The Annotated Guide to Logistic Equations with Python

Olivia Zhou

and

Bo-Wen Shen*, Ph.D.

Web: http://bwshen.sdsu.edu

Department of Mathematics and Statistics

San Diego State University

Summer 2024

*Dr. Bo-Wen Shen (bshen@sdsu.edu)

San Diego State University, Summer 2024

Abstract

In this report, we analyze the continuous and discrete forms of the logistic equations, showcasing solutions derived through manual calculations, symbolic computation, and numerical methods. We begin by introducing the logistic ordinary differential equation (ODE), Robert May's contributions involving the logistic ODE, and Euler's method for computing numerical solutions. Next, we outline our approach by solving the analytical solution of the logistic ODE, deriving the logistic difference equation, which is mathematically equivalent to the logistic map, and utilizing Python libraries SymPy, NumPy, and Matplotlib to verify, compute, and visualize solutions. We then compare analytical, symbolic, and numerical solutions, presenting a bifurcation diagram to illustrate the impact of grid spacing on various solution types. Additionally, within numerical solutions, we explore sensitive dependence on initial conditions, commonly referred to as the butterfly effect, as well as chaos. The report concludes with a final section summarizing our findings.

1. Introduction

1.1 Ordinary Differential Equations (ODEs) and the Logistic ODE

An ordinary differential equation, abbreviated as ODE, is an equation consisting of one independent variable and at least one of that variable's derivatives. The order of an ODE is defined by the highest-order derivative present in that ODE.

The logistic ODE is a first-order ODE with one linear and one quadratic term. It can be modeled as:

$$\frac{du}{dt} = \sigma u(1-u), \tag{1}$$

where σ is a parameter.

1.2 Robert May's Work (1976)

Robert May was a theoretical physicist turned biologist. As he started pursuing ecology, he studied the behavior of simple populations.

While testing different parameters for the logistic difference equation, which models the behaviors of populations, May realized that increasing the value of the parameter increased the degree of nonlinearity in his system. Nonlinearity describes a situation where non-proportional relationships between variables lead to amplified changes. By

increasing the degree of nonlinearity in his system, the population changed from steady-state to oscillating to unpredictable. May focused on the boundary between steadiness and oscillation, discovering period doubling, which refers to the phenomenon where populations suddenly oscillate between 2 values, then 4, 8, etc., until the population's behavior becomes apparently random, exhibiting chaos.

To visualize his findings, May created bifurcation diagrams, which reflect the behavior of populations across different values of the parameter. From these diagrams, he was able to find windows of periodic stability within chaos, as the populations oscillated between values of 3, 6, 12, etc., as well as between 7, 14, 28, etc.

In this report, we will explore and recreate the bifurcation diagram of the logistic difference equation, discussing the three states May discovered: steady-state, periodic, and chaotic. We will also explore definitions of chaos and its related concepts.

1.3 Euler's Method and The Forward Finite Difference Method

To derive the logistic difference equation, we can use Euler's method, a type of finite difference method, to "approximate" or discretize the logistic ODE. The finite difference method uses finite differences to approximate derivatives. It is a numerical method that solves both ODEs and PDEs (partial differential equations) through pointwise approximation.

The forward finite difference method is expressed as follows:

$$\frac{du}{dt} = \frac{u_{n+1} - u_n}{\Delta t}$$

- There are two other difference methods, i.e., the backward difference method, where we write $\frac{du}{dt}$ as $\frac{u_n - u_{n-1}}{\Delta t}$, and the central difference method, where we write $\frac{du}{dt}$ as $\frac{u_{n+1} - u_{n-1}}{2\Delta t}$.

Euler's method is one of the forward finite difference methods:

$u_{n+1} = u_n + \Delta t \times u'(t_n, u_n)$ (Euler's Method)

$u_{n+1} = u_n + \Delta t \times \frac{u_{n+1} - u_n}{\Delta t}$ (Applied the forward finite difference method on $u'(t_n, u_n)$)

$u_{n+1} = u_n + u_{n+1} - u_n$

$u_{n+1} = u_{n+1}$

In Section 2.1.2, we will use Euler's method to derive the logistic difference equation from the logistic ODE.

2. Methodology

2.1 Mathematical Equations and Solutions

2.1.1 Analytical Solution of the Logistic ODE

When we set the initial condition (IC) $u(0) = u_0$, the logistic ODE can be written as:

$$(2) \qquad u(t) = \frac{u_0 e^{\sigma t}}{1 - u_0 + u_0 e^{\sigma t}}$$

through the following manipulations:

$$(1) \quad \frac{du}{dt} = \sigma u(1 - u)$$

$$\int \frac{1}{\sigma u(1-u)}\, du = \int dt$$

PFD: $\frac{A}{\sigma u} + \frac{B}{1-u} = \frac{1}{\sigma u(1-u)}$

$$A - Au + B\sigma u = 1$$

$$A = 1, \quad B = \frac{1}{\sigma}$$

$$\int \frac{1}{\sigma} \cdot \frac{1}{u}\, du + \int \frac{1}{\sigma} \cdot \frac{1}{1-u}\, du = \int dt$$

$$\frac{1}{\sigma} \ln|u| - \frac{1}{\sigma} \ln|u - 1| = t + C_1$$

$$\ln\left|\frac{u}{u-1}\right| = \sigma t + C_2 \quad (\text{Note: } C_2 = \sigma C_1)$$

$$\frac{u}{u-1} = e^{\sigma t + C_2}$$

$$\frac{u}{u-1} = C_3 e^{\sigma t} \quad (\text{Note: } C_3 = e^{C_2})$$

$$\text{Plug in } t = 0 \text{ to find } C_3 \text{ with IC } u(0) = u_0\text{:}$$

$$\frac{u_0}{u_0 - 1} = C_3$$

$$1 + \frac{1}{u-1} = C_3 e^{\sigma t}$$

$$u - 1 = \frac{1}{C_3 e^{\sigma t} - 1}$$

$$u = \frac{C_3 e^{\sigma t}}{C_3 e^{\sigma t} - 1}$$

$$\text{From earlier: } C_3 = \frac{u_0}{u_0 - 1}$$

$$u = \frac{\frac{u_0}{u_0 - 1} e^{\sigma t}}{\frac{u_0}{u_0 - 1} e^{\sigma t} - 1}$$

