Noise, Chaos, and the Verhulst Population Model

Laurel J. Leonhardt
University of Wyoming, lleonha2@uwyo.edu

Follow this and additional works at: http://repository.uwyo.edu/honors_theses_16-17

Part of the Non-linear Dynamics Commons

Recommended Citation
http://repository.uwyo.edu/honors_theses_16-17/14

This Honors Thesis is brought to you for free and open access by the Undergraduate Honors Theses at Wyoming Scholars Repository. It has been accepted for inclusion in Honors Theses AY 16/17 by an authorized administrator of Wyoming Scholars Repository. For more information, please contact scholcom@uwyo.edu.
Abstract. The history of Verhulst’s logistic equation is discussed. Bifurcation diagrams and the importance of the discrete logistic equation in chaos theory are introduced. The results of adding noise to the discrete logistic equation are computed. Surprising linearity is discovered in the relationship between error bounds placed on the period two region and the amount of noise added to the system.

Key words and phrases. Bifurcation, Chaos Theory, Logistic Growth, Sensitive Dependence on Initial Conditions (SDOIC), Verhulst Population Model.
## Contents

1. Introduction 3
2. Method 4
2.1. A Note on Coding 5
3. Results 5
4. Conclusion 5
References 6
Appendix A. Code 6
Appendix B. Figures 10
1. Introduction

Researchers in all areas of study are interested in modeling their data, often with the intent of predicting future trends. Ecologists are no different. They want to know, based on previous data and current values, what a population will do over time. One of the first papers to address this was Thomas Malthus’s 1798 “Essay on the Principle of Population,” in which he argued that “population, when unchecked, increases in a geometrical ratio” [3]. However, as Malthus acknowledges in his essay, there is “a strong and constantly operating check on population from the difficulty of subsistence” [3]. In 1838, Pierre François Verhulst proposed that if $p$ is the population, then $\frac{dp}{dt}$ is an infinitesimally small increase that it receives in a very short period of time $dt$. If the population increases by geometric progression, we would have the equation $\frac{dp}{dt} = mp$. However, as the rate of population growth is slowed by the very increase in the number of inhabitants, we must subtract from $mp$ an unknown of $p$, so that the formula to be integrated can be written as

$$\frac{dp}{dt} = mp - \varphi(p)$$

The simplest hypothesis that can be made on the form of the function $\varphi$ is to suppose that $\varphi(p) = np^2$ [6]. Integration of (1) results is what is now commonly called the logistic equation and has been widely applied across numerous fields, including applications to “chemical autocatalysis, Michaelis-Menten kinetics, cancer chemotherapy, the Hill equation, the Langmuir isotherm, velocity equations of the first and second order of magnitude, oxidation-reduction potentials, erythrocyte haemolysis, the flow of streaming gases, etc” [1].

Years later, for various reasons (the use of discrete time intervals in data collection and discarding of the exact equation in favor of numerical methods for solving the ODE) the same equation was discretized. Written as a difference equation with the assumption that $m = r$ and $n = r/K$ for some growth rate $r$ and carrying capacity $K$, (1) gives the discrete logistic equation

$$p_{n+1} - p_n = r \cdot p_n \left(1 - \frac{p_n}{K}\right).$$

Even before Tien-Yien Li and James A. Yorke published their paper “Period Three Implies Chaos”, which used the discrete logistic equation as their primary example and gave the science of Chaos its name, people like Robert May were studying variations of (2) as an example of a system exhibiting deterministic chaos [2, 4]. Ecologists had previously computed results for individual values of $r$ using an iterative process in similar manner to Code1 (Appendix A). This produces three varieties of end behaviors- convergence to equilibrium, periodicity, and chaos, as seen in Figure 1.

An iterative process allows us to produce a plot of $p$ values for as many time steps as necessary, however, this only enables us to see the equation’s behavior for one particular growth rate. Given a certain population, we can determine the growth rate and carrying capacity within error, but we do not have the ability to predict $r$ or $K$ with absolute certainty. In order to easily compare the long-term behaviors for a range of $r$ values, a bifurcation diagram is used. As in Code2 (Appendix A),
a bifurcation diagram (Figure 2) is created by storing approximately the last 10% of population values for a set $p_0$ and $K$ over a range of $r$ values. These diagrams allow us to easily see the sensitivity to initial conditions in which small variations of the parameter $r$ results in widely different long-term behaviors of the population, including regions where the population stabilizes at the carrying capacity, fluctuates periodically, or exhibits chaotic behavior [5].

It is unrealistic to expect a system to maintain consistent $r$ and $K$ parameters, due to both internal and external factors. In short, there is noise in the system. This brings us to the question, “how much random noise does is required to destroy the intricate structure of the logistic attractor?” It is this question which I investigate for the remainder of this paper.

