What are the strengths and weaknesses of the most common mathematical methods used to model population growth?

IB Mathematics HL

Candidate Code: hnn524

Word Count: 3,971

Table of Contents

Introduction

The world of living things is known for being dynamic, complex and inscrutable. We share a planet with 1.3 million discovered living species, and up to 8.7 million undiscovered ones[1], all engaged in an epic battle for survival. Naturally, the Earth's biosphere is incredibly complex, and the successes and failures of each species are determined by countless factors. Nevertheless, within this complexity, there are signs of simple patterns that seem to govern the behaviour of any ecosystem. The field of mathematical ecology is devoted to the study of these patterns, building mathematical models to explain and predict the interactions between organisms sharing a habitat.

The following question is therefore crucial to our understanding of ecology:

What are the strengths and weaknesses of the most common mathematical methods used to model population growth?

In this investigation, I will aim to answer that question by exploring a range of simple to complex mathematical models and simulations, addressing the advantages and disadvantages of each, and discussing their implications for the real world.

[1] Mora et al., 2011

Population Models

The most common method of modelling ecological systems is population modelling. A population model takes as input a population size and a duration of time, and outputs a predicted population size after that length of time has elapsed.

The simplest phenomenon often approached by models is population growth. In most cases, organisms reproduce individually (an exception being hive-based insect species, in which one or more ‘queens’ fulfil all reproductive responsibilities in the population). In individually-reproducing organisms, increasing the number of organisms leads to an increased number of offspring per year, and thus a higher rate of reproduction – that is, the total rate of growth is proportional to the population size. This relationship is most simply expressed in the exponential growth model.

The Exponential Growth Model

The proportionality between population size and rate of population growth can be described as a first-order differential equation.

$$\frac{dN}{dt} = kN$$

Where N is the number of organisms, t is time, and k is a constant of proportionality. k can be said to represent the fertility of the species; it is the average offspring produced per capita in 1 unit of time. Naturally, k must be positive unless the organisms die faster than they reproduce. The equation can be solved algebraically for N using separation of variables and integrating:

$$\frac{dN}{N} = kdt$$

$$\int \frac{dN}{N} = \int k\,dt$$

$$lnN = kt + c$$

$$N = e^{kt+c}$$

$$N = N_0 e^{kt}\text{, where } N_0 = e^c$$

The solution is an exponential function. Where $t = 0$, $N = N_0$, so N_0 represents the starting population of the organism. This model grows at an exponentially increasing rate:

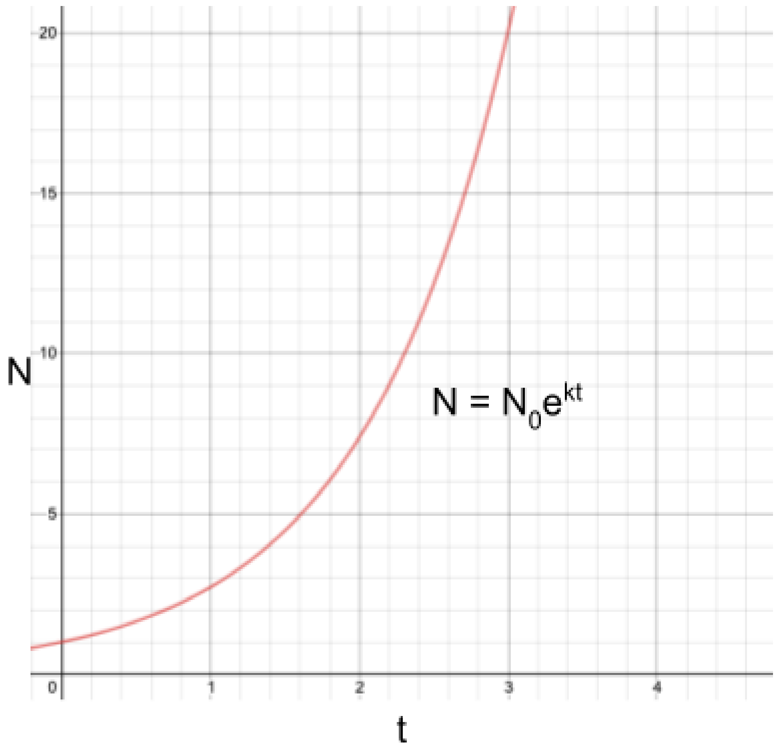


Figure 1: A graph of $N = N_0e^{kt}$ where $N_0 = 1$ and $k = 1$. All images in this essay were created by the author.

Unfortunately, this model has shortcomings. The first is that in an ecosystem, the growth of the population is discrete with respect to time, not continuous. In a continuous model, if a parent produces 1 child per year, then in 0.5 years, it will produce 0.5 children. That half-child is then considered part of the population, so the model assumes that that child will immediately begin reproducing, but at half the rate of a full organism. This means that the continuous solution will *overestimate* the population size, since it assumes that undeveloped organisms can reproduce. To avoid this, a discrete-time iterative method can be used.

Discrete-time model – the Euler Method

One example of a discrete-time model is the implementation of the Euler Method, an iterative process that approximates a solution to a differential equation. In the case of the exponential model:

$$\tfrac{dN}{dt} = kN$$

The Euler Method approximation is a sequence N_t (t being an integer time duration):

$$N_t = N_{t-1} + \tfrac{dN}{dt}dt$$

$$N_t = N_{t-1} + kN_{t-1}dt$$

Here, dt is the time interval of the approximation. Any positive interval can be used, and the accuracy of the method increases as dt decreases (the continuous method is the limit as $dt \to 0$ of the Euler approximation). However, since we use arbitrary units for t, it is simpler to use $dt = 1$. The above equation can be written as:

$$N_t = N_{t-1}(1 + k)$$

This is a geometric sequence with common ratio $1 + k$:

$$N_t = N_0(1 + k)^t$$

Like the continuous model, this is an exponential function. However, it is less than or equal to the continuous model for all t:

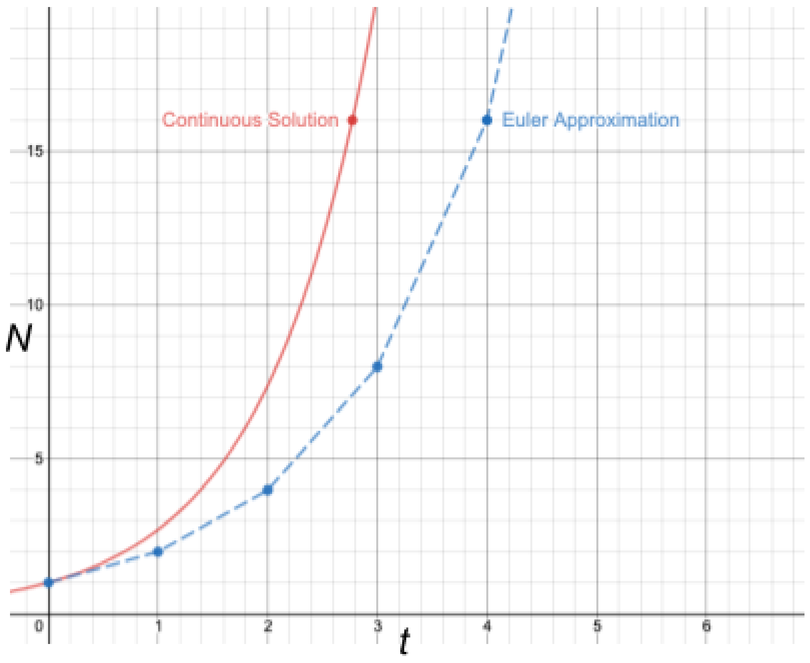


Figure 2: Graph of $N = N_0 e^{kt}$ *and its Euler approximation*

Because living populations are themselves discrete-time systems, this Euler approximation is more accurate than the continuous model.

Other discrete/continuous issues

There are other problems caused by modelling populations as a continuous process. Firstly, most complex organisms have discrete stages of development (e.g.: child, juvenile, adult). Each stage will have different rates of reproduction and resource needs. In the case of a population of 100,000 organisms, where 40% are children, 50% adults and 10% juveniles, we can build a model where all organisms are identical 4:5:1 hybrids of child, adult and juvenile. While this is clearly false, their behaviours on a large scale tend to average out.

However, this becomes a problem in smaller populations. A population of 1 adult and 2 infants is weak, because if the adult starves, so will the infants – only one death is required for extinction. With 10,000 adults and 20,000 infants, a single death is insignificant. Since a death can happen almost randomly, models of small populations are less accurate. Clearly, these models are more accurate for species with no developmental stages: for instance, bacteria reproduce by splitting, so both the ‘parent’ and ‘offspring’ cells are functionally identical[2].

The Resource Abundance Issue

A major problem is that in this model, $\lim_{t \to \infty} N = \infty$. With limited resources, the population can’t grow infinitely. Also, reproduction rate can’t be proportional to population size: more organisms means less food per organism, and reduced reproduction rates. This problem can be solved using the logistic model.

[2] Carlson, 2007

The Logistic Model

To deal with the problem of scarcity, biologist Robert May proposed the logistic model in 1976[3]. The model proposes that the rate of population growth is dependent on both the existing population size and the resource abundance of the environment, and is recursively defined by the sequence:

$$N_t = rN_{t-1}(1 - \tfrac{N_{t-1}}{K})$$

Where N and t are population and time as before. r is a constant analogous to e^k in the exponential model (r is used by convention), representing the average rate of reproduction. $(1 - \tfrac{N}{K})$ represents the abundance of available resources. Where $N = K$, the population will consume all its resources and starve (N_t drops to zero if $N_{t-1} = K$); therefore K can be described as the 'maximum capacity' supportable by the ecosystem. A single iteration of the sequence is called the "logistic map".