$$(2) \quad u(t) = \frac{u_0 e^{\sigma t}}{1 - u_0 + u_0 e^{\sigma t}}$$

2.1.2 The Logistic Difference Equation

The logistic difference equation is represented by:

$$(3) \qquad u_{n+1} = (1 + \sigma \Delta t) u_n - \sigma \Delta t u_n^2.$$

By using the forward finite difference method, we can manipulate the logistic ODE to solve for the logistic difference equation:

$$(1) \qquad \frac{du}{dt} = \sigma u (1 - u)$$

$$\frac{du}{dt} = \frac{\Delta u}{\Delta t}$$

$$\frac{u_{n+1} - u_n}{\Delta t} = \sigma u_n (1 - u_n)$$

$$u_{n+1} = \Delta t \sigma u_n (1 - u_n) + u_n$$

$$(3) \qquad u_{n+1} = (1 + \sigma \Delta t) u_n - \sigma \Delta t u_n^2$$

If we continue to manipulate Equation 3...

$$u_{n+1} = u_n + \sigma \Delta t u_n - \sigma \Delta t u_n^2$$

$$u_{n+1} = u_n + \Delta t (\sigma u_n - \sigma u_n^2)$$

...we get Euler's method with previous term u_n, step size Δt, and slope of previous point $\sigma u_n - \sigma u_n^2$ (as represented by Equation 1, i.e., the logistic ODE)

2.1.3 The Logistic Map

The logistic map is an equation often represented as $x_{n+1} = r x_n (1 - x_n)$.

The logistic difference equation in this study, $u_{n+1} = (1 + \sigma \Delta t) u_n - \sigma \Delta t u_n^2$, and the logistic map, $x_{n+1} = r x_n (1 - x_n)$, are mathematically identical. They only differ in the absolute values of their parameters and solutions.

We can show the resemblance between the logistic difference equation and logistic map through manipulation.

(3) $u_{n+1} = (1 + \sigma\Delta t)u_n - \sigma\Delta t u_n^2$ (logistic difference equation)

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

$= (\rho Y_n) - (\rho Y_n^2)$

Let $Y_n = \alpha u_n$

$\rightarrow Y_{n+1} = \alpha u_{n+1}$

Plug into Eq. (4):

$\alpha u_{n+1} = \alpha\rho u_n - \alpha^2\rho u_n^2$

(A) $u_{n+1} = \rho u_n - \alpha\rho u_n^2$

From Eq. (3) and (A):

$\rho = 1 + \sigma\Delta t, \qquad \alpha\rho = \sigma\Delta t$

$\therefore \alpha = \frac{\sigma\Delta t}{1+\sigma\Delta t}$

$\therefore Y_n = \alpha u_n = \frac{\sigma\Delta t}{1+\sigma\Delta t} u_n$

2.2 Fundamental Python Programming Concepts

2.2.1 SymPy

Symbolic computing, also called computer algebra, resides at the intersection between computer science and mathematics. While numerical computing approximates solutions by manipulating numerical values, symbolic computing provides exact solutions through symbolic expressions and equations. Thus, the strength of symbolic computing lies in its accurate analytical computations and efficient automation of numerical methods.

By utilizing SymPy, a Python library created for symbolic mathematics, we can easily simplify expressions that include integrals or derivatives. Such expressions are an important aspect of symbolic computing. Since they are also crucial to Calculus 1 and 2, as well as other STEM classes, students can utilize SymPy to verify their answers. Due to its efficiency, SymPy is also a great tool for simplifying complex expressions that would otherwise be very tedious to solve.

2.2.2 NumPy

NumPy is the numerical counterpart to SymPy; that is, it is a numerical computing Python library that approximates solutions by manipulating numerical values. NumPy is based on the `ndarray` object, which stands for an n-dimensional array and can create arrays of any dimension. We will use these arrays to hold the values of the independent and dependent variables we want to plot. NumPy's arrays also support element-wise operations, which means that they efficiently perform operations like addition and multiplication on all alements inside an array. This is much faster than a Python for loop.

2.2.3 Matplotlib

Matplotlib is a Python library for creating visualizations. It has a submodule called pyplot, which provides a high-level, user friendly interface for producing and customizing plots. Pyplot acts like MATLAB.

Below is a code snippet that demonstrates the basic function of pyplot:

In [2]:
```
import matplotlib.pyplot as plt

# create arrays for (t, u) values to plot:
t = [1, 2, 3, 5]
u = [2, 4, 8, 9]

plt.plot(t,u)
plt.show()
```

