ability to explain both steady and chaotic behavior makes it essential for understanding complex systems across math and science.

5 Author Contributions:

Conceptualization, Michael Williams and Raynell Labayog; methodology, Michael Williams and Raynell Labayog; software, Michael Williams and Raynell Labayog; writing—original draft preparation, Michael Williams and Raynell Labayog; writing—review and editing, Michael Williams and Raynell Labayog, All authors have read and agreed to the published version of the manuscript.

6 Appendices

References:

Michael Williams, 2024: Report on Homework Assignment 9. Math 340 Programming in

Mathematics. San Diego State University, 2024.

Michael Williams, 2024: Report on Homework Assignment 10. Math 340 Programming in

Mathematics. San Diego State University, 2024.

Michael Williams, 2024: Report on Lab Week 9. Math 340 Programming in

Mathematics. San Diego State University, 2024.

Agus Kartono, Setyanto Tri Wahyudi, Ardian Arif Setiawan, and Irmansyah Sofian, 2024: Predicting of the Coronavirus Disease 2019 (COVID-19) Epidemic Using Estimation of Parameters in the Logistic Growth Model. 2021.

[ ]: