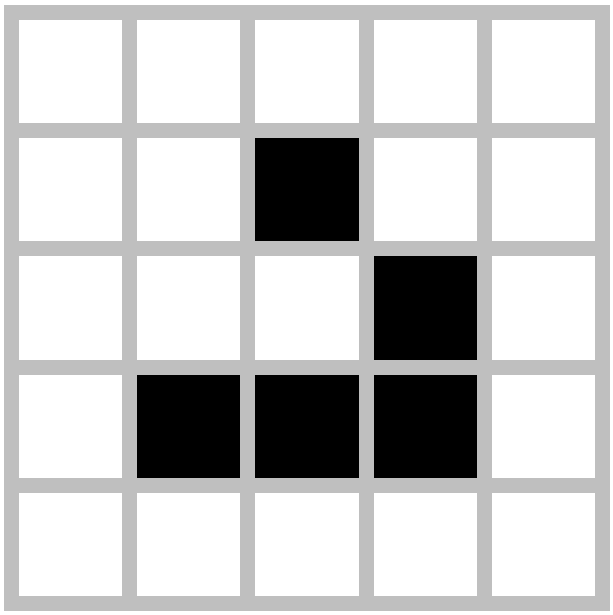
A Predator-Prey Model for Conway's Game of Life



dino

In case you think that anything here is a bit weird, consider this was a math undergrad project and hasn't been worked on since it was turned in.

February 17th, 2024

Contents

1 Introduction **1**

2 The theoretical case **2**

3 The real case **4**

4 Conclusion **7**

Bibliography **8**

Appendix **9**

1 Introduction

John Horton Conway (1937-2020) was a celebrated mathematician, well-known for his work in knot theory, game theory, number theory or maybe most notably in group theory. He has been described as the "world's most charismatic mathematician" [Wikb]. One of his most famous inventions is perhaps the *Game of Life*, which he hated for a part of his life and eventually neither loved or hated. [HC14a]

The Game of Life was invented in 1970, before computers were available enough to run it. As soon as mathematicians and computer scientists got their hands on computers that were powerful enough, the game was executed, studied and celebrated.

It consists of a grid in the plane in which each cell can be either *dead* or *alive*. Then, the state of all cells is updated in each *step* according to a few rules. If the *neighbours* of a cell are the 8 surrounding cells forming a 3×3 square with this cell as its center, these rules are as follows:

1. A living cell with 1 or fewer living neighbours dies of "underpopulation" in the next step.
2. A living cell with 4 or more living neighbours dies of "overpopulation" in the next step.
3. A dead cell with exactly 3 living neighbours becomes alive in the next step.

In any other case, the cell continues to the next step in the same state.

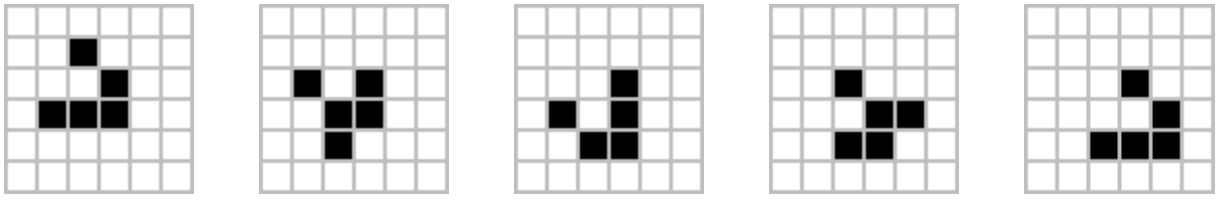


Figure 1: The four stages of a *glider*. After a fourth step, the glider returns to the first stage but translated in a diagonal direction and this cycle repeats. This way, it "glides" across the plane. Living cells are portrayed in black and dead cells in white. (Edited from [Muz22]. Same for cover.)

The Game of Life is inspired by Stanislaw Ulam's and von Neumann's idea of self-replicating automata, but the rules of the game are also partly drawn from real life: they are an oversimplification of how organic life reproduces and dies [HC14b]. For this reason, the object of this project is to study the Game of Life using a Predator-Prey model with the idea that living cells are "preying on" dead cells by reproducing.