Where resources are abundant (ie. $K \to \infty$), the model behaves as a geometric sequence with common ratio r:

$$\lim_{K\to\infty} rN_{t-1}(1 - \tfrac{N_{t-1}}{K}) = rN_{t-1}$$

A continuous exponential model can be derived from this:

$$N_t = rN_{t-1}$$

$$\therefore N = N_0 r^t = N_0 e^{ln(r)t}$$

[3] Du Sautoy, 2017

The logistic model can thus be seen as a continuation of the exponential model, extended to situations of resource scarcity.

Continuous behaviour of the logistic model

This model can also be expressed as a continuous differential equation:

$$\frac{dN}{dt} = \frac{N_t - N_{t-dt}}{dt}$$

As this is a discrete sequence, dt is a fixed interval. For simplicity, we let $dt = 1$, an arbitrary unit of time:

$$\tfrac{dN}{dt} = N_t - N_{t-1} \quad (1)$$

$$N_t = rN_{t-1}(1 - \tfrac{N_{t-1}}{K}) \quad (2)$$

Substituting (2) into (1):

$$\tfrac{dN}{dt} = rN_{t-1}(1 - \tfrac{N_{t-1}}{K}) - N_{t-1}$$

We can now rename N_{t-1} to just N, since it’s the only population variable:

$$\tfrac{dN}{dt} = rN(1 - \tfrac{N}{K}) - N$$

To show that this model also solves the problem of infinite growth observed in the exponential model, we find N such that $\tfrac{dN}{dt} = 0$:

$$rN(1 - \tfrac{N}{K}) - N = 0$$

$$N(r - \tfrac{r}{k}N - 1) = 0$$

$$-N(\tfrac{r}{K}N - (r - 1)) = 0$$

$$N = 0 \text{ or } N = K(1 - \tfrac{1}{r})$$

The solutions $N = 0$ and $N = K(1 - \frac{1}{r})$ are important because they represent the horizontal asymptotes of the model. Since $\frac{dN}{dt}$ is positive where $0 < N < K(1 - \frac{1}{r})$, the population will increase as $t \to \infty$; that is, it will approach the greater of the two asymptotes: $K(1 - \frac{1}{r})$. $K(1 - \frac{1}{r})$ will always be less than K, since $r > 0$. Thus while K is the 'maximum capacity', r determines how large the population can grow, relative to that capacity.

Solving the differential equation

The differential equation can be solved by separation of variables:

$$\frac{dN}{dt} = rN(1 - \frac{N}{K}) - N$$

$$\frac{dN}{rN(1-\frac{N}{K})-N} = dt$$

$$\int \frac{dN}{rN(1-\frac{N}{K})-N} = \int dt$$

$$\int \frac{1}{N(r-1-\frac{r}{K}N)} dN = t + c$$

Using partial fractions, we can see that:

$$\frac{1}{\frac{r}{K}N} + \frac{1}{r-1-\frac{r}{K}N} = \frac{r-1-\frac{r}{K}N+\frac{r}{K}N}{\frac{r}{K}N(r-1-\frac{r}{K}N)} = \frac{r-1}{(\frac{r}{K})} \frac{1}{N(r-1-\frac{r}{K}N)}$$

And therefore:

$$\frac{r}{K(r-1)}\left[\frac{1}{\frac{r}{K}N} + \frac{1}{r-1-\frac{r}{K}N}\right] = \frac{1}{N(r-1-\frac{r}{K}N)}$$

We can substitute:

$$\frac{r}{K(r-1)}\int\left[\frac{1}{\frac{r}{K}N}+\frac{1}{r-1-\frac{r}{K}N}\right]dN = t + c$$

At which point integration by inspection becomes trivial:

$$\frac{r}{K(r-1)}\left[\frac{K}{r}ln(\tfrac{r}{K}N) - \frac{K}{r}ln(r-1-\tfrac{r}{K}N)\right] = t + c$$

Taking the absolute value of the natural logarithms is unnecessary where $0 < N < K(1-\frac{1}{r})$,

which constitutes the full range of possible populations. Simplifying, we get:

$$\frac{1}{(r-1)}\left[ln(\tfrac{r}{K}N) - ln(r-1-\tfrac{r}{K}N)\right] = t + c$$

$$ln\left(\frac{\frac{r}{K}N}{r-1-\frac{r}{K}N}\right) = (r-1)t + c'$$, where $c' = (r-1)c$

$$\frac{\frac{r}{K}N}{r-1-\frac{r}{K}N} = Ae^{(r-1)t}$$, where $A = e^{c'}$

We can redefine A in terms of the initial population, N_0. Substituting $t = 0,\ N = N_0$ gives:

$$\frac{\frac{r}{K}N_0}{r-1-\frac{r}{K}N_0} = A$$

Therefore:

$$\frac{\frac{r}{K}N}{r-1-\frac{r}{K}N} = \frac{\frac{r}{K}N_0}{r-1-\frac{r}{K}N_0}e^{(r-1)t}$$

$$N = \frac{N_0}{r-1-\frac{r}{K}N_0}e^{(r-1)t}(r-1-\tfrac{r}{K}N)$$

$$N = (r-1)\frac{N_0}{r-1-\frac{r}{K}N_0}e^{(r-1)t} - \frac{r}{K}N\frac{N_0}{r-1-\frac{r}{K}N_0}e^{(r-1)t}$$

$$N + \frac{r}{K}N\frac{N_0}{r-1-\frac{r}{K}N_0}e^{(r-1)t} = (r-1)\frac{N_0}{r-1-\frac{r}{K}N_0}e^{(r-1)t}$$

$$N\left(1+\frac{r}{K}\frac{N_0}{r-1-\frac{r}{K}N_0}e^{(r-1)t}\right)=(r-1)\frac{N_0}{r-1-\frac{r}{K}N_0}e^{(r-1)t}$$

$$N=\frac{(r-1)\frac{N_0}{r-1-\frac{r}{K}N_0}e^{(r-1)t}}{1+\frac{r}{K}\frac{N_0}{r-1-\frac{r}{K}N_0}e^{(r-1)t}}$$

Simplifying, we get:

$$N=\frac{N_0e^{(r-1)t}(r-1)}{r-1-\frac{r}{K}N_0+\frac{r}{K}N_0e^{(r-1)t}}$$

$$N=\frac{KN_0e^{(r-1)t}(r-1)}{rK-K-rN_0+rN_0e^{(r-1)t}}$$

Finally:

$$N=\frac{KN_0e^{(r-1)t}(r-1)}{K(r-1)+N_0r(e^{(r-1)t}-1)}$$

I will refer to this equation as “the continuous solution to the logistic model” throughout this essay, or just “the continuous solution” within this section. I will now explore some behaviours of this model.

Exponential behaviour as $K\to\infty$

It is clear that this model is a general case of the exponential model discussed earlier. If we consider $\lim_{K\to\infty} N$, the second term in the denominator is a K^0 term and thus becomes negligible:

$$\lim_{K\to\infty}\frac{KN_0e^{(r-1)t}(r-1)}{K(r-1)+N_0r(e^{(r-1)t}-1)}=\frac{KN_0e^{(r-1)t}(r-1)}{K(r-1)}=N_0e^{(r-1)t}$$

As the maximum number of organisms supported by the ecosystem approaches infinity (i.e. available resources are unlimited), this model turns into an exponential model. This means that the logistic model can be properly thought of as a mathematical extension of the exponential model into situations where resources are limited.

Exponential behaviour of the discrete model

The discrete-time solution to $\lim_{K\to\infty} N$ was discussed earlier in the essay:

$$\lim_{K\to\infty} N = N_0 e^{ln(r)t}$$

The continuous solution is similar, only replacing $ln(r)$ with $(r-1)$. The latter will always be greater than or equal to the former:

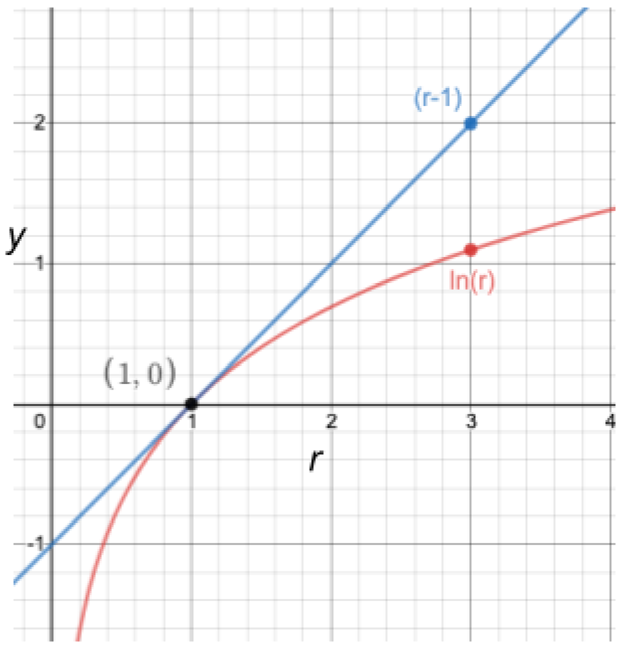


Figure 3: Graph of $y = (r-1)$ and $y = ln(r)$ where r is the horizontal axis.