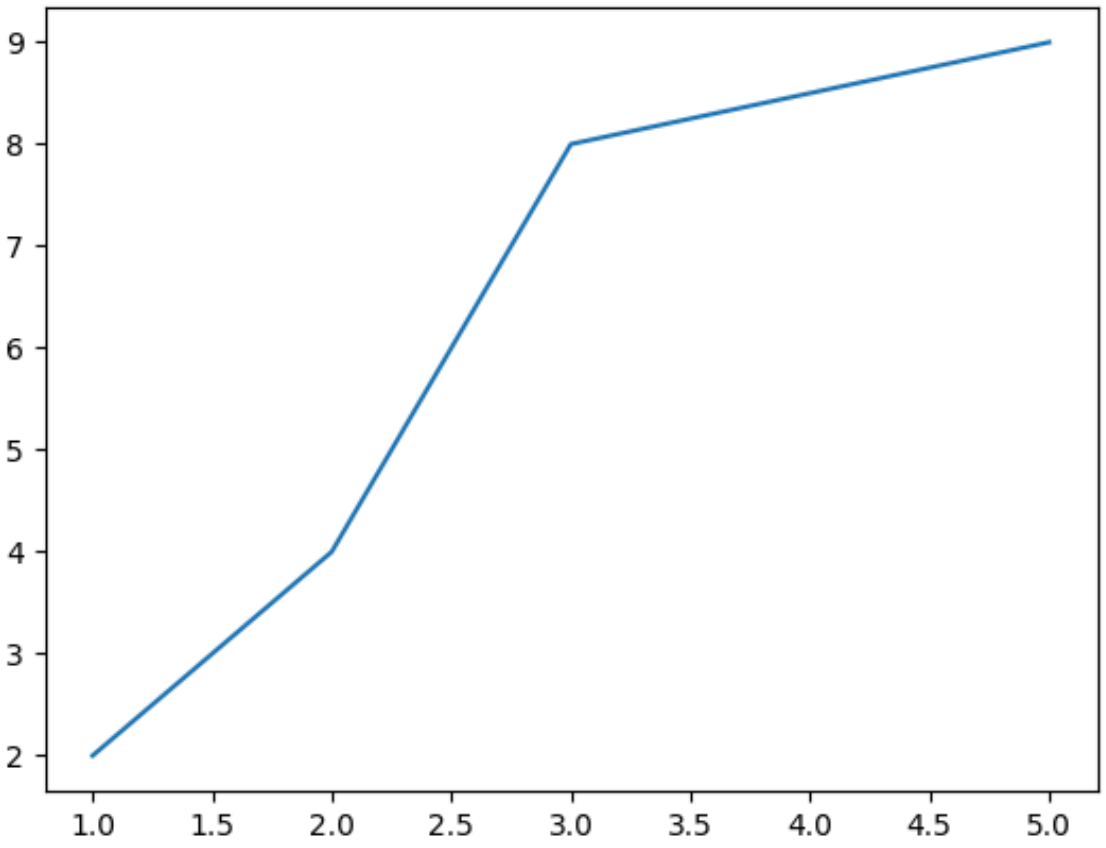

Figure 1. Example of a simple plot made with Matplotlib.

While there are many pyplot functions we can use to customize our graphs, if we do not specify any, pyplot will default to the above configuration, automatically creating the range and frequency of tick marks on the t and u, or x and y, axes.

2.2.4 Graph of the Symbolic Solution of the Logistic ODE

Here is an example of how we can use SymPy, as well as NumPy and Matplotlib, to find and graph the symbolic solution of the logistic ODE.

Import necessary modules

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

Define symbolic variables

```python
u = sp.Function('u')
t = sp.Symbol('t')
u_ = sp.Derivative(u(t), t)
```

Define parameters (domain t & σ)

```python
t_start = 0.
t_end = 10.
dt = 0.1
sigma = 2
```

Set up initial conditions (ICs), solve for u(t) given logistic ODE, isolate solution of u(t)

```python
ics = {u(0):0.001}
ana_eq = sp.dsolve(u_ - sigma * u(t) * (1 - u(t)), u(t), ics=ics)
ana_expr = ana_eq.rhs
```

Translate SymPy expression to a numerical function that can be used with NumPy

```python
# lambdify ana_expr to accept numpy arrays
lam_ana = sp.lambdify(t, ana_expr, modules='numpy')
```

Create NumPy arrays for t and u

```python
t_array = np.arange(t_start, t_end+dt, dt)
u_ana = lam_ana(t_array)
```

Plot all (t, u) and customize graph

```python
plt.figure(figsize=(10,6))
plt.grid(True)
plt.title("Analytical Solution Using SymPy")
plt.xlabel(r'$t$')
plt.ylabel(r'$u_t$')
plt.plot(t_array, u_ana, color="plum")
plt.show()
```

We obtain the analytical solution of the logistic ODE by using `dsolve`, a SymPy function that solves any ODE. Once we obtain the analytical solution, we need to use `lambdify` to translate it to a numerical function that can be used with NumPy. After that, we create two NumPy arrays, one for values of t and one for values of u, and use as inputs for pyplot's `plot` function.

Below is the output of our code. We will revisit this later in Section 3.1.1.

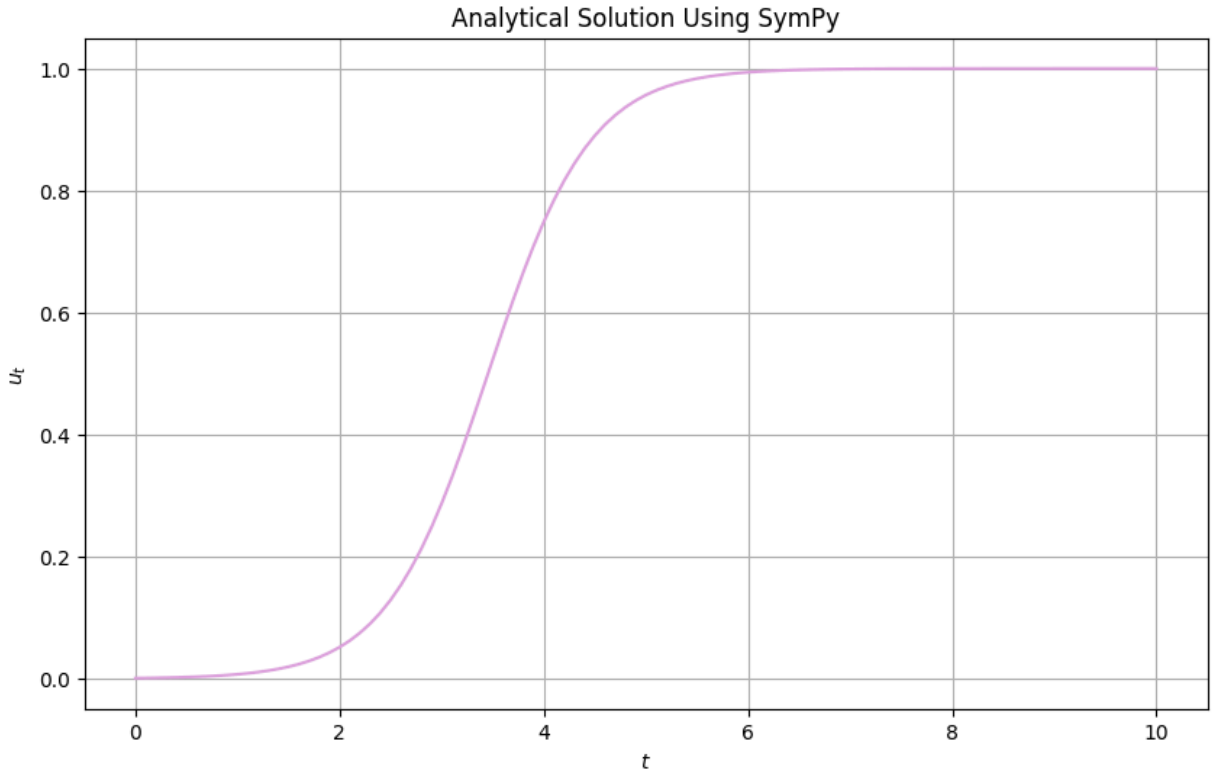

Figure 2. Symbolic solution of Equation (3), the logistic difference equation, obtained through SymPy.

