

# Into Chaos Theory

By Ana F. Dalipi

Ecologists have a long history of using mathematical models to make predictions about population biology. Predictions about a population increasing or decreasing, or epidemics spreading in host populations, can be made by turning populations into dynamical systems. One way to do this is to use differential equations. Differential equations are good for describing processes that change or evolve over time, however, they are less well suited for populations that change in discrete steps. Another alternative is difference equations that are often helpful for describing processes that jump from state to state. For example, many insects, like orchard pests, have a single annual breeding season. In order to predict next season's population, we need to know the size of the population for this year. If we assume that next year's population depends only on this year's population, then next year's population is a function of this year's population, the following year's population is the function of the preceding year's, and so on.

The question then is, what is the long term trajectory of the population? One might expect the population to settle down over time to a fixed stable level. Many times that is the case, however, even if the function involved is of a simple mathematical form, the population levels from year to year can demonstrate quite bizarre behaviors sometimes.

There are many functions that can be used to model a population of seasonally breeding organisms whose generations do not overlap[1]. Perhaps the easiest approach would be the Malthusian scheme for population growth which increases the population by a certain percentage each year[2]. If  $r$  represents the rate of population growth, then:

$$x_{n+1} = rx_n$$

This is a linear equation and easy to solve, however, it lacks the flexibility to adequately model actual populations. For example, if  $r < 1$ , the population becomes extinct, and if  $r > 1$  the population will grow exponentially, however, no real population can sustain exponential growth indefinitely. In the real world ecologists have to account for many factors like competition between animals or the scarcity of food. For example, a small population of mice can increase rapidly, however, as there are more members in the population there is less food for everyone. This will inevitably cause the population of mice to level off. Thus ecologists realised that they needed a way to model population growth so that growth is restrained if the population becomes large. Mathematically, the simplest way to archive this is to assume that growth rate  $r$  itself decreases as population grows. If we arbitrarily set a cap of 1 for the maximum population then we could assume that the growth rate is proportional to  $1 - x$ . This would lead to a new function:

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

As  $x$  becomes closer to 1,  $(1 - x)$  is going to bring it back down and make sure the growth is within bounds. This is known as the logistic equation. Although this equation is simple, something that the ecologists favor, it tends to produce some rather erratic behaviours[2].

As  $r < 1$ ,  $x$  tends to decrease to 0 like before. If  $r$  is between 1 and 3,  $x$  does approach a limit given by  $\frac{r-1}{r}$ . Yet as  $r \geq 3$ , the numbers begin to misbehave. They do not grow without restraint, but they do not

lead to a steady level either. Instead they keep oscillating around, refusing to settle down. At the time, ecologists shrugged it off as and assumed there might be some hidden equilibrium that the populations kept bouncing around or some error in the calculations. Robert May would not agree.

Although Robert May was a biologist, he had an unusual amount of mathematics in his background and was interested in the abstract problems of stability. He was particularly interested in the logistic equation to the point that he was starting to get obsessed. He wanted to understand what happened when a population's rate of growth passed a critical point, specifically what happened between steadiness and oscillation. May tried different values for the rate of population growth  $r$  and found that he could change dramatically the behavior of the system. When the parameter was low, the model reached a steady state. When it was high, it would break apart and act unpredictably. May increased  $r$  slowly to gather as much data as he could using a bifurcation graph. A bifurcation graph shows how a change in one parameter can cause drastic change in the ultimate behaviour of the entire system [2].

In [1]:

```
import pylab as plt #plots
import numpy as np #mathematical functions
%matplotlib inline
```