This means the continuous solution will always be greater than the discrete solution (except for the trivial case $r = 1$).

Graphing the logistic model

The continuous logistic model can be graphed to produce what is known as the logistic curve:

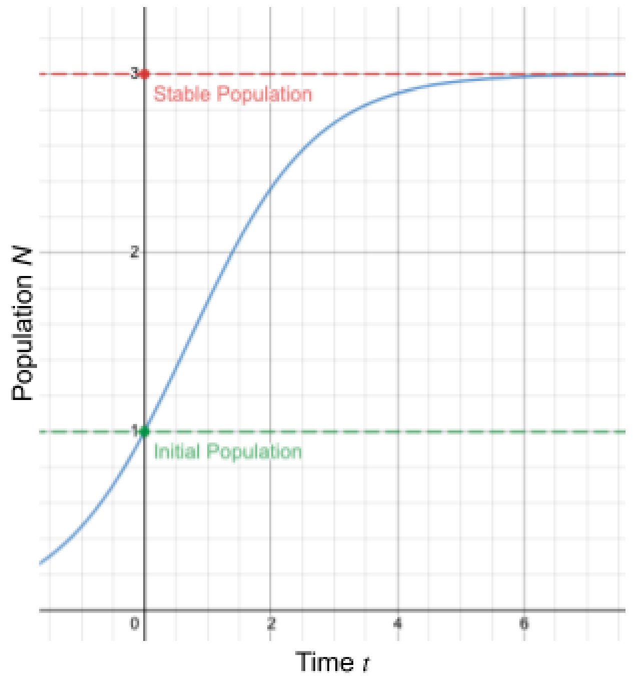


Figure 4: Graph of the continuous solution to the logistic model, where the horizontal axis is time and the vertical axis is population. Parameters for this function are $N_0 = 1$, $r = 2$, and $K = 6$.

The dashed lines labelled “initial population” and “stable population” are defined by $y = N_0$ and $y = K(1 - \frac{1}{r})$ respectively. As I determined earlier, the stable population is the second solution to $\frac{dN}{dt} = 0$, the first being $N = 0$. In the graph, the initial population is one-third of the stable population, so the population grows in a characteristic "s" shape, asymptotically approaching $N = 3$.

As I discussed earlier, changing the value of N_0 has no effect on the resulting stable population:

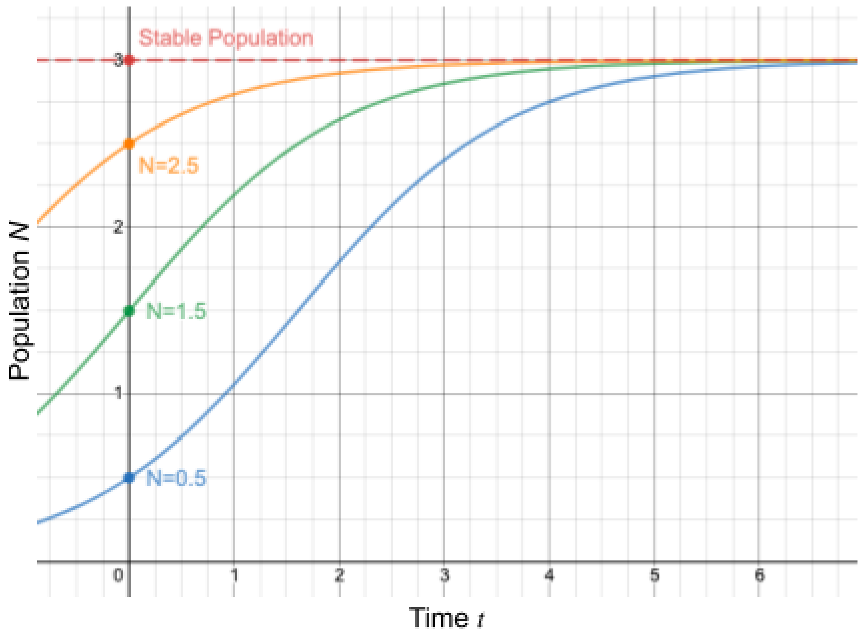


Figure 5: Graphs of continuous logistic curves for N_0 values of 0.5, 1.5 and 2.5, each with $r = 2$ and $K = 3$.

Here, the stable population is determined by $N_{stable} = K(1 - \frac{1}{r})$, which approaches K as $r \rightarrow \infty$.

As discussed earlier, a continuous model is useful for large populations, but has difficulty explaining unpredictability in smaller populations. It makes sense to explore the discrete-time form of the model, the logistic map.

Discrete-time approximation of the logistic model

The logistic map recursively defines a sequence of successive populations over a specific time interval:

$$N_t = rN_{t-1}(1 - \tfrac{N_{t-1}}{K})$$

This can become quite complex. For example:

Step	Calculation
N_0	$N_0 = 0.2$
N_1	$rN_0(1 - \frac{N_0}{K}) = 2 \times 0.2 \times (1 - 0.2) = 0.32$
N_2	$rN_1(1 - \frac{N_1}{K}) = 2 \times 0.32 \times (1 - 0.32) = 0.4352$
N_3	$rN_2(1 - \frac{N_2}{K}) = 2 \times 0.4352 \times (1 - 0.4352) = 0.49160192$

Table 1: Three iterations of the logistic map with parameters $N_0 = 0.2,\ K = 1,\ r = 2$

Due to this complexity, we will analyze overarching patterns in the sequence's behaviour. The logistic map can be presented as a Poincaré section, which is a plot where the value input to the map is graphed against the next value output[4]. This allows us to graphically represent a single step of the logistic map, which takes the form of a negative parabola:

[4] Teschl, 2012

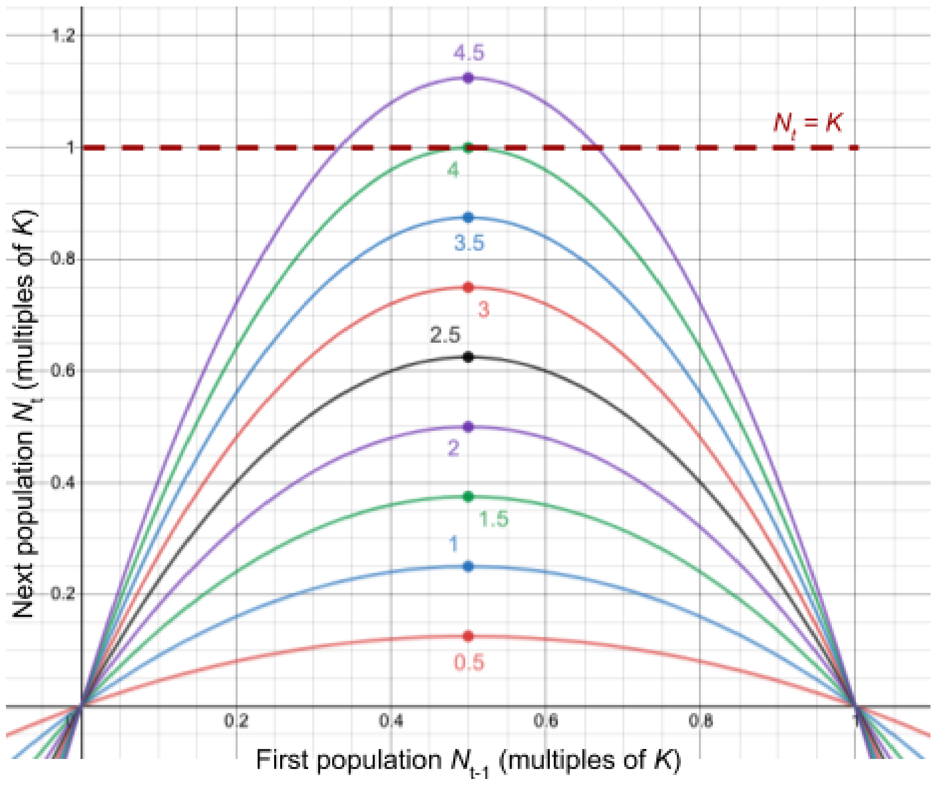


Figure 6: Poincaré section showing 9 logistic maps, each with a different value of r (labelled below the curves). All populations are in multiples of K.

As is clearly visible from fig. 6, a value of r greater than 4 can produce a subsequent population greater than K. In these cases, the model breaks down, because where $N_{t-1} > 1K$, the value of N_t will be negative:

$$N_{t-1} > K$$

$$\frac{N_{t-1}}{K} > 1$$

$$0 > 1 - \frac{N_{t-1}}{K}$$

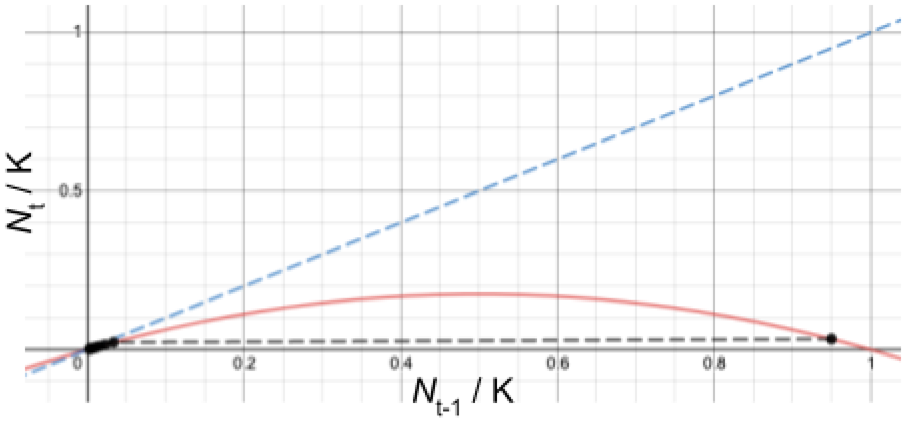
$$0 > rN_{t-1}(1 - \frac{N_{t-1}}{K})$$

$$0 > N_t$$

This means the map is only defined for $0 \le r \le 4$.