This model was proposed separately by Alfred J. Lotka (1880-1949) in 1920 and Vito Volterra (1860-1940) in 1926 and is also known as the *Lotka-Volterra equations*. It is a system of differential equations that aims to describe how the populations of a "predator" species and a "prey" species interact and

evolve. This model shows how the periodic fluctuations of these populations arise and behave from a mathematical standpoint.

The usual examples of this model are those of actual predator and prey species, like foxes and rabbits; or deer and vegetation. However, as both Volterra and Lotka proposed themselves, the model can be used to describe many other phenomena, like chemical reactions between two compounds; or even the contraction and relaxation of muscles.

With the foxes and rabbits example in mind, the Lotka-Volterra equations are

$$\begin{aligned} \frac{dR}{dt} &= aR - bRF, \\ \frac{dF}{dt} &= -cF + dRF \end{aligned} \tag{1}$$

where R and F denote the population or mass of rabbits and foxes respectively; a, b, c, d are positive parameters; and the differentiation is with respect to time t. In the rabbits' equation, the R term is multiplied by a positive coefficient because rabbits are expected to reproduce on their own, while the F coefficient in the foxes' equation is negative since these are expected to die of overpopulation. Both equations share the "interaction" term RF which has negative coefficient for the rabbits since these are hunted by the foxes; and has positive coefficient for the foxes because these reproduce or grow when fed by the rabbits. These parameters are not necessarily constant, as they could vary with time or for a variety of reasons. [**notes**]

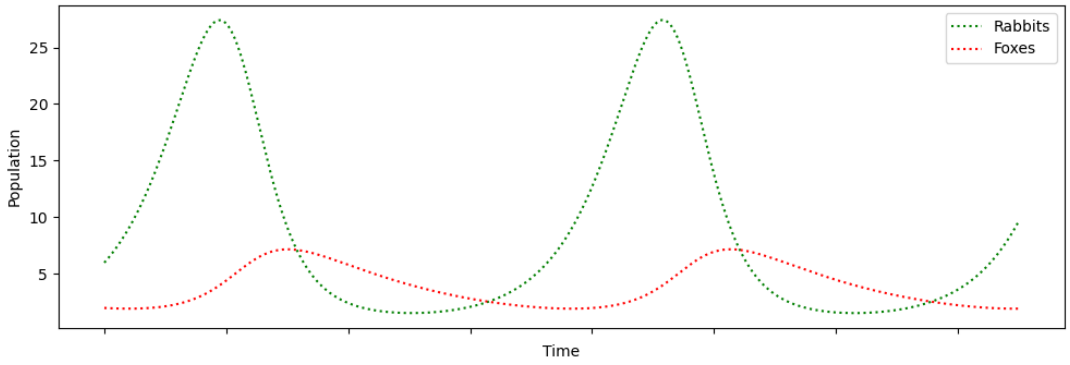


Figure 2: Example plot of numerical solutions to Equations 1 using Euler's method with $a = 0.4$, $b = 0.1$, $c = 0.09$ and $d = 0.01$.

2 The theoretical case

Conway's Game of Life is well-known for the complexity that arises from its simple rules. It is Turing complete and could be used to make any kind of logical machine. Interesting patterns are found frequently and the game serves as inspiration for multiple projects and challenges (such as this project), both serious and recreational.

In particular, multiple structures or patterns are known that behave in a variety of interesting ways: Some disappear immediately; some are constant or periodic; others serve as "factories" or "guns" for other moving structures. Because of this, it might seem like there is no general rule for the evolution of the population of living cells. However, these interesting structures are rare, in the sense that they need to be designed and the slightest difference could completely change their evolution in a few steps. In general, a random pattern usually dissolves and spreads into smaller patterns like *gliders* (Figure 1) or 2×2 cubes that are constant. [Wika]

The hypothesis in this project is that, given some random pattern, the densities of living and dead cells are expected to behave following an analogue of the Lotka-Volterra equations. Reading the rules of

the game from the standpoint of the Predator-Prey model, we notice that rule 3 could be read as living cells preying on dead cells and rule 2 has living cells dying of overpopulation. This sounds like the foxes in the previous example. However, rule 1 could be read as dead cells preying on living cells, so both living and dead cells are both prey and predator in this case, unlike in the usual cases. This would mean that we allow for non-positive values for the parameters in Equation 1.