In [2]:

```
'''
CREDIT:https://ipython-books.github.io/121-plotting-the-bifurcation-diagram-of-a-chaotic-dynamical-
system/
'''

def logistic(r, x):
    return r * x * (1 - x) #the logistic equation

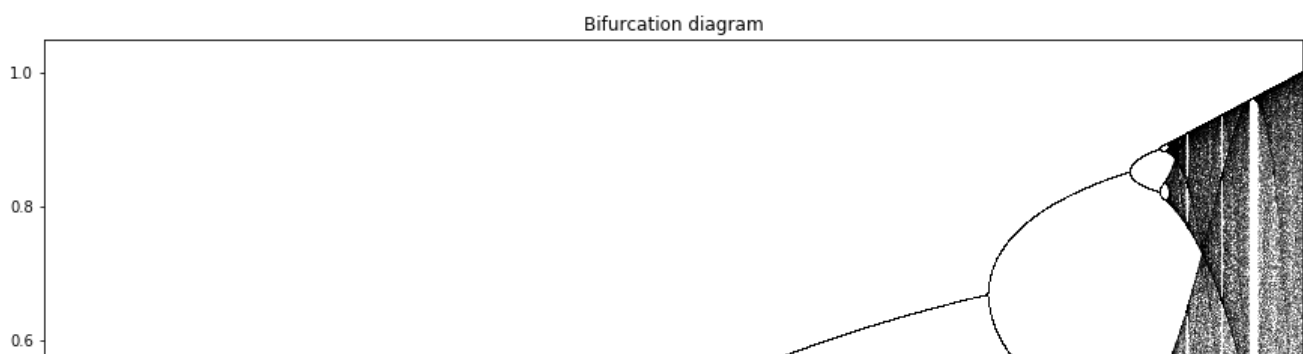
n=10000#define n
r = np.linspace(0, 4.0, n)#define r
iterations = 1000
last = 100
x = 1e-5 * np.ones(n)#define x

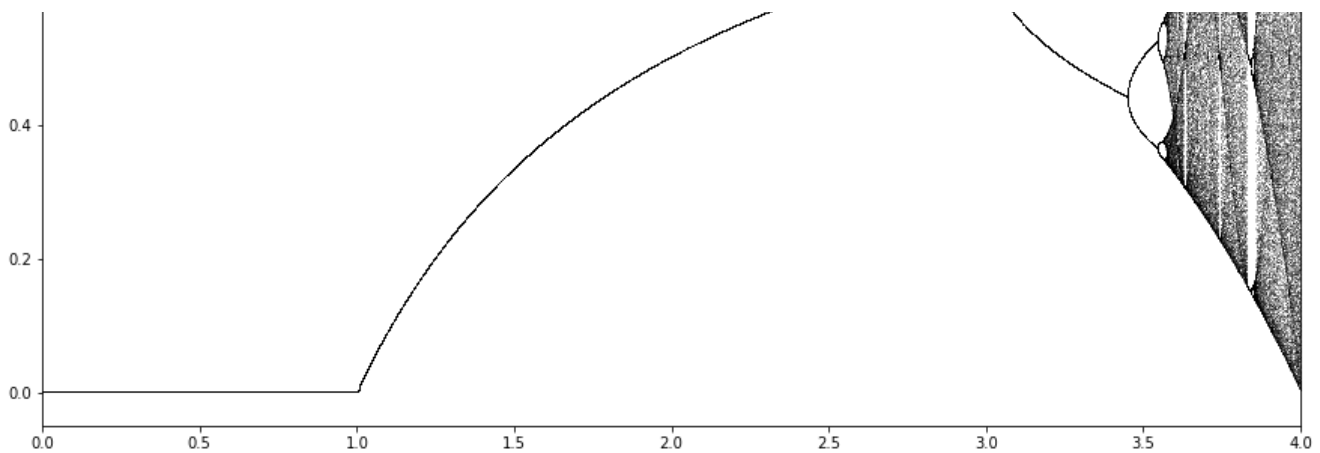
fig, ax1 = plt.subplots( 1, figsize=(15, 9),sharex=True)#figure parameters

for i in range(iterations):#iterate
    x = logistic(r, x)
    # the bifurcation diagram.
    if i >= (iterations - last):
        ax1.plot(r, x, 'k', alpha=.25)
ax1.set_xlim(0, 4)
ax1.set_title("Bifurcation diagram")
```

Out[2]:

Text(0.5, 1.0, 'Bifurcation diagram')





For this graph the parameter  $r$  was represented on the  $x$ -axis, increasing from left to right, while the population was plotted on the  $y$ -axis. He noticed that as  $r$  started increasing, the final population also increased, creating a line that rose from left to right on the graph. Between 1 and 3, the system leveled into a stable population. When  $r$  passed 3, May noticed the line breaking into two and continuing to alternate between these values. As he raised the parameter more, it would continue to split again and the cycle doubled again. This process continued on, going faster each time, from 2, 4, 6, 8, 16, 32 and it suddenly broke off. Still there were some odd periods like 3 and 7 in the middle of this exotic behaviour. Then the period doubling started again only this time faster[2].