2. Method

When considering the effect of noise on the logistic attractor, I began by augmenting my bifurcation code with an option to add noise. Rather than adding to either $r$ or $K$ specifically, I chose to add a general random noise term to the population at each iteration. Next, since the effect of noise on the bifurcation diagram is influenced by the number of iterations, in addition to the magnitude of the noise, I performed trials to determine the minimum number of iterations which still presented a defined image of the period two section of the diagram without over-populating the chaotic region with so many data values as to be indistinguishable. With 2000 iterations, the bifurcations were defined with sufficient visible sharpness and the data values in the chaotic region remained distinct enough that dense bands could still be seen. For the remainder of my experimentation, I used 2000 iterations. For all calculations, $K = 100$ was used, with the intent that $p$ could then be interpreted as the percentage of carrying capacity.

To obtain a visual representation of the results of adding random noise, I used Code2 (Appendix A) to create a series of diagrams with progressively larger magnitudes of random noise added (See Figure 2). The effect of the noise was seen first as the values in the chaotic region began changing, then as the bifurcations began losing definition, and finally when the bands of periodicity within the chaotic region began to disappear. Eventually, the whole diagram lost its structure.

At this stage, it would be possible to continue creating diagrams, refining the amount of error, until I pinpointed precisely when I felt the finer structure of the logistic attractor disappeared enough that chaos was no longer discernible. This would depend on my individual judgment, which would cause it to be inconsistent, and the process would be difficult to generalize. To describe the effect in a more mathematically rigorous manner, I implemented an error bound $\epsilon$ and measured the magnitude of random noise which could be added while the majority of data remained within $\pm \epsilon$ of the data values computed without noise. Imposing the error bound on the chaotic region would hardly be practical. I chose to look at a sample of $r$ values from 2.1 to 2.4, which encapsulates much of the period 2 region, and increased the magnitude of noise in the system so long as 90% of the last 10% of data values for 90% of sampled $r$ values stayed within the error bound.

Allow me to explain. Code3 (Appendix A) iterates (2) with $p_0 = 25$ and $K = 100$ for a specific $r$ value, saving the last 10% of data values in a vector (called ydata). It then recomputes for the same value of $r$, with random noise added, still saving the last 10% of data values (called xdata). Then, the values from the second vector
are compared with two values from the first vector (since it is period two) to see if that difference is less than the set error tolerance $\epsilon$ 90% of the time. This process is repeated for several $r$ values ranging from 2.1 to 2.4. If 90% of the $r$ values had 90% of population values inside the error bound, then the whole process is repeated with the magnitude of noise increased. The code reports the first magnitude of noise which causes data outside the error bound and the percent of $r$ values which satisfy the requirement of having 90% data inside the bound. Since the noise is random, it is necessary to run the code several times to get a good idea of the noise limit.

I found the noise limits for three separate sets of $r$ values with the same five $\epsilon$ values for each. Within each set, for each $\epsilon$ value, three trials were performed and the full range of results was reported. Set A used 10 evenly distributed $r \in [2.1, 2.106012024]$. Set B consisted of 10 evenly spaced $r \in [2.1, 2.4]$. The final set, set C contained 100 evenly distributed $r \in [2.1, 2.4]$.

2.1. A Note on Coding. All of my code is written for use with Python 2.7. Updating the code for use with Python 3.5 is a relatively simple matter consisting of appending the command “.split(‘,’)” to any line asking for multiple inputs and specifically declaring the type for each variable (e.g., inserting “r_min=float(r_min)” to turn the input for r_min into a decimal value).

3. Results

The results of my numerical experimentation are tabulated below. The values from set A and set B have been plotted in Figure 3, along with their respective lines of best fit. It is worth noting that all three sets of data express nearly linear relationships between $\epsilon$ and the noise limit.

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>Noise A</th>
<th>Noise B</th>
<th>Noise C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.510 - 0.511</td>
<td>0.326 - 0.385</td>
<td>0.396 - 0.406</td>
</tr>
<tr>
<td>0.5</td>
<td>0.253 - 0.254</td>
<td>0.186 - 0.199</td>
<td>0.208 - 0.210</td>
</tr>
<tr>
<td>0.25</td>
<td>0.120 - 0.121</td>
<td>0.090 - 0.100</td>
<td>0.100 - 0.102</td>
</tr>
<tr>
<td>0.1</td>
<td>0.050 - 0.051</td>
<td>0.036 - 0.040</td>
<td>0.040 - 0.043</td>
</tr>
<tr>
<td>0.05</td>
<td>0.023 - 0.024</td>
<td>0.020 - 0.023</td>
<td>0.020 - 0.022</td>
</tr>
</tbody>
</table>

4. Conclusion

These results seem to indicate that there is a linear relationship between the magnitude of random noise which can be added to the discrete logistic equation and the error bound on the period two region, despite the non-linearity of the difference equation.

To further investigate this phenomenon, it is necessary that we increase the number of trials and decrease the step between error values, to obtain more accurate and more specific values for the noise limits. It is unclear if a similar relationship would hold in other periodic regions. It seems reasonable to expect this linearity to generalize and it would be a straightforward procedure to modify Code3 (Appendix A) to investigate other periodic regions. Since each successive bifurcation is related to the previous bifurcation by Feigenbaum’s number $\delta$, it would not be entirely unexpected if Feigenbaum’s number made an appearance in the relationship between the noise limits of two periodic regions for proportionate $\epsilon$ values. However, this may be complicated by the number of iterations necessary for each bifurcation.
to become clearly defined. Considering the effect of the number of iterations was beyond the scope of this paper, but it would also be worth investigating.