To study the game's behaviour, a finite $n \times n$ grid is simulated. We write A and D for the number or density of living (alive) and dead cells respectively. We then have the equations

$$\begin{aligned} \frac{dA}{dt} &= aA + bAD \\ \frac{dD}{dt} &= cD + dAD \end{aligned} \tag{2}$$

for some $a,b,c,d \in \mathbb{R}$. Obviously, the game is run in discrete steps of time, so it doesn't make much sense to talk about derivatives with respect to time. A and D also take discrete values; are integers if they represent the number of cells; and are actually bounded in this finite grid. We may talk about the theoretical values of A and D, which are governed by these equations and are functions of continuous time; and the real empirical values of A and D, which are obtained experimentally and are discrete. We would then expect the experimental values to follow the theoretical ones under our hypothesis.

Let's assume that A and D are the densities of cells. That is, the number of alive or dead cells divided by n^2. Since all cells are either dead or alive, we must have the conditions $A,D \in [0,1]$ and $A+D=1$. These constrains simplify our model, since we have

$$D = 1 - A, \quad \frac{dD}{dt} = -\frac{dA}{dt} \tag{3}$$

Then, we can simplify Equations 2 into a single differential equation and study only A:

$$\frac{dA}{dt} = aA + bAD = aA + bA(1-A) = (a+b)A - bA^2 \tag{4}$$

Since $a,b \in \mathbb{R}$, we can rewrite $a+b =: \alpha$ and $b =: \beta$ to have a general quadratic equation

$$\frac{dA}{dt} = \alpha A - \beta A^2 \tag{5}$$

where $\alpha,\beta \in \mathbb{R}$. We have β following a negative sign because we will see that in practice these values are positive. After another change of variables with $r := \alpha$ and $K := \frac{\alpha}{\beta}$, we can write this equation again as

$$\frac{dA}{dt} = rA\left(1 - \frac{A}{K}\right) \tag{6}$$

This is a well-known equation called the *logistic equation*, the *law of population growth* or the *Verhulst-Pearl equation*. It was originally proposed by Pierre-François Verhulst (1804-1849) in 1838 and consequently rediscovered by multiple mathematicians, including Lotka in 1925. It models the growth of an exponentially growing populations with limited resources. r is known as the *growth rate* and K is the *carrying capacity*, which is the maximum population of an organism the environment can sustain. The analytical solution to this equation is given by

$$A(t) = \frac{KA_0e^{rt}}{(K-A_0)+A_0e^{rt}} = \frac{\alpha A_0 e^{\alpha t}}{(\alpha - \beta A_0) + \beta A_0 e^{\alpha t}} \tag{7}$$

where $A_0 := A(0)$ is the initial value of the population. We can easily see from Equation 6 that the system finds its equilibrium at $A=0$ and $A=K$. We will also have $A(t) \xrightarrow{t\to\infty} K$ if $A \neq 0$. These solutions are called *sigmoid functions* or *logistic functions*. This model, along with these functions, is used in a variety of fields, including machine learning, economics and even linguistics. [Wei] [Wikc]