It took a long time for May to see the whole picture, yet the fragments that he could notice were shocking. He realized that in the real world ecologists were only looking at a vertical slice of what was actually occurring and it all depended on the initial parameters that they chose. They did not realize that that with some slight changes in the parameter, the system could produce an entirely different pattern. James Yorke described this behavior in his paper “Period Three Implies Chaos”. According to Yorke, in any one dimensional system, if a regular cycle of period 3 appears then the system will also display regular cycles of every other length, as well as completely chaotic cycles[2].

Before Robert May and James Yorke, there were already other scientists that were realizing that there was a sensitive dependence on initial conditions for certain non-linear dynamical systems[2]. Edward Lorenz, titled as the father of Chaos Theory, first noticed this strange phenomenon in the early 1960s. As a MIT meteorology professor, Lorenz was convinced that he could use mainframe computers to make accurate weather forecasts. It is common knowledge that in the short term weather can be determined by factors like temperature, pressure, and wind velocity. So Lorenz believed that data measured by a computer simulated weather using 12 simple, but nonlinear equations, could be used to predict the weather conditions for the future [5]. His goal was to invent the ultimate weather predictor [6]. However, one day, Lorenz made a decision that would change the course of his study and bring attention to a new upcoming field of mathematics. Instead of inputting the same 6-digit number as before, he put only the first three digits into the system. According to the school of thought at the time, this action would only produce minimal changes in the output, so Lorenz did not think much of this decision. However, once he looked at the graph produced, he realized that it predicted an entirely different forecast. Lorenz’s equations were a simple set of equations, yet a small change in its initial conditions yielded a radically different behavior [5]. This would later be known as the ‘Butterfly effect’: the extreme sensitivity to initial conditions means that the flapping of a butterfly’s wings over the Amazon could cause a tornado in Texas [5].

Chaos Theory focuses on the unpredictable and unexpected. Unlike most sciences that deal with predictable

phenomena, Chaos Theory deals with non-linear dynamical systems that cannot be predicted like the weather or, as Robert May investigated, the logistic equation.

## Lorenz Attractors and Feigenbaum's Constant

Lorenz was also responsible for identifying the key mechanisms of Chaos Theory. He decided to investigate more about the strange behavior he had encountered by simplifying the 12 equations he worked with into 3 nonlinear equations inspired by the convection process in fluids. When he graphed his data along several axes, he noticed the strange property that iterating any two nearby points resulted in their ultimate separation [5]. In Lorenz's model, when the rising heat of his system pushed the fluid forward, the trajectory would stay in the right and if the motion stopped and reversed itself, the trajectory would swing to the left [2]. The space between them would grow larger with each iteration until they would be in completely different areas of the cloud of information. When points were off the cloud, once iterated, they would quickly approach it. This "cloud" is called a strange attractor, or the Lorenz attractor [5]. Strange attractors can be found when the system is somehow oddly constrained in a finite space, yet never settles into a fixed point or a steady level. Instead it is nonperiodic, jumping around different values without ever repeating a value twice.[2]