References
p = float(input("enter starting value: "))
r = float(input("enter r value: "))
nmax = input("enter maximum # of iterations: ")

pop=[p]
x=[0]

print (p,r,nmax)
raw_input("press<enter>")

for n in range (1,nmax):
p = p + r * p * (1 - p/100.)
pop.append(p)
x.append(n)

import pylab as pl
pl.plot(x, pop, 'ro', markersize=5)
pl.xlabel('Time Step')
pl.ylabel('Population')
pl.title('Logistic Model')
pl.show()
```python
print("For multiple input values on one line, separate inputs with a comma.")

r_min, r_max = (input("enter minimum and maximum values of r: 
"))
r_num = input("enter number of r values: ")
p_start, p_cc = input("enter starting value of population and carrying capacity: ")
p_min, p_max = input("enter minimum and maximum population: ")
maxiter = input("enter maximum number of iterations: ")
n_int = (input("enter interval of noise: "))
n_maxint = (input("enter number of intervals: "))

import pylab as pl
import random

for k in range (0,n_maxint+1):
    rand_mag = k*1.*n_int
    rdata = []
pdata = []

    for i in range(0,r_num):
        r = r_min + (r_max - r_min) * float(i) / (r_num - 1.)
        x = p_start
        for j in range(0,int(.9*maxiter)):
            rand = random.uniform(-rand_mag,rand_mag)
            x = x + r*x*(1-x/p_cc) + rand
        for j in range(int(.9*maxiter),maxiter):
            rand = random.uniform(-rand_mag,rand_mag)
            x = x + r*x*(1-x/p_cc) + rand
        if p_min<x<p_max:
            rdata.append(r)
pdata.append(x)
        pl.scatter(rdata,pdata,s=.01,c='r',marker='o')
pl.xlabel('r Value')
pl.ylabel('Population')
pl.title('%.5f, %.5f, %.5f Logistic Attractor' % (float(r_min),r_max, float(rand_mag)))
pl.savefig('%s_%s_%s.jpeg' % (float(r_min),float(r_max),float(rand_mag)),bbox_inches='tight')
pl.close()
```

```python
code3.py

import random
# uniform(a,b) returns random values in [a,b]

# hardcoded
rmin, rmax = 2.1, 2.4
maxiter = 2000
val = 1
count = 1

# get user input
error = float(input("Enter error bound: "))
num = input("How many r values would you like to check? ")
start = input("Enter initial amount of noise: ")
step = input("Enter increment which noise increases by: ")

# begin calculating
while val==1:
    rand_mag = step*count + start
    count += 1
    p=0
    for i in range(0,num):
        r = rmin + (rmax - rmin) * float(i)/float(num-1)
        y = 25.
        ydata = []
        for j in range(0,int(.9*maxiter)):
            y = y + r*y*(1-y)/100.
        for j in range(int(.9*maxiter),maxiter+1):
            y = y + r*y*(1-y)/100.
        ydata.append(y)
        # Modified - noise added
        xdata = []
        x = 25.
        for j in range(0,int(.9*maxiter)):
            rand = random.uniform(-rand_mag, rand_mag)
            x = x + r*x*(1-x)/100. + rand
        for j in range(int(.9*maxiter),maxiter+1):
            rand = random.uniform(-rand_mag, rand_mag)
            x = x + r*x*(1-x)/100. + rand
        xdata.append(x)
        s=0
        for j in range(0,len(xdata)):
            check = min(abs(ydata[1]-xdata[j]), abs(ydata[2]-xdata[j]))
            if check <= error:
                s+=1
        length = float(len(xdata))
        if float(s/length) >= 0.9:
            p'=1
            print rand_mag, p/float(num)
        if p/float(num) < .9:
            val = 0
```

This code snippet seems to involve numerical calculations and user input to determine certain values. It appears to be a part of a larger program that calculates and processes data according to user inputs.
Figure 1. Long-term Behaviors
Figure 2. Noise

2_0_3_0_0.jpeg
2_0_3_0_5.jpeg
2_0_3_0_0_1.jpeg
2_0_3_0_3_1.jpeg
2_0_3_0_5_1.jpeg
2_0_3_0_7_1.jpeg
2_0_3_1_0.jpeg
2_0_3_5_0.jpeg

Number of Fish

r Value

0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140

0 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140
Figure 3. Results

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure3.png}
\caption{Results}
\end{figure}

\textbf{Department of Mathematics, The University of Wyoming, Laramie, WY, 82071}
\textit{E-mail address: lleonha2@uwyo.edu}

\textbf{University Honors Program, The University of Wyoming, Laramie, WY, 82071}
\textit{E-mail address: Shive@uwyo.edu}