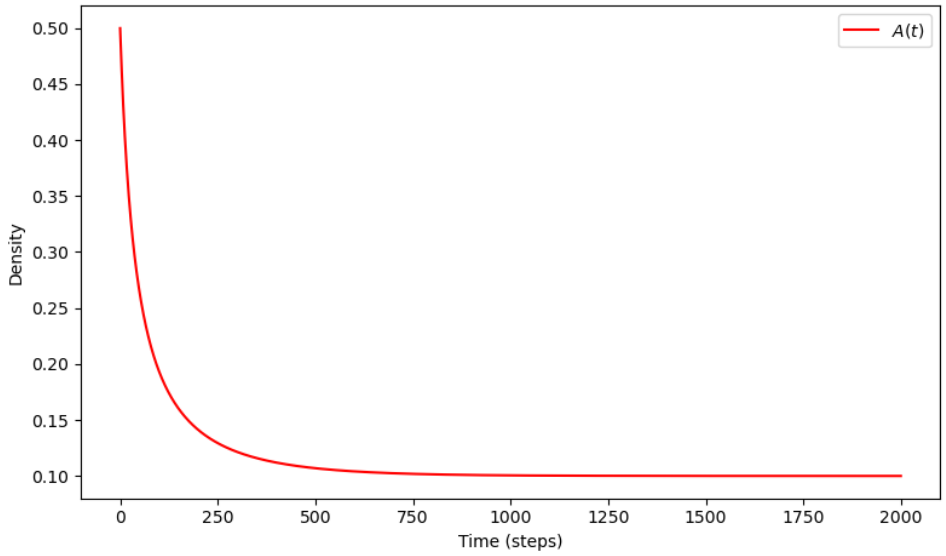


Figure 3: Plot of numerical solution $A(t)$ to Equation 5 using Euler's method with $A_0 = 50\%$, $\alpha = 0.005$ and $\beta = 0.05$. We can see that the solution converges to $K = \frac{\alpha}{\beta} = 0.1 = 10\%$.

Looking at Figure 3, we see how the population $A(t)$ monotonically converges to K. At first, the population decreases sharply. The rate at which it decreases slowly lowers and approaches 0 as A approaches K. D would behave in a similar manner but increasing and converge to $1-K$.

This is the case where $A_0 > K$. If $A_0 < K$, $A(t)$ would again converge similarly to K but increasing (and D would decrease to $1-K$). If $A_0 = K$, then both A and D would be constant.

3 The real case

We can populate our grid with an initial density and run the game to study the experimental values. Figure 4 shows the first 100 steps of some runs with different initial densities. The grid is initially populated by letting each cell be alive with a probability equal to the desired density, so the initial densities in these runs only approximate the desired ones.

We see in Figure 4 how the experimental densities resemble the theoretical densities from Figure 3. However, we notice a different behaviour at the start of the runs (Figure 5): Runs with initial densities around 30% have their density go up with the first step and oscillate with the few following steps, while runs with high initial densities will have their density drop to low densities in a single step. This is to be expected as many cells die immediately of overpopulation, but does not match the behaviour of logistic growth as the density starts increasing after this drop.

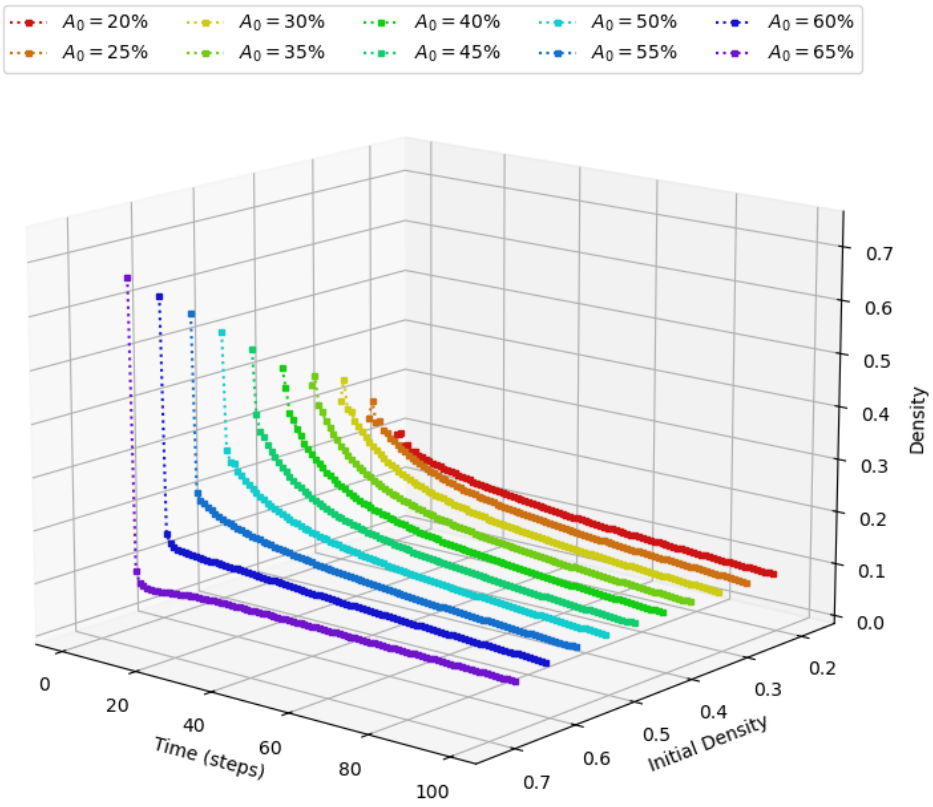


Figure 4: Plot of first 100 steps of 10 different runs of the game with $n = 4000$. Ranging from 20% initial density to 65%.