In [9]:

```
'''
CREDIT:https://matplotlib.org/gallery/mplot3d/lorenz_attractor.html
'''

import matplotlib.pyplot as plt
# This import registers the 3D projection, but is otherwise unused.
from mpl_toolkits.mplot3d import Axes3D # noqa: F401 unused import

def lorenz(x, y, z, s=10, r=28, b=2.667):
    '''
    Given:
        x, y, z: a point of interest in three dimensional space
        s, r, b: parameters defining the lorenz attractor
    Returns:
        x_dot, y_dot, z_dot: values of the lorenz attractor's partial
                             derivatives at the point x, y, z
    '''
    x_dot = s*(y - x) #Lorenz's 3 nonlinear equations
    y_dot = r*x - y - x*z
    z_dot = x*y - b*z
    return x_dot, y_dot, z_dot

dt = 0.01
num_steps = 10000

# Need one more for the initial values
xs = np.empty((num_steps + 1,))
ys = np.empty((num_steps + 1,))
zs = np.empty((num_steps + 1,))

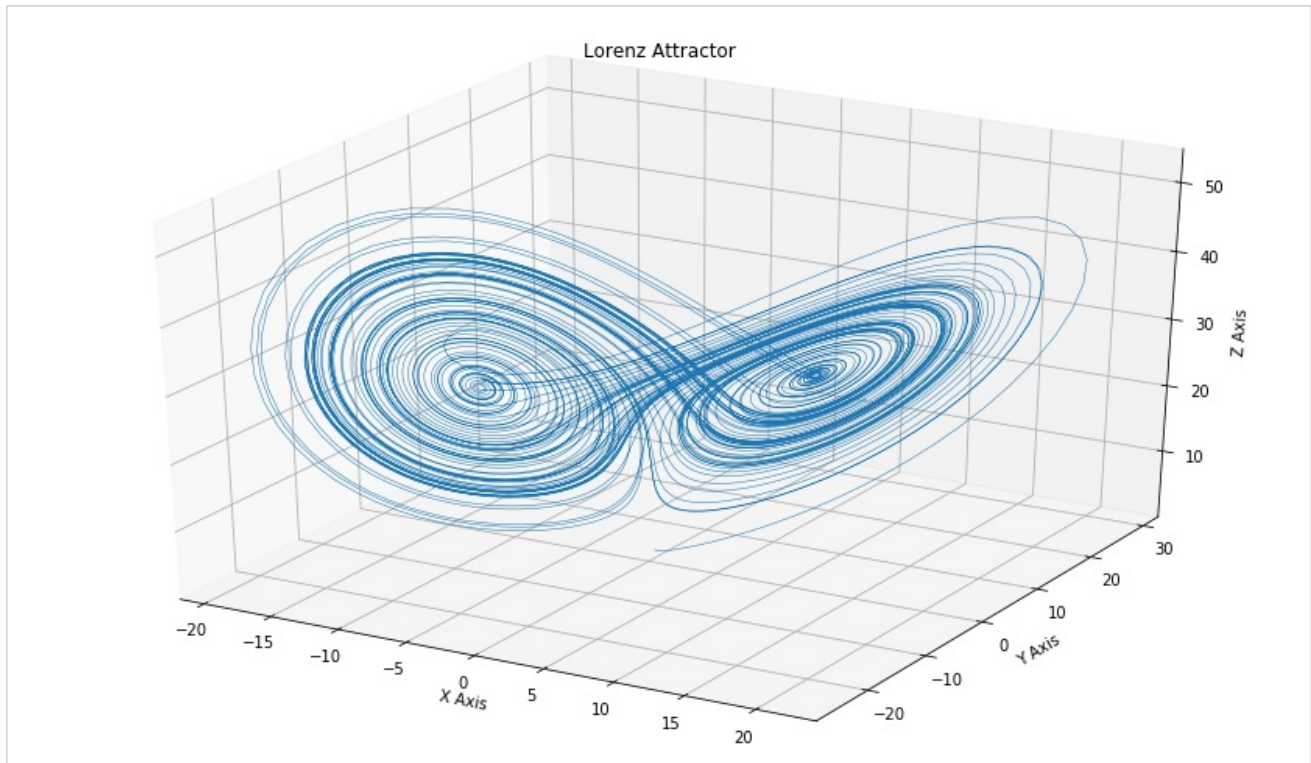
# Set initial values
xs[0], ys[0], zs[0] = (0., 1., 1.05)

# Step through "time", calculating the partial derivatives at the current point
# and using them to estimate the next point
for i in range(num_steps):
    x_dot, y_dot, z_dot = lorenz(xs[i], ys[i], zs[i])
    xs[i + 1] = xs[i] + (x_dot * dt)
    ys[i + 1] = ys[i] + (y_dot * dt)
    zs[i + 1] = zs[i] + (z_dot * dt)
```

```
# Plot
fig = plt.figure(figsize=(15, 9)) #figure parameters
ax = fig.gca(projection='3d')

ax.plot(xs, ys, zs, lw=0.5)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
ax.set_title("Lorenz Attractor")

plt.show()
```