2.2.5 Graph of the Numerical and Analytical Solutions of the Logistic ODE

Using NumPy and Matplotlib, we can also create nine subplots comparing the numerical and analytical solutions of nine different $\sigma\Delta t$ values.

Import necessary modules

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

Define parameters (domain t) & create array for t

```python
t_start = 0
t_end = 1000
dt = 20
num_t = int((t_end-t_start)/dt) + 1  # num sample coordinates
t = np.linspace(t_start, t_end, num_t)
```

Set up initial conditions and values of σΔt

```python
u_0 = 0.01
sigmadt_list = [1.8, 2.0, 2.1,
                2.3, 2.5, 2.6,
                2.7, 3.0, 3.01]
```

Define function to solve for u analytically

```python
def analytical(t, u0, sigma):
    return u0 * np.exp(sigma*t) / (1 - u0 + u0*np.exp(sigma*t))
```

Create lists to hold u values for numerical and analytical solutions for all values of σΔt
Set up arrays to temporarily hold u values for each value of σΔt

```python
u_num_list = []
u_num = np.zeros(num_t)

u_ana_list = []
u_ana = np.zeros(num_t)
```

Iterate through all values of σΔt
Calculate values of u numerically and analytically.
Append the two arrays of u values to their two lists

```python
# sigmadt is r
for sigmadt in sigmadt_list:
    # numerical solution
    u_num[0] = u_0
    for i in range(num_t-1):
        u_num[i+1] = (1+sigmadt)*u_num[i] - sigmadt*u_num[i]**2
    u_num_list.append(u_num.copy())

    # analytical solution
    sigma = sigmadt / dt
    u_ana = analytical(t, u_0, sigma)
    u_ana_list.append(u_ana.copy())
```

Graph numerical vs analytical solutions for all values of σ

```python
plt.figure(figsize=(15, 7))

for i in range(len(sigmadt_list)):
    plt.subplot(3, 3, i+1)
    plt.grid(True)
    plt.title(fr"$\sigma \increment t$={sigmadt_list[i]}")
    plt.xlabel(r'$t$')
    plt.ylabel(r'$u_t$')
    plt.plot(t, u_num_list[i], color="plum")
    plt.plot(t, u_ana_list[i], color="olive", linestyle="dashed")
    plt.legend(["Numerical Solution", "Analytical Solution"])

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

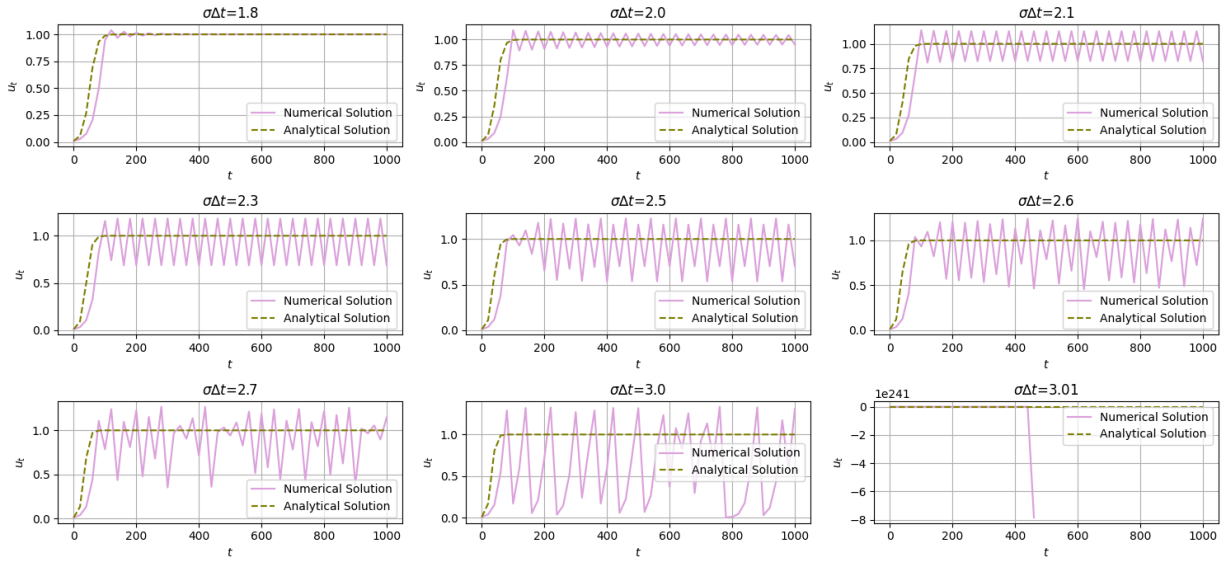

Figure 3. Solutions of Equation (3), the logistic difference equation. The nine subplots display steady-state, periodic, and chaotic behavior, as well as overflow.

We will revisit this figure later in Section 3.1.2.

2.2.6 The Logistic Bifurcation Diagram

The bifurcation diagram of the logistic difference equation is a diagram that provides an overview of the behavior of u across different values of $\sigma\Delta t$. Here is the code to graph it, utilizing NumPy and Matplotlib:

Import necessary modules

Define range of σΔt to test

Number of times (repetitions) to test the end behavior of one σΔt

Create start and end variables for the last 200 numbers to plot

Create figure (graph) & specify its size

For all σΔts in range to test, calculate end behavior values and store them in the results array,

then plot the last 200 numtoplot number of results

Customize graph

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

# sdt = sigma delta t
sdt_start = 1.85
sdt_end = 3.1
dsdt = 0.0001
reps = 1000  # number of repetitions
numtoplot = 200  # number of repetitions to plot
results = np.zeros(reps)  # array to store results of all repetitions

rep_start = reps - numtoplot  # start from the last numtoplot repetitions
rep_end = reps + 1

xsdt = np.ones(numtoplot)  # array of x values to plot

fig = plt.figure(figsize=(12,5))

results[0] = np.random.rand()  # random initial condition on [0, 1)
for sdt in np.arange(sdt_start, sdt_end, dsdt):
    for n in range(reps - 1):
        results[n + 1] = (1+sdt)*results[n] - sdt*results[n]**2

    # plot last numtoplot number of results
    plt.plot(sdt*xsdt, results[rep_start:rep_end], "b.", markersize=0.01)

plt.grid(axis='x', linestyle=(0, (10, 10)), linewidth=1)
plt.xticks(np.arange(sdt_start, sdt_end, 0.05), size=8)
plt.yticks(size=8)
plt.xlabel(r"$\sigma \Delta t$", size=15)
plt.ylabel(r"$u_t$", size=15)
plt.title(r"Bifurcation Diagram of the Logistic Difference Equation", size=18)
plt.savefig("bifurcation.png")
plt.show()
```