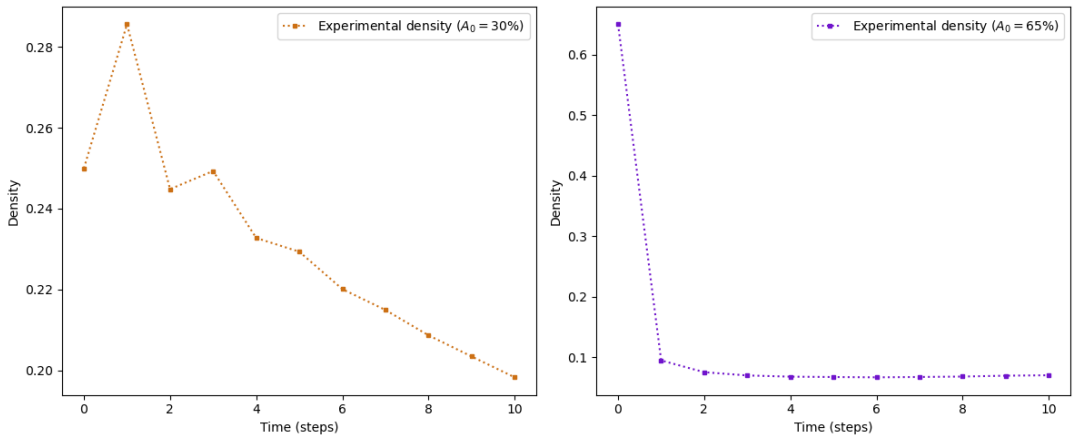


Figure 5: Close up of first 10 steps of the runs with $A_0 = 25\%$ and $A_0 = 65\%$ from Figure 4.

Let's look at one of these runs on its own. We simulate a run $A_e(t)$ with $n = 4000$ and $A_0 = 50\%$; and look at the steps 50 to 300. We can use $A_e(t) - A_e(t-1)$ as an estimate for $\frac{dA}{dt}$. Plotting these differences against $A_e(t)$ like in Figure 6, we expect them to follow a parabola according to equation 5. We could find the best values for α and β using the least squares method, which gives a quadratic fit. We obtain the values $\alpha = 0.0054802$ and $\beta = 0.0978101$,