To further investigate the logistic map, I have created Poincaré sections and population-time graphs to show its behaviour over sequential iterations, using different values of r. The blue dashed line shows where $N_t = N_{t-1}$; in other words, the stable population. The black dashed lines show the change in population over time. To make the changes easier to see, I've set the initial population, N_0, to $0.95K$ for all examples.

1. $r = 0.7$, $N_0 = 0.95K$:



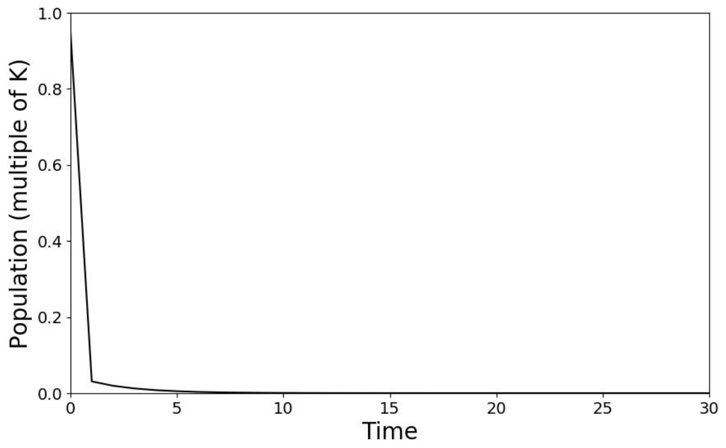
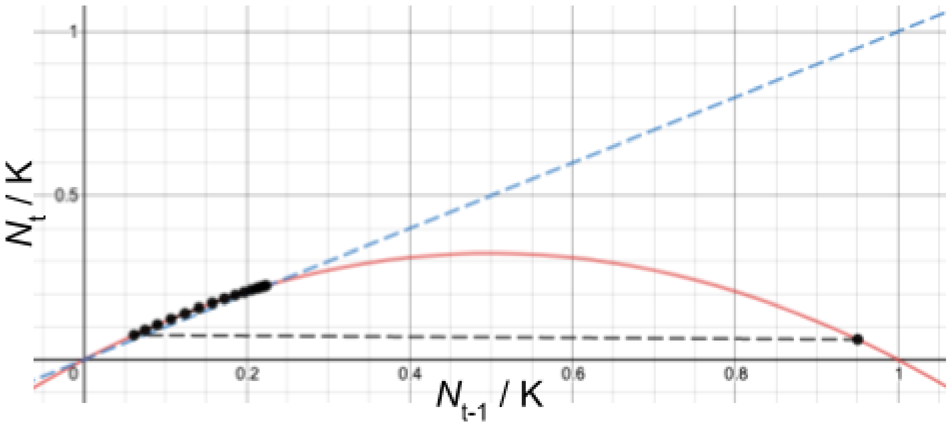


Figure 7: N quickly converges towards 0, the only stable population.

2. $r = 1.3$, $N_0 = 0.95K$:



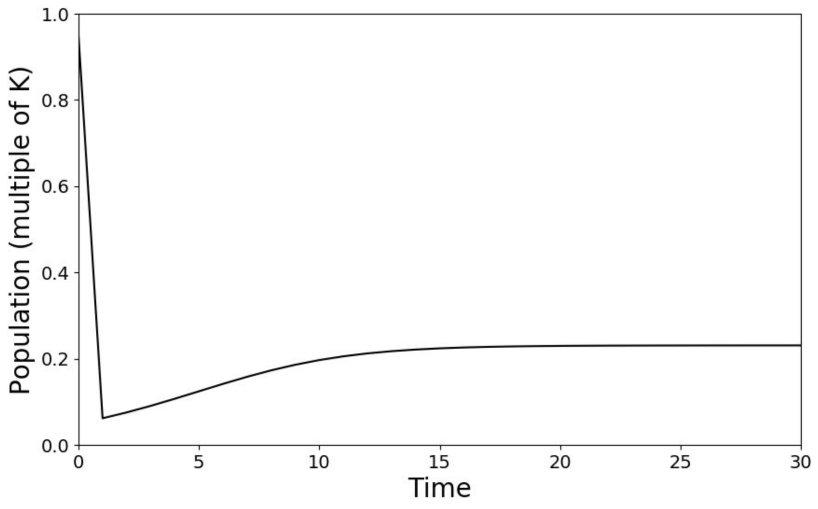
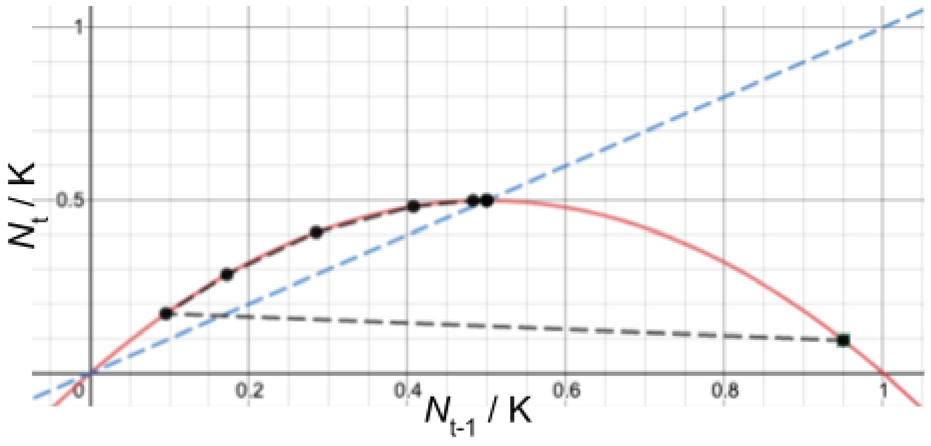


Figure 8: N *converges towards* $0.2308K$ *, the intersection with* $N_t = N_{t-1}$ *. This value is also equal to* $K(1 - \frac{1}{1.3})$ *as proven earlier.*

3. $r = 2$, $N_0 = 0.95K$:



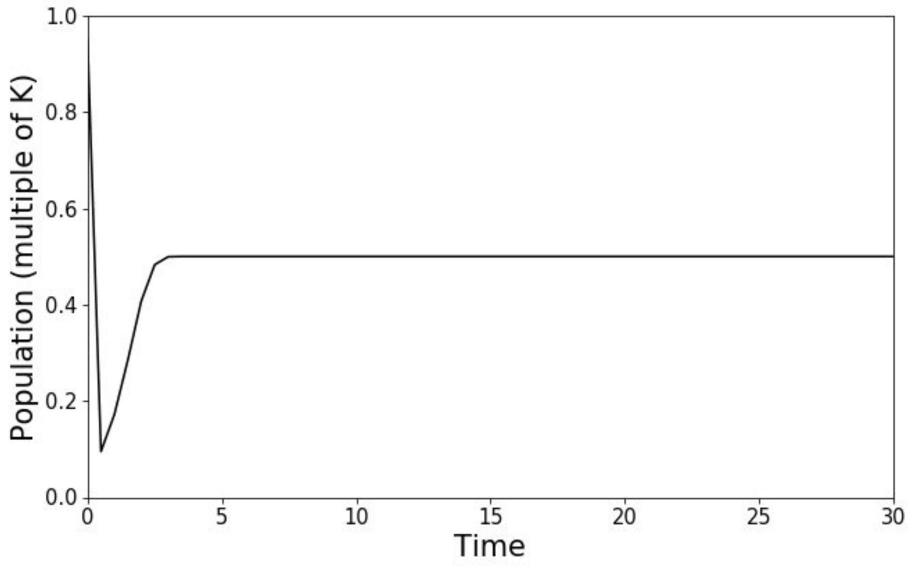
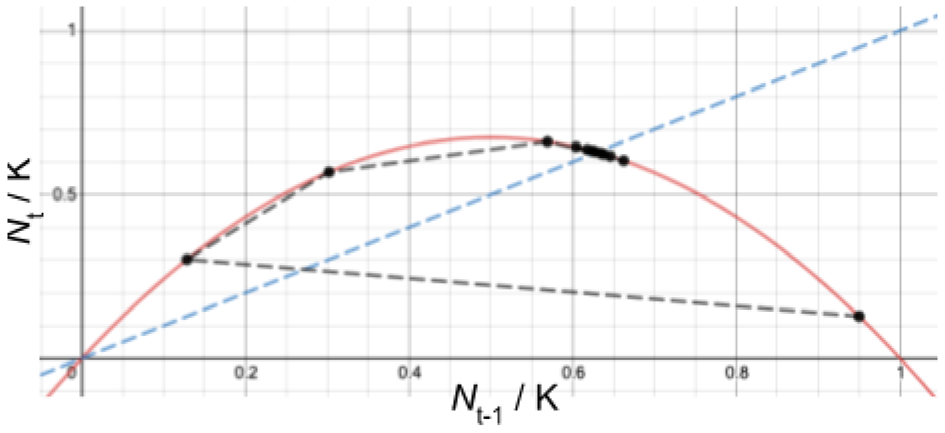


Figure 9: Converges to $0.5K$. Note the resemblance to the continuous logistic curve.