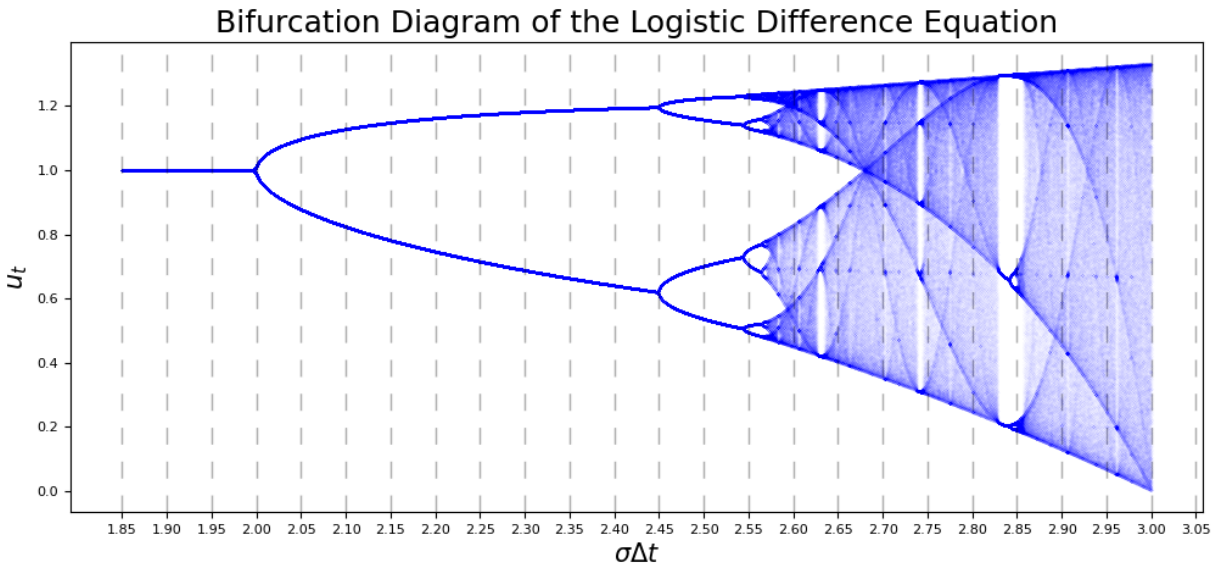

Figure 4. Bifurcation diagram of the logistic difference equation. Demonstrates the behaviors of u across different values of $\sigma\Delta t$.

We will revisit this diagram in Section 3.1.4.

3. Python-Driven Discovery of Key Characteristics

3.1 Comparison of Symbolic, Analytical, and Numerical Analyses

3.1.1 Analytical Solution vs. Symbolic Solution

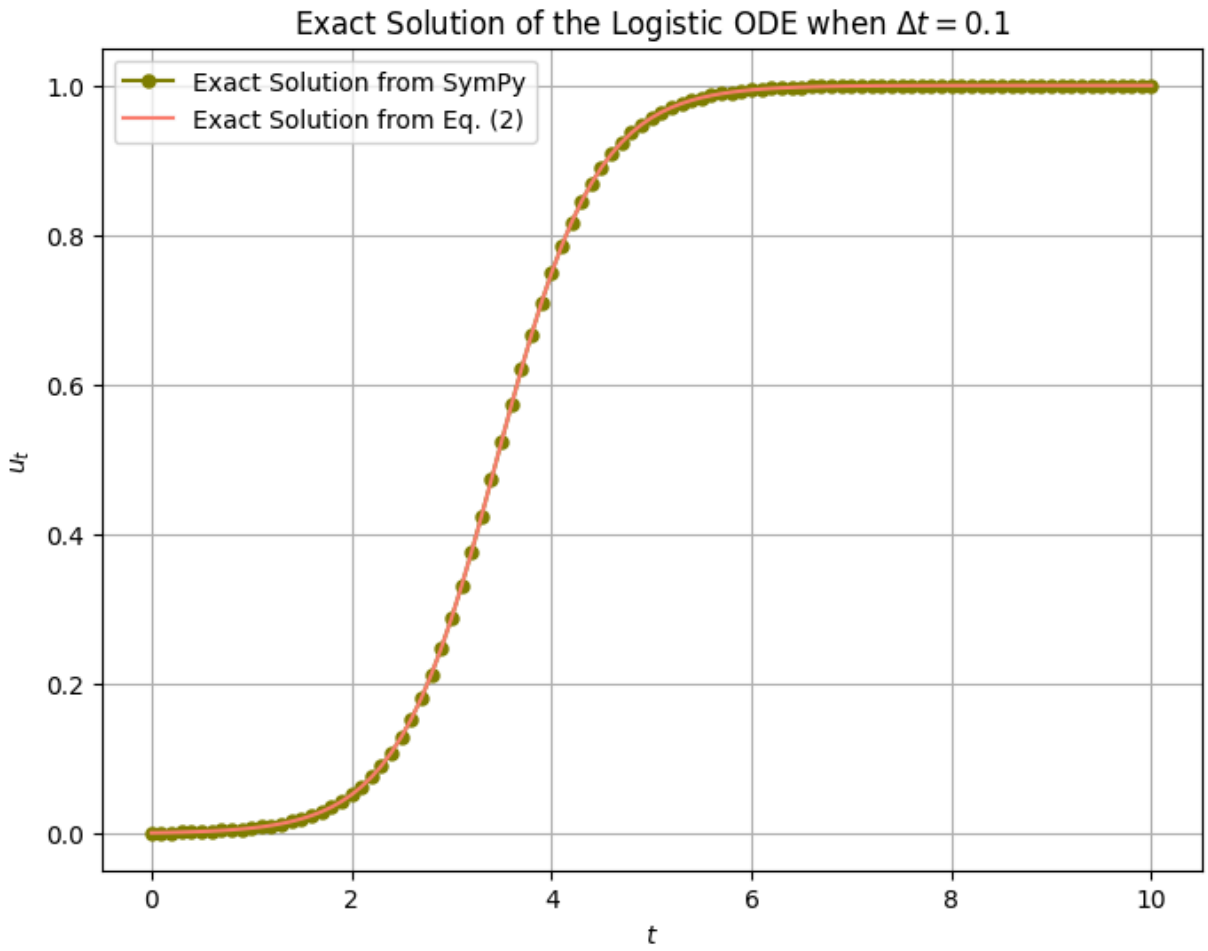

Figure 5. Two exact solutions of the logistic ODE: one symbolic solution from SymPy, and one manually-derived analytical solution.