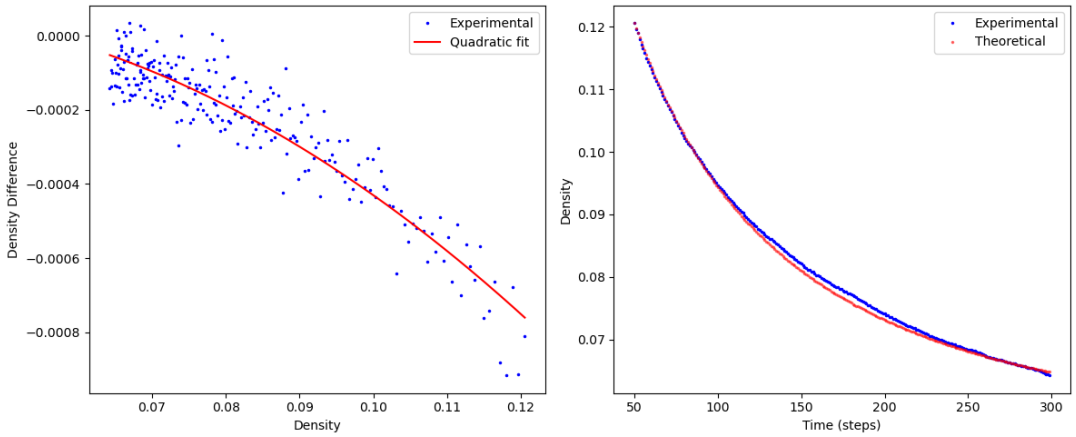


Figure 6: Plots of $A_e(t) - A_e(t-1)$ against $A_e(t)$ (blue, left); and $A_e(t)$ against t with t from 50 to 300 (blue, right). Fitting a parabola with zero intercept to the left plot (red, left) we obtain the values $\alpha = 0.0054802$ and $\beta = 0.0978101$ and find a numerical solution $A(t)$ to Equation 5 with Euler's method with $A(50) = A_e(50)$ (red, right).

It looks like the run does follow our hypothetical behaviour during these steps. According to these values, $A(t)$ should slowly converge to $K = \frac{\alpha}{\beta} \approx 0.0560 = 5.60\%$ and not drop under this value. However, in the following steps, the run drops well under 5.60%. After 2000 steps, the density has dropped below 4% and doesn't seem to slow down either (Figure 7).

Repeating this process for steps from 300 to 2000 gives values $\alpha = 0.0011031$ and $\beta = 0.0330395$, which have $K \approx 0.0334 = 3.34\%$ Finding a numerical solution like before that matches $A(300) = A_e(300)$ fits the real densities after 300 steps but doesn't match for the first 300 steps (Figure 8).

Will the run match this new $A(t)$ after the 2000th step? We can run the game for longer and check. Taking 8000 more steps, we see in Figure 9 that the same thing as in Figure 7 happens again: The real density separates from $A(t)$ after $t = 2000$ and drops below K again.

It looks like $A_e(t)$ does not match our hypothesis in the long run. Trying to find a logistic function $A(t)$ that behaves like $A_e(t)$ in some interval of time won't seem to work outside of this interval.

We have been taking constant values α and β. As mentioned earlier, this is not necessarily the case in many applications of the Predator-Prey model. It is possible that $A_e(t)$ does follow Equation 5 but for variable values of α and β.

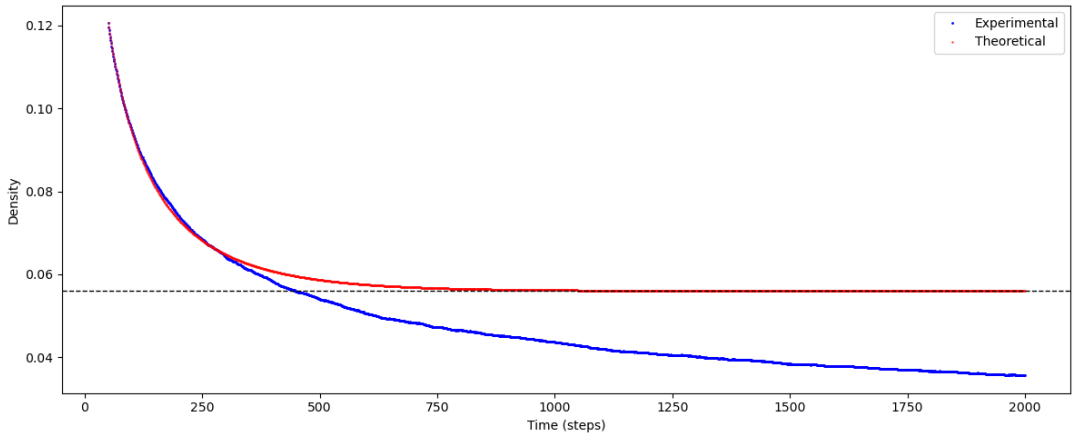


Figure 7: Continuation of Figure 6 up to 2000 steps. $A(t)$ converges to $K \approx 0.0560$ (black) while $A_e(t)$ separates from the $A(t)$ as early as $t = 300$ and continues to decrease.