4. $r = 2.7$, $N = 0.95K$:



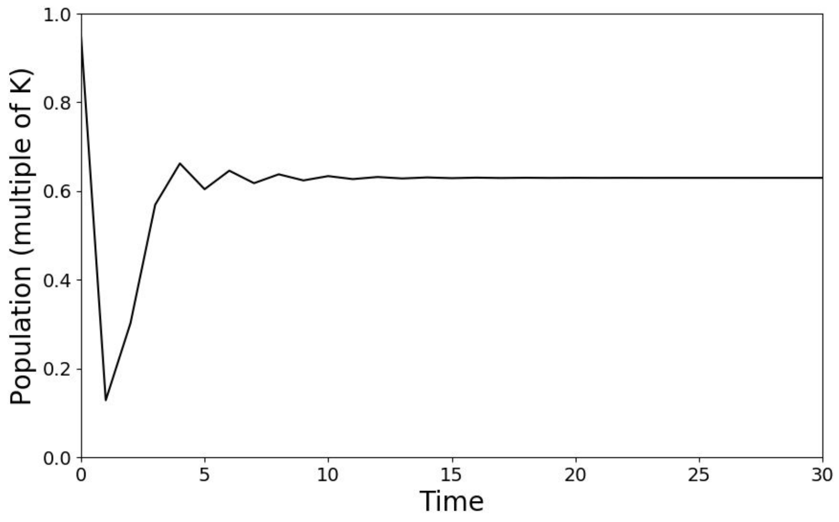
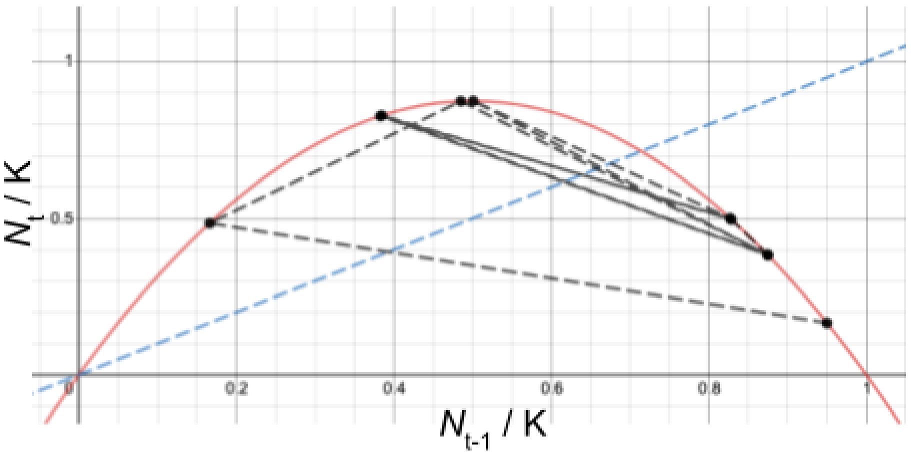


Figure 10: Converges to $0.6296K$ *but overshoots and approaches it by damped oscillation about the stable point.*

5. $r = 3.5$, $N = 0.95K$:



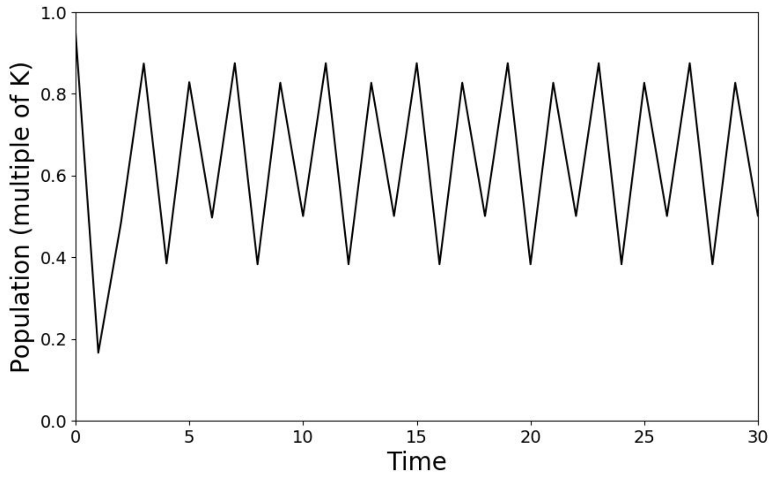
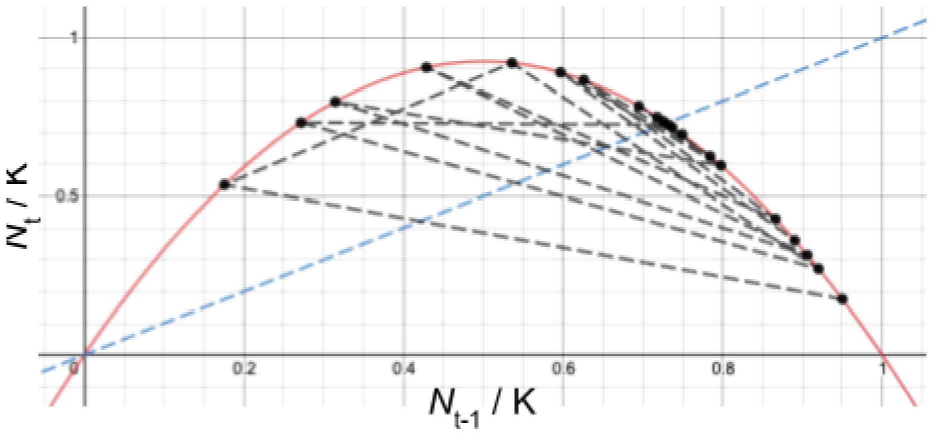


Figure 11: Theoretical stable population is $0.7143K$*, but the map oscillates constantly around this point and never reaches it.*

6. $r = 3.7$, $N = 0.95K$:



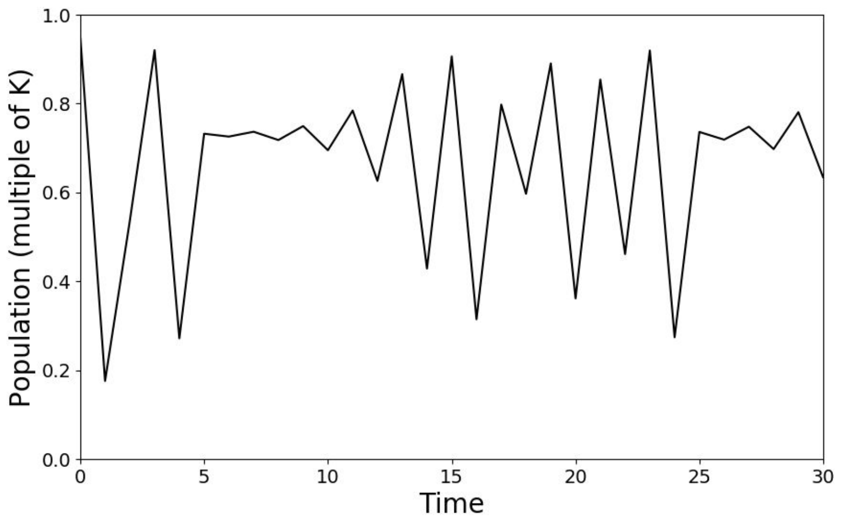


Figure 12: $r = 3.7$, $N_0 = 0.95K$, theoretical stable population is $0.7297K$, but the map produces chaotic results and becomes unpredictable.

As r increases, the discrete-time model and the continuous model behave increasingly differently. It is worth mentioning that the discrete model is more accurate than the continuous one, because real-life populations grow in discrete increments. However, the key difference between the two models is that at high values of r, the discrete-time model behaves chaotically.