Here is a graph of the exact solution of the logistic ODE when $\Delta t = 0.1$ Notice that there are two curves that overlap each other. Earlier in Section 2.2.4, I explained the code for the green curve, which utilizes the exact solution of the logistic ODE from SymPy. The

orange curve is the exact solution from Equation 2, which is manually derived in Section 2.1.1.

In [3]:
```
import sympy as sp
from IPython.display import display

u = sp.Function('u')
t = sp.Symbol('t')
u_ = sp.Derivative(u(t), t)
sigma = sp.Symbol('sigma')

ana_eq = sp.dsolve(u_ - sigma * u(t) * (1 - u(t)), u(t))
display(ana_eq)
```

$$u(t) = \frac{1}{C_1 e^{-\sigma t} + 1}$$

Above is some code that shows what the exact solution from SymPy is. It is the same code we used to symbolically graph the logistic ODE using SymPy.

Here is Equation 2 for reference:

$$u(t) = \frac{u_0 e^{\sigma t}}{1 - u_0 + u_0 e^{\sigma t}}$$

While these two equations may look different, they are actually the equivalent to each other.

We can manipulate the equation from SymPy to show that it is equal to Equation 2:

Use $u(0) = u_0$ to find C_1:

$$u_0 = \frac{1}{C_1 + 1}$$

$$C_1 = \frac{1}{u_0} - 1$$

$$C_1 = \frac{1 - u_0}{u_0}$$

Plug back into original equation:

$$u(t) = \frac{1}{\frac{1-u_0}{u_0} e^{-\sigma t} + 1}$$

$$u(t) = \frac{u_0}{e^{-\sigma t} - u_0 e^{-\sigma t} + u_0}$$

$$u(t) = \frac{u_0 e^{\sigma t}}{1 - u_0 + u_0 e^{\sigma t}}$$

This is Equation 2. We have thus proved that the analytical solution we derived by hand, i.e., Equation 2, is equivalent to the symbolic solution we found using SymPy.

3.1.2 Numerical Solutions vs. Analytical Solutions

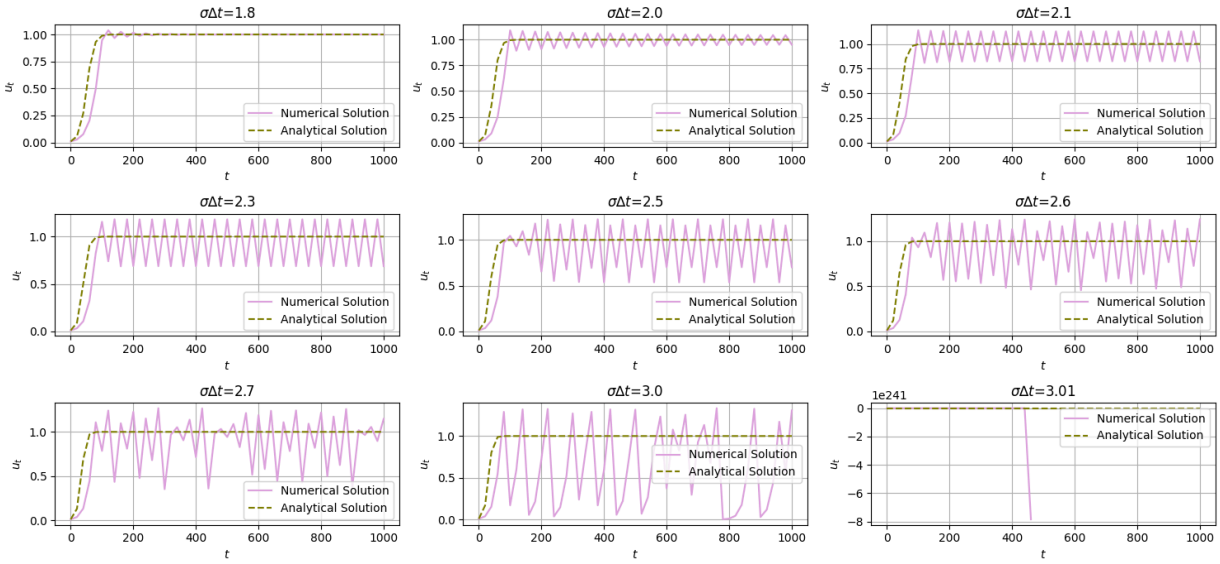

Figure 3. Solutions of Equation (3), the logistic difference equation. The nine subplots display steady-state, periodic, and chaotic behavior, as well as overflow.

Here is the graph we created in Section 2.2.5, which compares the behaviors of the numerical and analytical solutions of nine different values of $\sigma\Delta t$.

- In the first two graphs, where $\sigma\Delta t = 1.8$ and 2.0, the numerical solutions converge to 1. In other words, they approach a "steady-state," making them steady-state solutions.
- In the third and fourth graphs, where $\sigma\Delta t = 2.1$ and 2.3, the numerical solutions appear to be of period 2, since they oscillate between two values.
- In the fifth graph, where $\sigma\Delta t = 2.5$, the numerical solution appears to be of period 4, as it oscillates between four values.
- In the sixth, seventh, and eighth graph, where $\sigma\Delta t = 2.6$, 2.7, and 3.0, the graphs exhibit chaotic behavior, as their populations jump between apparently random values.
- The last graph, where $\sigma\Delta t = 3.01$, demonstrates what happens when the population values overflow, i.e., grow without bound.