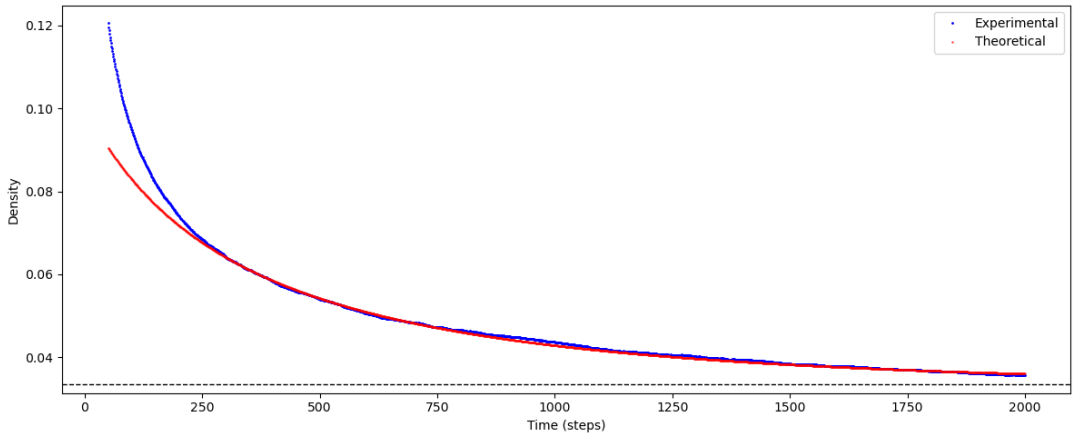


Figure 8: Plot of the $A(t)$ (blue) and $A_e(t)$ (red) with the new values α and β and imposing $A(300) = A_3(300)$. $A(t)$ will converge at $K \approx 0.0334$ (black).

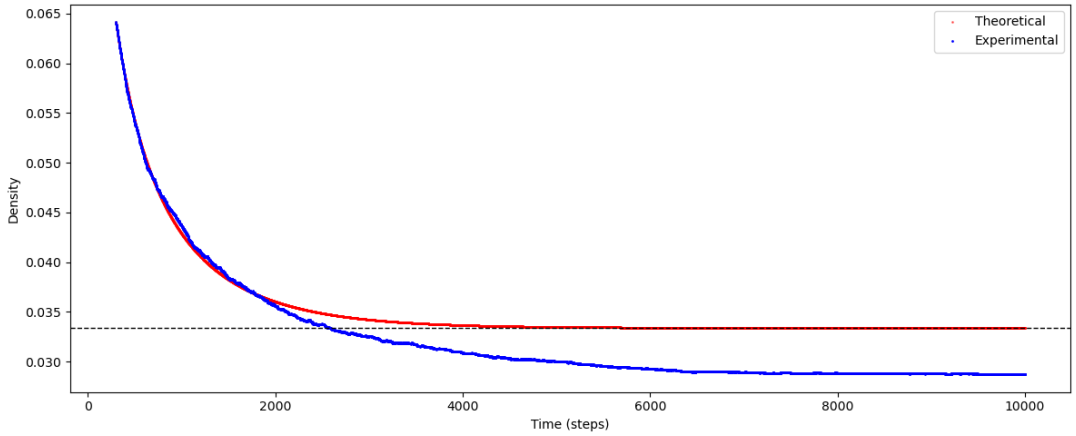


Figure 9: Continuation of Figure 8 up to $t = 10000$. We see the same problem as in Figure 7.

4 Conclusion

We have proposed a Predator-Prey model for Conway's Game of Life with the justification that the game's rules were made to mimic the behaviour of real population growth. Studying the density of living and dead cells of a game on a finite grid, we realize both species act as both predator and prey; and that the equations of the Predator-Prey model simplify to that of logistic growth.

Looking at real runs of the Game of Life, we conclude that this model does not fully work, or at least not for constant coefficients of the equations. However, it does seem to match the behaviour of the game in a finite period of time.

Bibliography

[HC14a] Brady Haran and John H. Conway. *Does John Conway Hate His Game of Life?* Mar. 3, 2014. URL: https://www.youtube.com/watch?v=E8kUJL04ELA. Uploaded to Numberphile [Accessed: 2024/02/13].

[HC14b] Brady Haran and John H. Conway. *Inventing Game of Life (John Conway) - Numberphile*. Mar. 6, 2014. URL: https://www.youtube.com/watch?v=R9Plq-D1gEk. Uploaded to Numberphile [Accessed: 2024/02/13].

[Wei] Eric W. Weisstein. *Logistic Equation*. URL: https://mathworld.wolfram.com/LogisticEquation.html. [Accessed: 2024/02/14].

[Wika] Wikipedia. *Conway's Game of Life*. URL: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life. [Accessed: 2024/02/13].

[Wikb] Wikipedia. *John Horton Conway*. URL: https://en.wikipedia.org/wiki/John_Horton_Conway. [Accessed: 2024/02/13].

[Wikc] Wikipedia. *Logistic Function*. URL: https://en.wikipedia.org/wiki/Logistic_function. [Accessed: 2024/02/14].

Figures

[Muz22] Muzik. *Glider*. Jan. 24, 2022. URL: https://conwaylife.com/wiki/File:Glider.png. [Accessed: 2024/02/13].

All plots were made with matplotlib.

Appendix

The following Java code was used to generate the experimental densities:

```java
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

public class Conway {

  public static void main(String[] args) {
    if (args.length != 7) {
      System.out.println("Was expecting seven arguments (grid size, time, initial, final, step, passes, filename)");
      System.exit(1);
    }
    int n = Integer.parseInt(args[0]);

    int T = Integer.parseInt(args[1]);

    float initial = Float.parseFloat(args[2]);
    float end = Float.parseFloat(args[3]);
    float step = Float.parseFloat(args[4]);

    int passes = Integer.parseInt(args[5]);

    for (int i = 1; i <= passes; i++)
      for (float d = initial; d <= end; d += step) {
        System.out.println("Starting " + i + "th pass of density " + d*100 + "%");
        new ConwayDoer(n, T, d, args[6] + "-" + ((int) (d*100)) + "-" + i);
      }
  }

}

class ConwayDoer {

  int n;
  float tol = 0.0001f;

  boolean[][] states;

  public ConwayDoer(int n, int T, float density, String name) {
    this.n = n;
    this.states = new boolean[n][n];

    Random random = new Random();
    for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) {
      states[i][j] = random.nextFloat() < density;
    }

    try {
      FileWriter fw = new FileWriter(name + ".csv");
      fw.write("time, density, difference\n");
      float prev = 0;
      float current;

      long start = System.currentTimeMillis();

      // Steps
      for (int t = 0; t < T; t++) {
        current = ((float) sum()) / n / n;
        fw.write(t + ", " + current + ", " + (current - prev) + "\n");
```

```
59          prev = current;
60
61          if (current < tol) {
62            System.out.println("Hit 0! Finishing early!");
63            fw.close();
64            return;
65          }
66
67          states = step();
68
69          if (t % 20 == 0) {
70            float time = (System.currentTimeMillis() - start)/1000.0f;
71            System.out.println("progress: " + t + "/" + T + " steps (" + 100.0*t/T
      + "%) - elapsed time: " + time + "s  - estimated remaining time: " + (t ==
     0 ? "??" : time*(T-t)/t) + "s - current density: " + current*100 + "%");
72          }
73        }
74
75        System.out.println(T + "/" + T + " steps (100%)");
76        System.out.println("Finished!");
77        fw.close();
78      } catch (IOException e) {
79        e.printStackTrace();
80      }
81    }
82
83    boolean[][] step() {
84      boolean[][] next = new boolean[n][n];
85      for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) {
86        int neighbours = neighbours(i, j);
87        next[i][j] = neighbours == 3 || (states[i][j] && neighbours == 2);
88      }
89
90      return next;
91    }
92
93    boolean getState(int i, int j) {
94      i %= n;
95      j %= n;
96      if (i < 0) i += n;
97      if (j < 0) j += n;
98      return states[i][j];
99    }
100
101   int neighbours(int i, int j) {
102     int neighbours = 0;
103       for (int a = -1; a <= 1; a++) for (int b = -1; b <= 1; b++) {
104         if (a != 0 || b != 0) neighbours += getState(i+a, j+b) ? 1 : 0;
105       }
106       return neighbours;
107   }
108
109   int sum() {
110     int m = 0;
111     for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) {
112       m += states[i][j] ? 1 : 0;
113     }
114     return m;
115   }
116 }
```