Chaotic behaviour of the discrete-time model

This is aptly illustrated with a colour-gradient graph describing the relationship between N and t across a spectrum of r values:

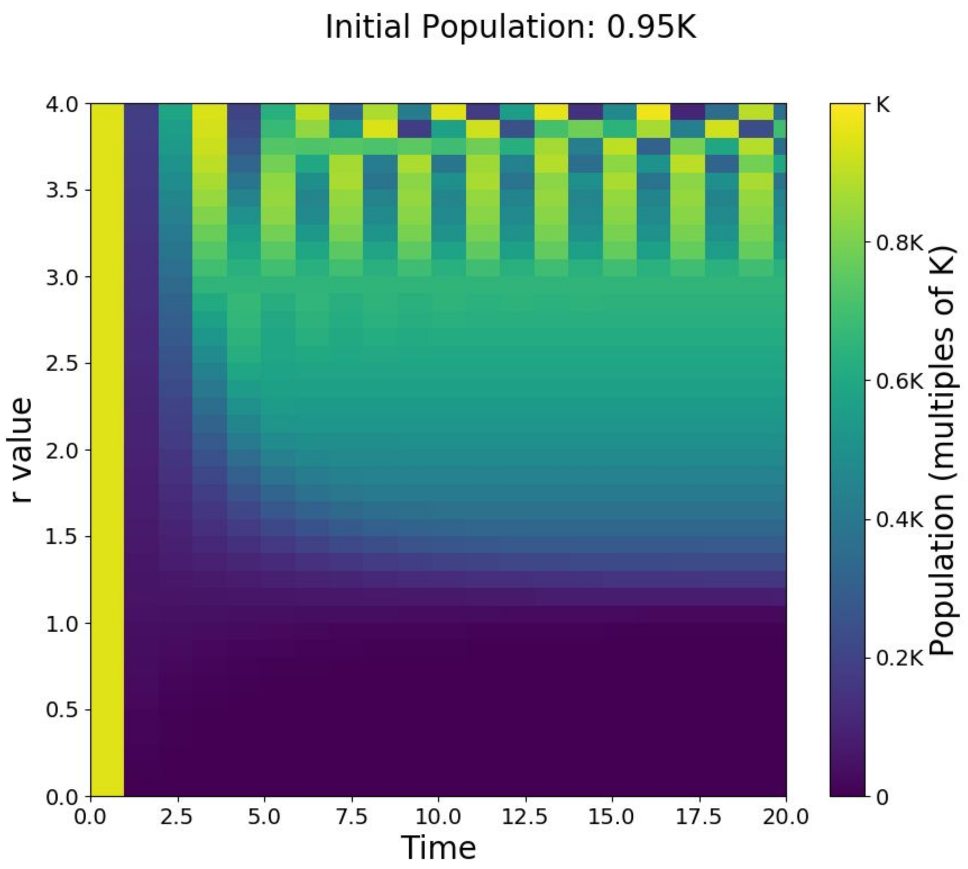


Figure 13: Colour gradient graph showing the behaviour of the map over time. The first 20 iterations of the map are shown for all 40 values of r tested, each with $N_0 = 0.95K$.

Fig. 13 shows that for values of r below 1, the population gradually dies out (dark purple), because there are more deaths than births. For values ranging from 1 to 2, the population quickly stabilizes at $N = K(1 - \frac{1}{r})$, appearing very similar to the continuous model. Between 2 and 3, the population approaches the same limit, but does so through damped oscillations. For values of $r > 3$, the oscillations are undamped and never reach $K(1 - \frac{1}{r})$. Studies of the logistic map have shown that at values of r greater than approximately 3.56995, the oscillations become chaotic and take on an infinite period, never repeating in finite time[5].

There are dangerous potential consequences for a population that reproduces too quickly and begins to exhibit chaos. Since the chaotic oscillations never repeat themselves, N eventually passes through every value in the interval $(0K,\ 1K)$ exclusive. While the value of N never reaches zero, the population will eventually fall below one organism. The issue is known in the field of mathematical ecology as the “Atto-fox Problem”, “Atto-” being the SI prefix meaning 10^{-18}.[6] The problem is that the model doesn’t understand that a population can’t be infinitely divided. There could be one fox remaining, or 10^{18} Atto-foxes, each being 10^{-18} of a fox. An individual Atto-fox will consume 10^{-18} times as much food and reproduce 10^{-18} times as frequently as one fox. In reality, if the population is reduced to 0.000000000000000001 foxes, the foxes are extinct and will no longer reproduce at all.

Therefore, for values of r greater than 3.56995, the population will eventually drop below one organism and become extinct. This suggests a trade-off: on the one hand, increasing r increases the stable population $K(1 - \frac{1}{r})$. On the other, high r can cause dangerous instability.

[5] Boeing, 2016
[6] Lobri et al., 2015

Comparison between continuous-time and discrete-time logistic models

The continuous logistic model is stable for $r > 1$. Being continuous with respect to time, it assumes the population responds smoothly and instantaneously to changes in the environment. The discrete model, on the other hand, behaves chaotically where the stable population is close to the carrying capacity K of the ecosystem. Therefore, if a change in the environment causes K to decrease, N will be much closer to K, which could lead to chaotic behaviour.

This provides valuable insight into real-life behaviours: due to the discrete-time nature of organisms, damage to the environment can catastrophically damage the population.

In general, as the length of time increases, the continuous model becomes more accurate because it is defined by $\frac{dN}{dt}$ with time in arbitrary units. As the length of time increases, the size of dt is smaller relative to the timescale. The continuous model uses $dt \to 0$, so as the length of time approaches infinity, the continuous model becomes more accurate.

Both models are deterministic, meaning that they will produce the same results if calculated multiple times for the same input values. This does not reflect reality, where in addition to these patterns there is random 'noise,' or externally-caused fluctuations in population.
Non-deterministic models are called *stochastic*, meaning "random". The most common method of dealing with random noise is to move beyond deterministic models, and instead use stochastic simulations.

Building a stochastic simulation

Mathematical models rely on simple averages: “all organisms reproduce at the same average rate”, “3 children is equivalent to 1 adult”, etc.. Simulations sacrifice this simplicity, but in exchange have the ability to reflect more accurately what is really happening.

Because of their complexity, no two simulations are alike. For this investigation, I have built a simulation that offers the following improvements over the logistic model:

- The organisms are simulated in two-dimensional space, similar to animals on land or bacteria in a petri dish. To feed, the organisms need to move to where food is, and if organisms cluster in one location, they will exhaust the food in that area.
- The organisms reproduce randomly: each organism has a fixed probability of reproducing every time interval. By contrast, the logistic models assume each animal reproduces constantly and smoothly.
- The population is discrete: if an organism has 50% of the food that it needs to survive, 100% of it dies, and it will no longer eat or reproduce.

However, there are still major simplifications:

- The food supply is constant: food is sprinkled around the simulation at a constant rate. This is more akin to bacteria in an industrial fermenter, fed by a machine, than animals on an island. In the latter case, food is itself an organism, and the food reproduction rate is *itself* proportional to the amount of food. In that case, predator-prey dynamics are needed to describe the population, another essay topic by itself.
- All offspring immediately reach adulthood. Again, similar to bacterial cell division.

The computer program I built to run this simulation is included in an appendix, but I'll summarise it here. Every time interval or "turn", each organism follows these steps:

1. Move a fixed distance in a random direction. Expend some food to do so.
2. Eat as much nearby food as possible, until full or no more nearby food.
3. If food level is below a fixed threshold, die.
4. If not dead, pick a number between 0 and 1. If that number is greater than the reproduction probability, create a new organism at a random location.

When the turn is over, new food is sprinkled randomly across the simulation. When running, the simulation looks like this:

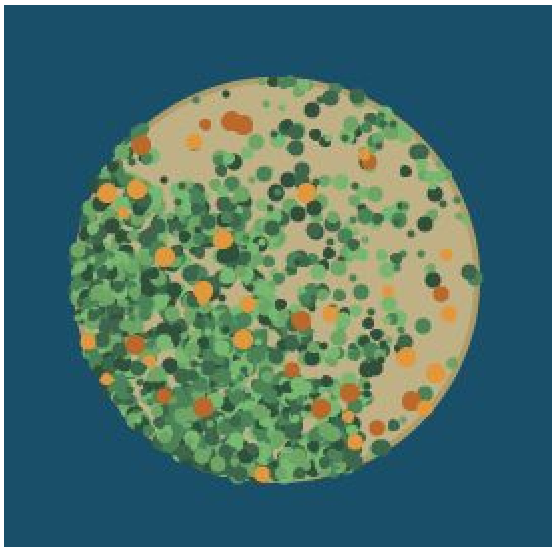


Figure 14: A still frame of the simulation. The orange particles are organisms, and the green particles are food.

In Fig. 14, the bottom-left half of the system has much more food, indicating animals were previously densely distributed in the top-right, where they consumed all the available food and eventually starved. I recorded the population data from the simulation and graphed it for different probabilities of reproduction, each with a starting population of 30, over 200 time-units:

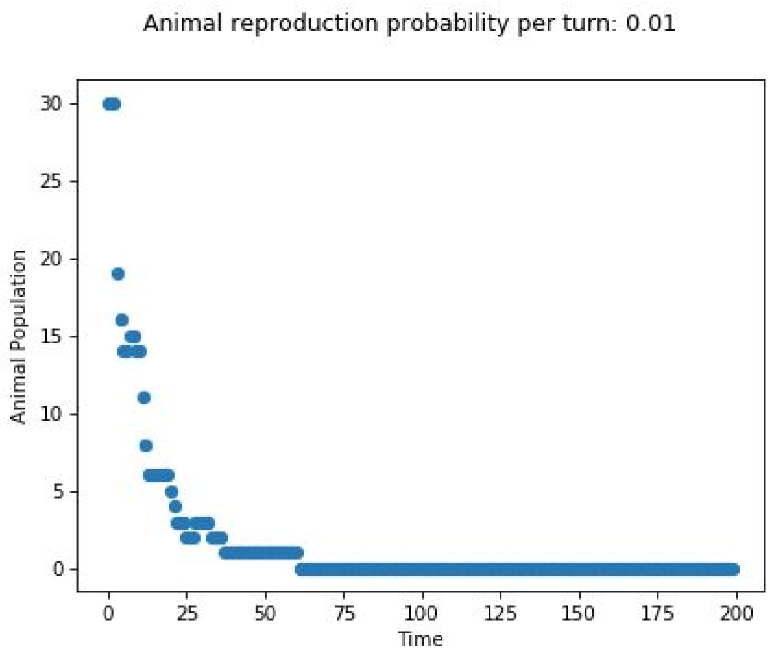


Figure 15: Simulated population over time, animals have a 0.01 probability of reproducing each turn. The population goes extinct due to infertility, similar to the logistic model where $r < 1$.

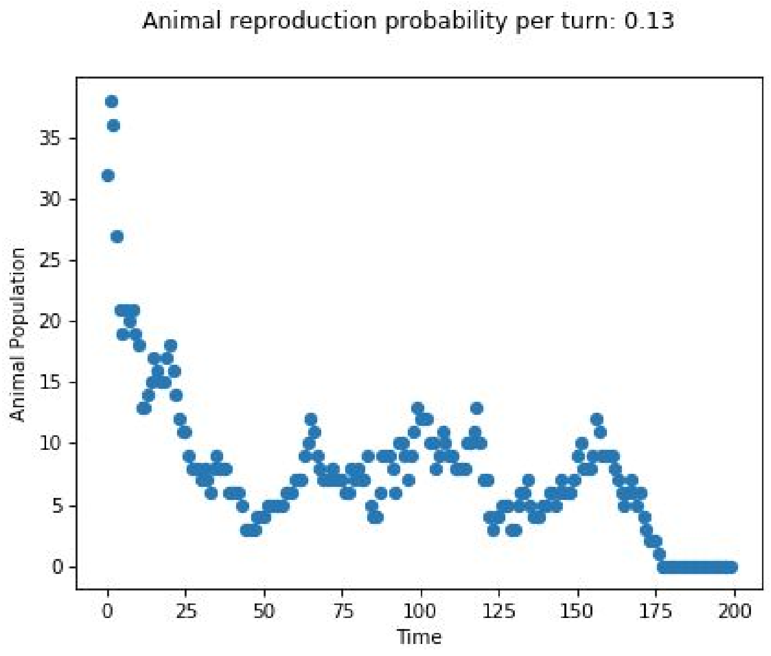


Figure 16: The population becomes extinct, but much more slowly.

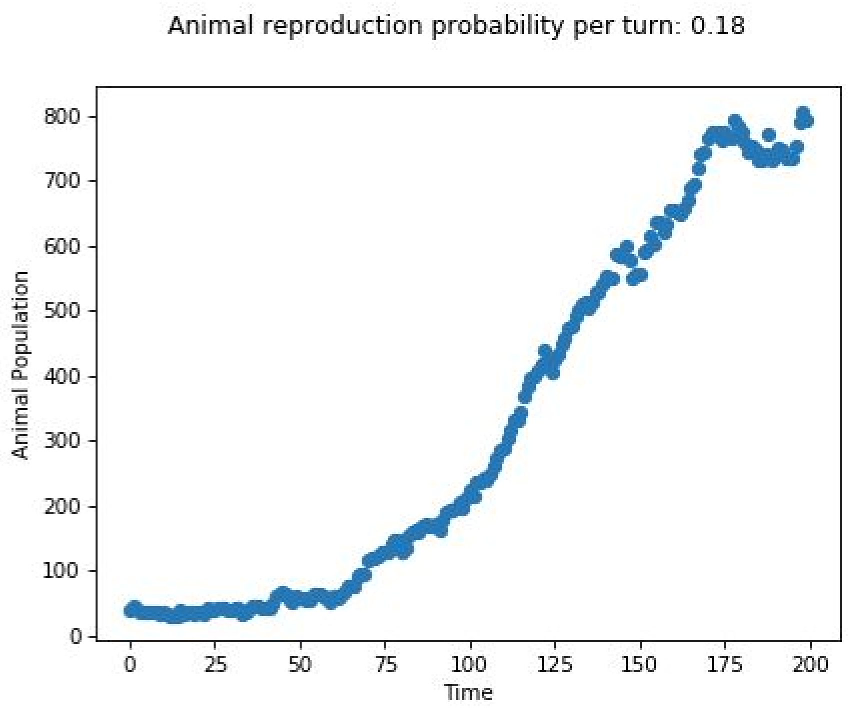


Figure 17: The population grows at an increasing rate.

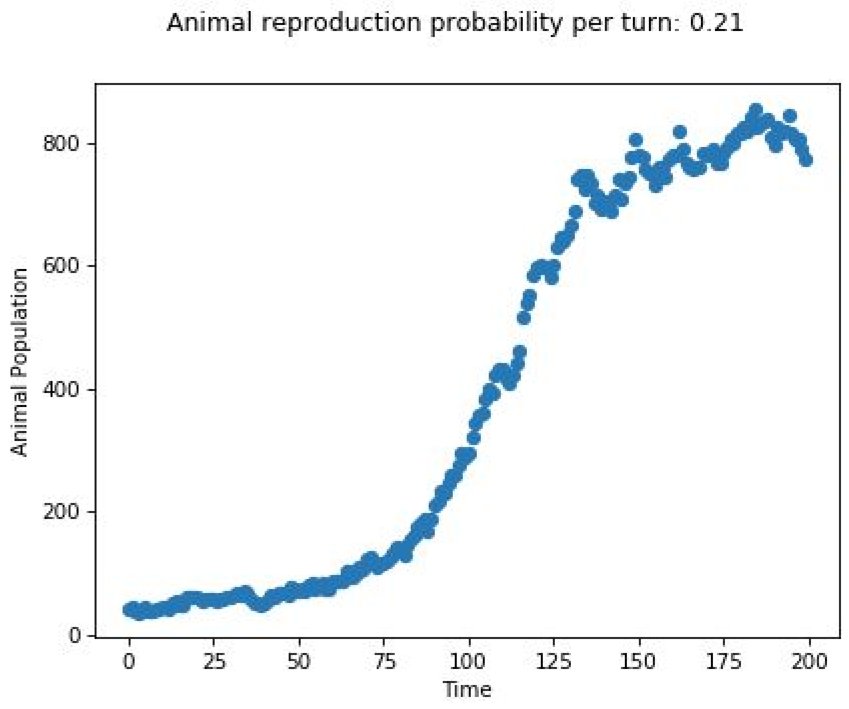


Figure 18: A clear resemblance to the logistic curve is visible, stabilizing close to 800 animals.

My simulation was itself a simplification of reality. While it may describe the population of bacteria in a petri dish, real-world ecosystems, especially those containing macro-organisms such as animals, contain enough factors to render the task of simulating them nearly impossible.

Even so, these results are fascinating. Despite the huge differences between the logistic model and the simulation, both produce very similar-looking growth curves. And the initial growth period (up to $t \approx 110$ in fig. 18), where food is still abundant, bears a close resemblance to the exponential model. This is strong evidence for the accuracy of the mathematical models.

Real-life implications of the logistic model

The logistic model suggests an interesting implication for real-life ecosystems. Since independently increasing either parameter r or K leads to an increase in the stable population, evolutionary theory suggests that species will evolve to maximize r and K.

How can these parameters be increased? Increasing r is relatively simple: the species evolves to reproduce as frequently and efficiently as possible: shorter pregnancy duration, more reliable fertility, etc.

To increase K (the number of organisms that will exhaust the available resources), the species can evolve to either:

1. Increase the quantity of resources to which it has access: for instance, it could gain the ability to run faster for more successful hunting, or a tongue that can reach into crevices to collect insects.

2. Increase the *efficiency* with which it processes these resources: a better digestive tract that can absorb more nutrients, or pack hunting, which reduces food waste.

However, beyond a certain point, theoretically there exists a trade-off between maximising r and maximising K. This is because adaptations that increase K usually take more energy and time to develop, which slows down reproduction. If an animal evolves a better digestive tract, it will need a longer pregnancy for the complex intestines to grow properly.

This trade-off has led to "r/K Selection Theory", the idea that organisms tend to be either "r-strategists" or "K-strategists"[7]. "K-strategists" are more efficient and better adapted to the environment, at the cost of reproduction rate. "r-strategists" reproduce quickly, but each offspring is less likely to survive.

The dandelion is an example of an r-strategist – it releases thousands of tiny, lightweight seeds every year, most of which don't survive[8]. An eagle is an example of a K-strategist – it builds a nest and fetches food for its children, which take months to reach maturity. This limits the rate of reproduction: a parent can't handle the upbringing of more than a few offspring at a time, but since the hatchlings don't need to spend energy on survival, they can grow complex, powerful wings that substantially improve their survival ability as adults.

While few species are strongly K- or r- selected, most tend to fall on a spectrum between being one or the other.

[7] Pianka, 1970
[8] Pianka, 1970

Conclusion

What are the strengths and weaknesses of using mathematical models to understand growth and decline in living populations?

Refining these models is crucial to understanding how changes in an environment will affect populations. This is especially relevant today, as we witness unprecedented climate-related habitat destruction and damage to ecosystems. Although this essay is an exploratory introduction, not a complete overview of Mathematical Ecology, the problems that arose in my research, such as the effect of discrete- and continuous-time models on chaotic behaviour, the trade-off between models and simulations, and the balance between growth rate and stability, are all studied in depth by academics in this field. Research depends heavily on the use of models including the ones that I described in this investigation, as well as simulations like the one I built.

I've demonstrated many shortcomings of mathematical models. They base their predictions on the assumption that growth, reproduction, and death are all smooth, gradual processes. They don't distinguish between one 21-year-old and three 7-year-olds, even though the latter would be unlikely to survive and reproduce by itself. They predict that 0.5 foxes will give birth to 0.5 baby foxes, refusing to consider a species extinct while there is half an organism remaining.

But despite these issues, mathematical models provide valuable insights into the way species behave. If a population reproduces too quickly, it can damage its food source by unsustainable consumption. This is reflected by the chaotic behaviour of the logistic map where $r > 3.57$.

Similarly, if there is enough food for 5.6 animals, only 5 will survive, not 6; this is shown in the behaviour of discrete models.

r/K selection theory offers an explanation as to why some animals have adaptations that sacrifice reproductive rate in favour of greater complexity, or vice versa, and why one must come at the expense of the other. And the asymptotic end-behaviour of the logistic model describes, as is visible in nature, a population that reaches a natural equilibrium.

While models do not come close to accurately encapsulating the entirety of an ecosystem, they remain an indispensable tool for studying population dynamics.

Bibliography:

1. Mora, C., Tittensor, D., Adl, S., Simpson, A. and Worm, B. (2011). How Many Species Are There on Earth and in the Ocean?. *PLoS Biology*, 9(8), p.e1001127.
2. Carlson, B. (2007). *Principles of Regenerative Biology*. Elsevier/Academic Press.
3. Du Sautoy, M. (2017). *The great unknown*. 4th ed. Penguin, pp.49-52.
4. Teschl, G. (2012). *Ordinary differential equations and dynamical systems*. American Mathematical Society.
5. Boeing, G. (2016). Visual Analysis of Nonlinear Dynamical Systems: Chaos, Fractals, Self-Similarity and the Limits of Prediction. *Systems*, 4(4), p.37.
6. Lobri, C., Sari, T. (2015). Migrations in the Rosenzweig-MacArthur model and the "atto-fox" problem. *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées*, pp.95-125.
7. Pianka, E. (1970). On r- and K-Selection. *The American Naturalist*, 104(940), pp.592-597.

Appendix: Simulation Code

```
# This code works as intended on Processing 3.3.6 running Python mode with packages random
and math installed

import random, math
random.seed(a=1) # Generates a constant pseudo-random seed

class logger(object):
    """Class containing storage and parameters for logging population data"""
    def __init__(self):
        # Simulation data containers
        self.log = []
        self.rlog = []
    def reset(self):
        self.foods = [] # Locations of food particles
        self.animals = [] # Locations of animals

        # Logger parameters
        self.graphboth = False # Whether to track food in addition to animals
        self.simulation_speed = 1 # Number of simulation turns for each rendered turn
        self.log_e_s = False # Whether to track the consumption of animals
        self.turn = 0 # Time index
        # Disabling rendering speeds up the simulation
        # Although you lose the pretty visuals that way
        self.render = True
logger = logger() # Initialize logger
logger.reset() # Set all of the parameters to their initial values
class globals():
    """Class containing the starting parameters for the simulation"""
    iap = 30 # Initial animal population (N_0)
    arr = 0.17 # Animal reproduction per turn
    prr = 100 # food replenish rate
    ipp = prr # Initial food population
    asp = 12 # Animal movement speed
    hrs = 25 # Square of hunting radius
    ira = 100 # Island radius
    irs = ira**2 # Island radius squared
    asr = 0.005 # Animal starvation rate
    fpp = 0.04 # nutrients obtained per food
    ifl = 0.1 * fpp * prr / iap # Initial foods level per animal
    agf = True # Animals get full after eating enough
def random_location(island):
    """Generates uniformly distributed points on a circular island"""
    r = sqrt(random.random())*island.radius
    t = random.random()*2*PI
    return r*cos(t)+500, r*sin(t)+350
class Food(object):
    """Class containing code for food particles"""
    def __init__(self, island):
        self.island = island
        self.x, self.y = random_location(self.island)
        self.radius = int(random.random()*3+2)
        self.c = random.choice(['#25523B', '#358856', '#5AAB61', '#62BD69', '#30694B'])
    def render(self):
        stroke(self.c)
        strokeWeight(self.radius)
        ellipse(self.x, self.y, self.radius, self.radius)
class Animal(object):
    """Class containing code for animals"""
    def __init__(self, island, foods=1):
        self.island = island
        self.x, self.y = random_location(self.island)
        self.radius = int(random.random()*3+3)
        self.c = random.choice(['#C46500', '#ED960B'])
        self.foods = foods
    def render(self):
        stroke(self.c)
        strokeWeight(self.radius)
        ellipse(self.x, self.y, self.radius, self.radius)
    def turn(self):
        # Make random decision whether to reproduce (probability is globals.arr)
        if random.random() <= globals.arr:
            # Reproduce, and give offspring half of your food (conservation of food)
            self.island.add_animals(1, self.foods/2)
            self.foods /= 2
        if self.foods < 0:
            # Die of starvation when not enough food
            self.island.animals.remove(self)
        # Move in a random direction by a preset speed (globals.asp)
        x = self.x + globals.asp*(random.random()-0.5)
        y = self.y + globals.asp*(random.random()-0.5)
        # Prevent animals from leaving the island
        if (x-width/2)**2 + (y-height/2)**2 <= self.island.rsq:
            self.x, self.y = x, y
        # Expend food
        self.foods -= globals.asr
        # Check if any food is near by the animal. If so, eat it
        # ...unless Animals Get Full is enabled, in which case only eat it if not full
        if not (globals.agf and self.foods >= 1):
            for food in self.island.foods:
                if (food.x-self.x)**2 + (food.y-self.y)**2 <= globals.hrs:
                    self.island.foods.remove(food)
                    self.foods += globals.fpp
class Island(object):
    # Class storing information about the island and its populations
    def __init__(self, radius):
        self.radius = radius
        self.rsq = (self.radius)**2
        self.foods = []
        self.animals = []
    def add_foods(self, num):
        """Sprinkle food around the map randomly"""
        for i in range(num):
            self.foods.append(Food(self))
    def add_animals(self, num, foods=globals.ifl):
        """Create new animals with specified amounts of food each"""
        for i in range(num):
            self.animals.append(Animal(self, foods))
    def turn(self, skip=0):
        """Code for one simulation tick"""
        if logger.render:
            # Draw the island
            stroke('#b19d5e')
            strokeWeight(3)
            fill(194, 178, 128)
            ellipse(width/2, height/2, self.radius*2, self.radius*2)
        # Calculate food consumption this turn
        e_s = globals.prr-len(self.foods)
        for food in self.foods:
            if logger.render:
                food.render()
            #food.turn()
        if logger.log_e_s:
            # Track food consumption
            try:
                print('Turn {}: e_s = {} ({} / {}): {} per animal'.format(logger.turn,
float(e_s)/globals.prr, e_s, globals.prr, float(e_s)/globals.prr/len(self.animals)))
            except ZeroDivisionError:
                print('Turn {}: No animals.'.format(logger.turn))
        # Add food to make up for the amount eaten
        self.add_foods(e_s)
        # Draw and run code for each animal
        for animal in self.animals:
            if logger.render:
                animal.render()
            animal.turn()
        # Increment time by 1
        logger.turn += 1
        # Run 'invisible' turns without rendering them
        for turn in range(skip):
            e_s = globals.prr-len(self.foods)
            self.add_foods(e_s)
            for animal in self.animals:
                animal.turn()
            logger.turn += 1
def reset():
    """Reset the entire simulation"""
    global island
    island = Island(globals.ira)
    # Add starting populations of food and animals
    island.add_foods(globals.ipp)
    island.add_animals(globals.iap)
    # Reset the population logger
    logger.reset()

# Reset the simulation before starting
reset()
def setup():
    """Set the display window size"""
    size(1000, 700)
def draw():
    """Code to be executed repeatedly"""

    if logger.render:
        background('#014E6D')

    # Run simulation code for one turn
    island.turn(logger.simulation_speed-1)

    # Log and graph animals and foods
    logger.animals.append(len(island.animals))
    if logger.graphboth:
        logger.foods.append(len(island.foods))

    # Code for displaying graphs
    if logger.render:
        fill(255)
        stroke(0)
        strokeWeight(1)
        rect(0, height-100, width, 100)
        strokeWeight(4)
        fill(0)

        # Adjust the graph's vertical scale to fit the data
        if logger.graphboth:
            gscale = max(max(logger.foods), max(logger.animals))
        else:
            gscale = max(logger.animals)
        gfactor = float(100)/gscale

        # Scroll the graph to the left if we run out of horizontal space
        scroll = 0 if len(logger.animals) < width-100 else len(logger.animals)-width+100

        # Render the graph itself
        text(int(gscale), 5, height-85)
        stroke(255, 0, 0)
        for i in range(len(logger.animals)-scroll):
```

```
        point(i, height-logger.animals[i+scroll]*gfactor)
    stroke(0, 0, 255)
    if logger.graphboth:
        for i in range(len(logger.animals)-scroll):
            point(i, height-logger.foods[i+scroll]*gfactor)
    strokeWeight(1)
    for i in range(floor(width/100)):
        line(100*i, height, 100*i, height-100)
    stroke(0)
    line(i, height-100, i, height)

# Run the simulation for iterated values of globals.arr, logging everything
# After 200 turns, save the data and reset the simulation for the next trial
if frameCount % 200 == 0:
    print('Arr = {}'.format(globals.arr))
    logger.log.append(logger.animals)
    logger.rlog.append(globals.arr)

    # Optionally print the log every trial
    #print(logger.log)

    # Increment globals.arr
    globals.arr += 0.01
    reset()

# Once all the trials are complete, end the simulation
if globals.arr > 0.3:
    print(logger.log)
    print('\n')
    print(logger.rlog)
    raise KeyboardInterrupt
```