Mitchell Jay Feigenbaum wanted to concentrate on the boundary region between chaos and order. He observed the period doubling in the logistic map that was splitting two-cycles into four-cycles and so on, and was fascinated by this pattern. Only using his calculator and paper, he started computing the exact values of each period doubling and writing them down. Feigenbaum focused on the attractors, fixed points that attract all others, which would split in two every period doubling. As the parameter rose, these values seemed to move further from each-other. Soon he started observing a clear pattern: the numbers were converging geometrically [2]. That meant that the period doubling was occurring at a constant rate. He calculated the ratio of convergence to be 4.669. Feigenbaum knew that geometric convergence meant that something was scaling which meant that some value was being preserved as everything else changed [2]. In order to understand more about this phenomenon, he decided to try other functions and was surprised to learn that the ratio of converge was the same. Eventually Feigenbaum calculated the constant to be 4.6692016090. This number would be very helpful as now Feigenbaum could predict when the period doubling would occur as well as the precise values of the attractors.

It is possible to graph by plotting the input  $x_n$  in the  $y$ -axis and the output,  $x_{n+1}$ , on the  $x$ -axis. To represent long-term behaviour of the system, Feigenbaum drew a trajectory that started with an arbitrary  $x$  on the  $x$ -axis. Since each output on the  $y$ -axis was entered back into the same function as a new input, the trajectory would bounce off the line where  $x=y$ . The behavior depended a lot on the degree of nonlinearity of the function. If it was too small, it

would eventually lead to extinction. Increasing the degree would produce a steady state equilibrium, however, after a specific point, there would be an oscillator with period two and the trajectory would refuse to settle down.

## Visualization Through Technology

On Fall 2018, I participated in a team research project where we investigated the logistic model studied by Robert May and its transition points by using different computational methods in Senior Capstone Experience MATH-406X. I became responsible for building interactive graphs using Python. At first, I was not sure how to tackle this project. Utilizing technology has been an integral part of Chaos Theory and creating another bifurcation graph would not explain a lot. My goal was to use technology in a way where we could explore the logistic map from another perspective.

Initially our group was only looking at bits and pieces of what was going on. We would graph the logistic equation with a particular  $x_n$  and  $r$  value and look at the output. As we were exploring different iterations of  $x_n$ , the idea of combining all the movements of the graph came to my mind. I wanted us to see how changing the  $r$  parameter affected the output of the graph in its entirety, and where it was exactly that the graph started its period doubling. We could just try inputting different values of  $r$ , however that would have been a slow and inefficient process. Using Python 3 in a Jupyter Notebook, I made a slide widget so we could slowly move  $r$  and watch how the  $x_n$  changed. Widgets are Python objects that can be used to build interactive Graphical User Interfaces. I used the `widgets.FloatSlider` to define  $r$  as a slider and set parameters for it from 0 to 4.0. Then I used a function to iterate through the logistic equation. The logistic equation was separated for speed. The first part  $z=x_{n-1}$  acquires the previous  $x_{n-1}$  value and then in the second part  $z$  is put into the equation. In the end, we got not just a fragment of the map but the whole picture of the  $x_n$  values in the long term as  $r$  slowly changed. As Robert May noted, between 1 and 3, the system appears to be stable and the output increases only a little. However once  $r$  gets closer to 3, the line breaks into two and continues to alternate between different values until the whole graph falls off when  $r$  reaches 4.