3.1.3 Impact of Grid Spacing (i.e., Control Parameter Δt)

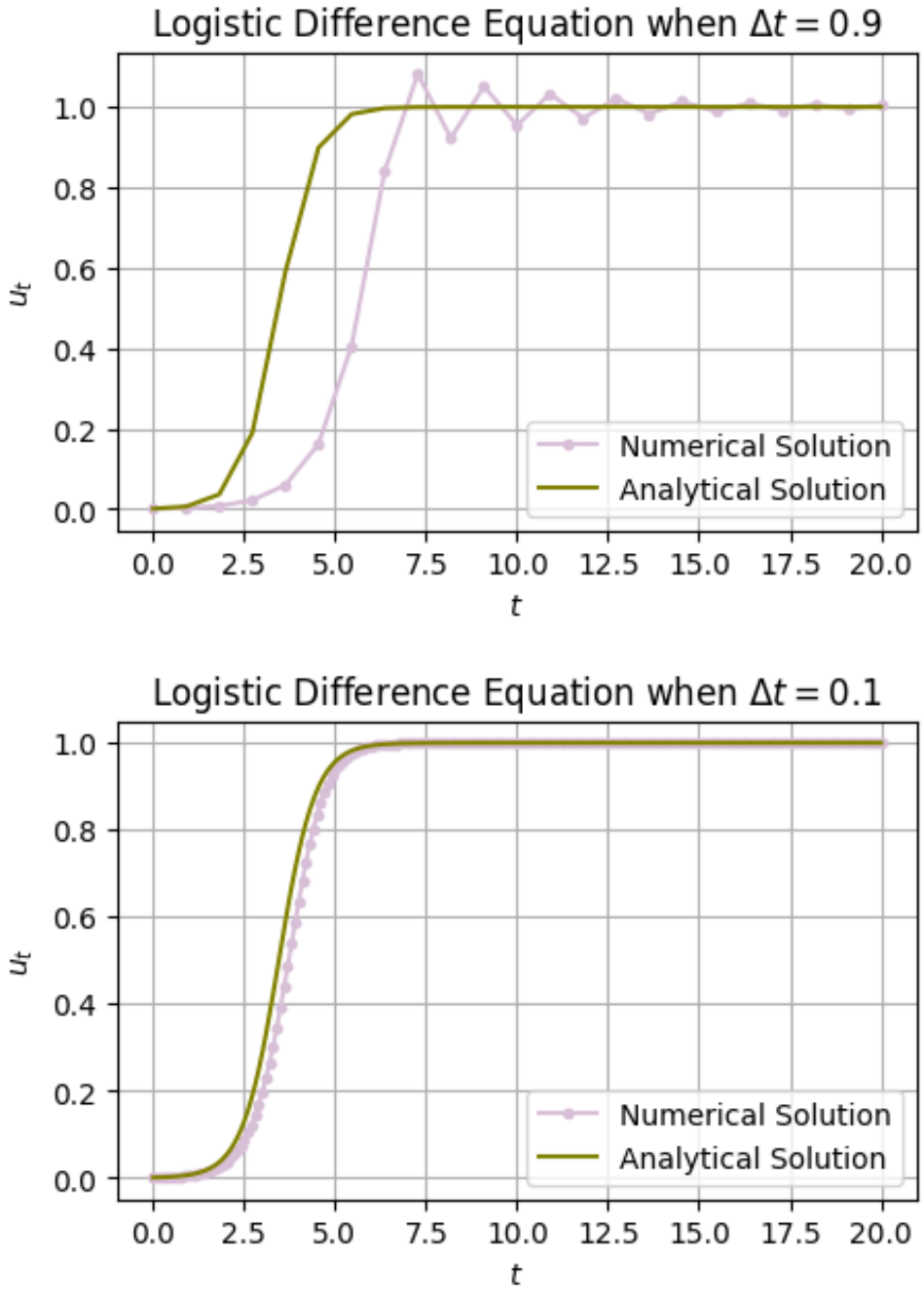

Figure 6. Comparison of the numerical and analytical solutions of the logistic difference equation under two different values of Δt.

When we keep σ and the initial conditions the same but decrease Δt, the accuracy of the numerical solution increases. In both of the above graphs, $\sigma = 2$ and $u_0 = 0.001$.

The numerical solution of the second graph, where $\Delta t = 0.1$, is much closer to the analytical solution compared to the numerical solution of the first graph, where $\Delta t = 0.9$, making it more accurate.

3.1.4 The Bifurcation Diagram of the Logistic Difference Equation

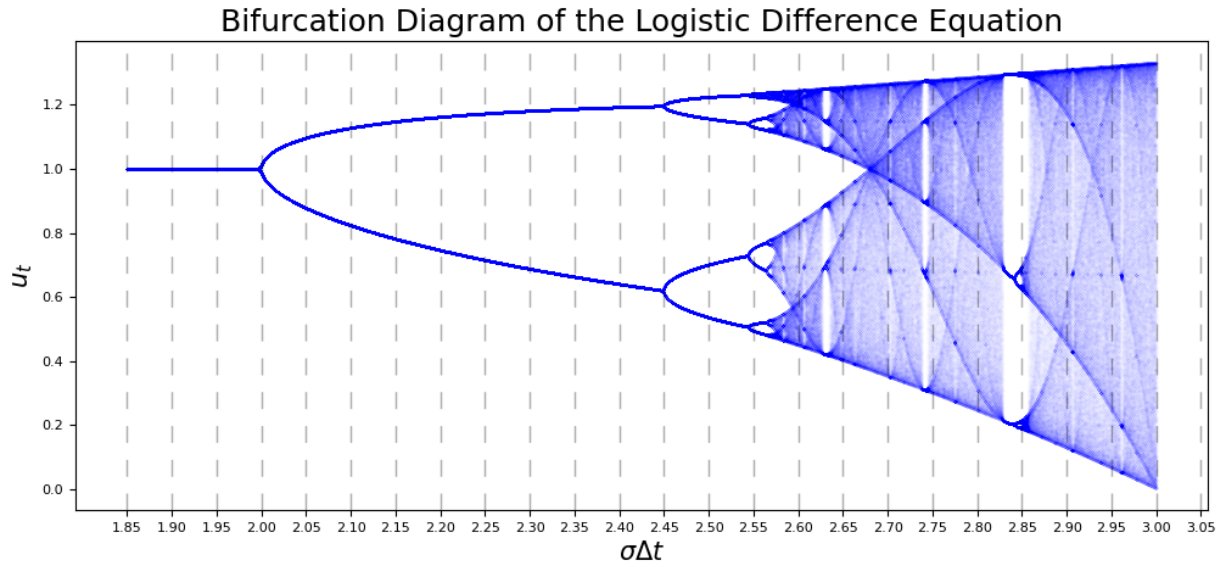

Figure 4. Bifurcation diagram of the logistic difference equation. Demonstrates the behaviors of u across different values of $\sigma\Delta t$.

Above is the bifurcation diagram of the logistic difference equation. I explained the code for this in Section 2.2.6.

A bifurcation occurs when small changes in the parameter (i.e., $\sigma\Delta t$) cause a "significant" change in the structure of the solutions of a system.

This bifurcation diagram reflects the behavior of the numerical solutions we previously looked at:

- u converges to 1 when $\sigma\Delta t$ is less than around 2
- u is in period 2 when $\sigma\Delta t$ is between around 2 and 2.45
- u is in period 4 when $\sigma\Delta t$ is between around 2.45 and before 2.55
- The rest of the graph demonstrates chaos, represented by shades of blue, with occasional periods of 3, 6, etc.

3.2 Examination of SDIC, the Butterfly Effect, and Chaos

Steven Strogatz defines chaos as “aperiodic long-termed behavior in a deterministic system that exhibits sensitive dependence on initial conditions.” Sensitive dependence of solutions on initial conditions, abbreviated as SDIC, is the most important factor that defines chaos and is commonly known as the butterfly effect. The "butterfly effect," coined by Edward Lorenz, who proposed that the wing flaps of a small butterfly could

cause a tornado, states that small changes in initial conditions can lead to immensely different outcomes.

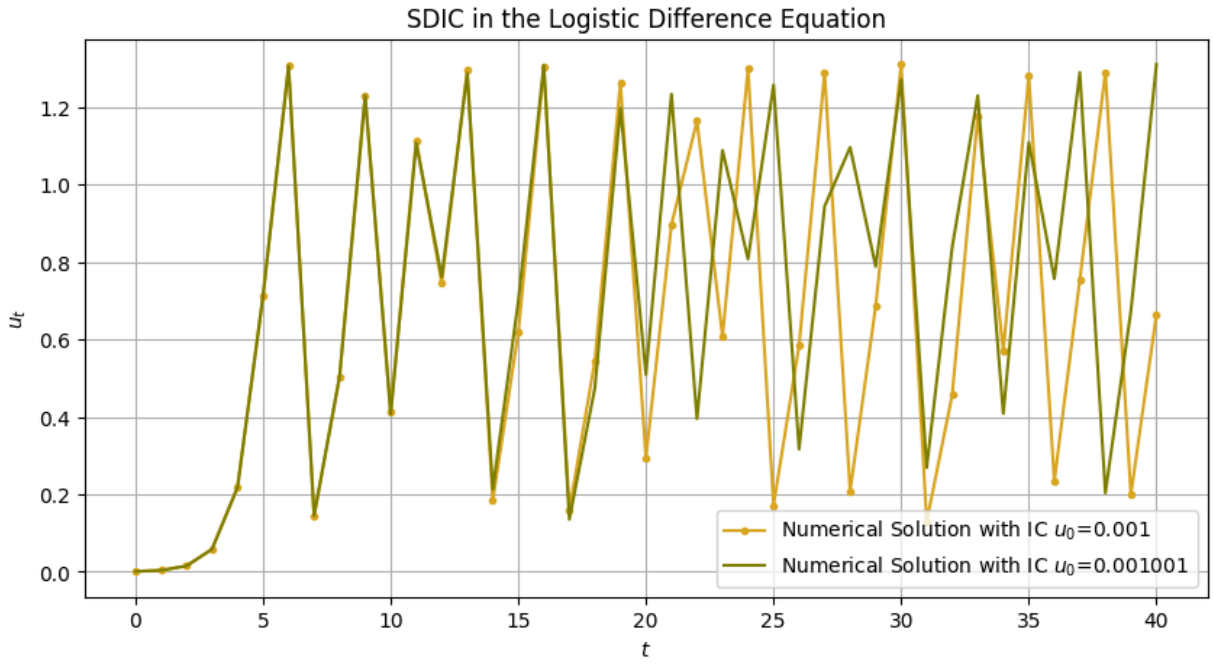

Figure 6. Time evolution of two initially nearby trajectories in the logistic difference equation, illustrating sensitive dependence on initial conditions (SDIC) and the butterfly effect.

SDIC can be shown through control runs and parallel runs. In the diagram above, the control run, with $u_0 = 0.001$, is in green, while the parallel run, with $u_0 = 0.001 + \epsilon, \epsilon = 10^{-6}$, is in gold.

Such a small difference of $\epsilon = 10^{-6}$ between the control and parallel runs eventually leads to completely different behaviors of u_t. This difference is especially noticeable after $t = 20$, when the control and parallel runs diverge.

4. Conclusions

In this report, we introduced the logistic ODE and discretized it to derive the logistic difference equation, collectively referred to as the logistic equations. Although the logistic difference equation in this study and the logistic map are mathematically identical, they differ in the absolute values of their parameters and solutions. Using different methods, we then explored various forms of solutions for the logistic equations, including analytical solutions, which are exact solutions derived manually; symbolic solutions, which are exact solutions obtained using SymPy; and numerical solutions, which are approximate solutions obtained by solving the logistic difference equation

using a finite difference scheme. Additionally, we presented various types of numerical solutions, which vary depending on the values of the control parameter ($\sigma\Delta t$). These numerical solutions can be steady-state, periodic, or chaotic (apparently random). Finally, we discussed sensitive dependence on initial conditions, the butterfly effect, and chaos. To support our efforts, we utilized modern Python libraries such as SymPy, NumPy, and Matplotlib.

Author Contributions

Olivia Zhou: Formal analysis, Derivations and verifications, Software, Visualization, Investigation, Writing.

Bo-Wen Shen: Conceptualization, Methodology, Derivations, Software, Investigation, Writing – review.

References

Shen, B.- W., 2024: Revisiting Lorenz's Error Growth Models: Insights. Encyclopedia 2024, 4(3), 1134-1146; https://doi.org/10.3390/encyclopedia4030073

Gleick, J. *Chaos: Making a New Science*; Viking Penguin: New York, NY, USA, 1987; pp. 57-80.