In [2]:

```
import ipywidgets as widgets
from IPython.display import display
import pylab as plt #plots
import numpy as np #mathematical functions
```

In [3]:

```
r=widgets.FloatSlider(min=0,max=4,value=0,description='Constant:')
n=1000
x=np.arange(0.5,200)#this makes it an array
x[0]=0.5

plt.rcParams['figure.figsize'] = [20, 9]#figure parameters

def update_plot(r):

    for n in range(1,200):
        z = x[n - 1];#z is previous x
        x[n] = r*z*(1 - z);#faster when you separate z

    plt.figure(figsize=(20,7))
    plt.plot(x,'ro')
    #plt.ylim(180,200)#adjust y-axis
```


```
#plt.xlim(100,200)#slower
plt.xlim(1,20)#faster
plt.ylim(-1,1)
```

```
widgets.interactive(update_plot,r=r)
```

In [4]:

```
from IPython.display import YouTubeVideo
# the video of the changes in the Logistic equation as we slowly change ther value.
# Video credit: Ana F. Dalipi
YouTubeVideo('PDBGLoGilqs',width=900,height=500)
```

Out[4]:



We still had more work to do. My next goal was to make an interactive Bifurcation graph. Using the Bokeh Library for interactive visualization I could zoom into the graph and gather the data on the points of interests. This would also give me a better view of what exactly is going on in the regions where the cycle splits and doubles. In theory all I had to do was use the same code for the bifurcation graph, add some code from Bokeh, and I would be able to use the zooming widget. Unfortunately, this part was difficult to implement and I kept getting only half of the graph, specifically I would get either the upper halves or the lower halves of the cycles. It took many tries until I was finally able to combine the graph and I finally had my interactive bifurcation graph. The graph can zoom to the  $10^{\text{th}}$  decimal place.

In [5]:

```
import pandas as pd

from bokeh.models import ColumnDataSource
from bokeh.models.tools import HoverTool
from bokeh.models.widgets import DataTable, DateFormatter, TableColumn
from bokeh.plotting import figure, output_notebook, show
from bokeh.models import HoverTool, BoxZoomTool, PanTool, WheelZoomTool
```

In [6]:

```
output_notebook()
```

```
output_notebook()
```

Loading BokehJS ...

In [7]:

```
n=10000#define n
r = np.linspace(0, 4.0, n)#define r
iterations = 1000
last = 10
x = 1e-5 * np.ones(n)#define x
df = {}
p = figure(plot_width=900, plot_height=800,x_range=(2.5,4))#figure parameters

for i in range(iterations):#iterate
    x = r*x*(1 - x);#function
    data={'x':x,'r':r}
    df[i]= pd.DataFrame(data)#prepare data
    if i >= (iterations - last):
        hover = HoverTool(
            tooltips= [
                ('x', '@x'),
                ('r', '@r')]
            p.circle(x='r',y='x',color="purple", size=5, source=ColumnDataSource(df[i]))#figure
parameters

p.add_tools(hover)
show(p)
```

Still I knew I could do more and I was bothered by how much data there was and how overwhelming it felt. It would be so much better if I could simply gather the location of all the points on the graph just by hovering over them. This was my most challenging work and it took some understanding of databases and the Pandas library. It is a data analysis tool which was necessary for the hover tool to work. I was able to store all the values of the  $x_n$  and  $r$  in the bifurcation graph so the Hover tool could work correctly.

I was able to produce the same interactive graph with other functions as well. As Feigenbaum realized, even for different functions, the graph seemed to produce the same results. I also tried the same code with functions  $x_{n+1}=r*\sin(\pi * x_n)$  and  $x_{n+1}=r*\cos(\pi * x_n)$  and the output was very similar.

In [8]:

```
n=1000#define n
r = np.linspace(0, 4.0, n)#define r
iterations = 10000
last = 10
x = 1e-5 * np.ones(n)
df = {}
p = figure(plot_width=900, plot_height=800,x_range=(0,4))#figure parameters

for i in range(iterations):#iterate
    x=r*np.sin(np.pi * x)#function
    data={'x':x,'r':r}
    df[i]= pd.DataFrame(data)#prepare data
    if i >= (iterations - last):
        hover = HoverTool(
            tooltips= [
                ('x', '@x'),
                ('r', '@r')]
            p.circle(x='r',y='x',color="purple", size=5, source=ColumnDataSource(df[i]))
            #print df[i]
```



```
p.add_tools(hover)
show(p)
```

In [9]:

```
n=1000
r = np.linspace(0, 4.0, n)
iterations = 10000
last = 10
x = 1e-5 * np.ones(n)
df = {}
p = figure(plot_width=900, plot_height=800, x_range=(0,4))

for i in range(iterations):
    #columns = [TableColumn(field="x", title="X values")]
    x=r*np.cos(np.pi * x)
    data={'x':x, 'r':r}
    df[i]= pd.DataFrame(data)
    if i >= (iterations - last):
        hover = HoverTool(
            tooltips= [
                ('x', '@x'),
                ('r', '@r')]
        )
        #p.circle(x='r',y='x',color="purple", size=5, source=ColumnDataSource(df))
        p.circle(x='r',y='x',color="purple", size=5, source=ColumnDataSource(df[i]))
        #print df[i]

p.add_tools(hover)
show(p)
```

The interactive graphs are still slow because of the amount of data and the processing power of the device. In the future, a better much better and powerful computer could be used for this project. However, our group learned a lot from these graphs and the added interactivity made it easier for us to look into the points of interest and interact with the data.

## Citations:

- [1] May, R. M. (1976). Simple Mathematical Models with very Complicated Dynamics. Nature, 261. Retrieved from [http://abel.harvard.edu/archive/118r\\_spring\\_05/docs/may.pdf](http://abel.harvard.edu/archive/118r_spring_05/docs/may.pdf)
- [2] Gleick, J. (1987). Chaos: Making a New Science. New York, New York: Penguin Group.
- [3] What is Chaos Theory? (n.d.). Retrieved February 2, 2019, from <https://fractalfoundation.org/resources/what-is-chaos-theory/>
- [4] Yorke, J. A., & Li, T. (1975). Period Three Implies Chaos. The American Mathematical Monthly, 82, 985-992.
- [5] Halpern, P. (2018, February 14). Chaos Theory, The Butterfly Effect, And The Computer Glitch That Started It All. Retrieved February 2, 2019, from <https://www.forbes.com/sites/startswithabang/2018/02/13/chaos-theory-the-butterfly-effect-and-the-computer-glitch-that-started-it-all/#794defa69f6c>
- [6] Oestreich, C. (2007). A History of Chaos Theory. Dialog. Clin. Neurosci. Retrieved February 2, 2019, from <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3202497/>
- [7] Lorenz, E. N. (1972). Predictability: Does the Flap of a Butterfly's Wings in Brazil Set Off a Tornado in Texas? Retrieved February 3, 2019, from [http://static.gymportalen.dk/sites/lru.dk/files/lru/132\\_kap6\\_lorenz\\_artikel\\_the\\_butterfly\\_effect.pdf](http://static.gymportalen.dk/sites/lru.dk/files/lru/132_kap6_lorenz_artikel_the_butterfly_effect.pdf)
- [8] Uittenbogaard, A. (2002, November 17). Chaos Theory for Beginners: An Introduction. Retrieved February 4, 2019, from [http://www.abarim-publications.com/ChaosTheoryIntroduction.html#.XK5\\_d-hKiM-](http://www.abarim-publications.com/ChaosTheoryIntroduction.html#.XK5_d-hKiM-)
- [9] Boeing, G. (2016, October 21). Chaos Theory and the Logistic Map. Retrieved February 15, 2019, from <https://geoffboeing.com/2015/03/chaos-theory-logistic-map/>
- [10] Bourke, P. (1997, April). The Lorenz Attractor in 3D. Retrieved February 16, 2019, from <http://paulbourke.net/fractals/lorenz/>
- [11] Bradley, J. (n.d.). Strange Attractors. Retrieved February 20, 2019, from <http://www.stsci.edu/~jbradley/seminar/attractors.html>

[1] Bradley, L. (n.d.). Strange Attractors. Retrieved February 20, 2019, from <http://www.3dschools.org/bradley/seminal/attractors.html>

In [ ]: