Game of Life

Edward Akapo and Hoang Tam Nhu

January 5, 2022

Contents

1 Introduction

The purpose of this assignment was to research the Conways' game of life theory, a two dimensional grid of square cells that are either alive or dead according to a specific set of rules. Each cell interacts with the eight cells that are vertically horizontally or diagonally adjacent to it, these are called its neighbours [1].

Moreover, we were to explain and modify the code given to us to create multiple simulations and create pseudo-population statistics with single or multiple species.

2 Short Program Explanation

Instead of using a full program to create Conway's Game of Life simulation, a shorter and more condensed program can be written as

```
function Game(n,m,t)

//Creates B as a random matrix n by m where the only values are 0 and 1
B = round(rand(n,m));

//Creates a loop of time steps starting from 1 and end at the value of t
entered
for time=1:t

    A = B;

        //Sets i as the value run from 1 to n, in this case i acts as the
        number of rows in nxm matrix
        for i=1:n

            //Sets j as the value run from 1 to m, in this case j acts as the
            number of columns in nxm matrix
             for j=1:m

                //Define the row neighbours where they are the above and
                below rows of a particular value i
                 cola = [i-1,i,i+1];

                //Define the neighbours in case the value is in row 1, then
                its neighbours is row n
                 cola(cola<1) = n;

                //Define the neighbours in case the value is in row n, then
                its neighbours is row 1
                 cola(cola>n) = 1;
                //Define the column neighbours where they are on the right
                and left columns of a particular value of j
                 colb = [j-1,j,j+1];

                //Define the neighbours in case the value is in column 1,
                then its neighbours is column m
                 colb(colb<1) = m;
                //Define the neighbours in case the value is in column m,
                then its neighbours is column 1
                 colb(colb>m) = 1;

            //Define the number of alive cells as R,
             R=sum(sum(A(cola,colb)));

            //Define the rules: If two or three alive neighbours and the cell is
             alive then it lives on, if there are three neighbours alive and the
            cell is not alive then it becomes a live cell
             B(i,j) = ((R==3 || R==4) && A(i,j)) || ( R==3 &&   B(i,j));

            end
        end
        spy(B);
        pause(0.05);
    end
```

3 Statistics

3.1 Population Graphs

By setting B as a 50 by 50 matrix over 100 time steps, the cells' behaviours, whether alive or dead are portrayed in Figure 1, this describes the population density over time of the cells for a single simulation where the x-axis presents time steps and the y-axis presents the population of living cells. It can be seen that, even though the graph fluctuates slightly over the time interval, its overall trend is a decrease in total population.

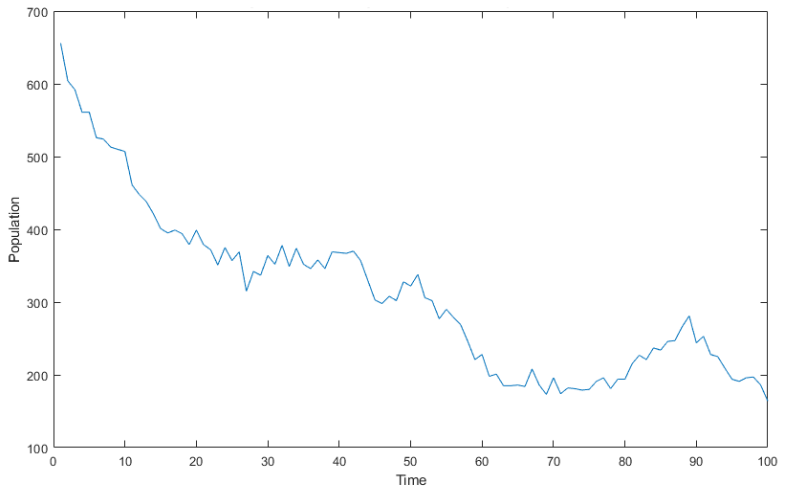

Figure 1: Single simulation plot

By increasing the number of simulations to twenty, Figure 2 presents the change in population over 100 time steps was created below. Through running multiple simulations, it can be seen that for every simulation, the initial population densities are nearly the same, at approximately 620 to 720 living cells. Even though an overall decrease in population occurs throughout all simulations, the living cells varies greatly towards the end from 180 to 320.

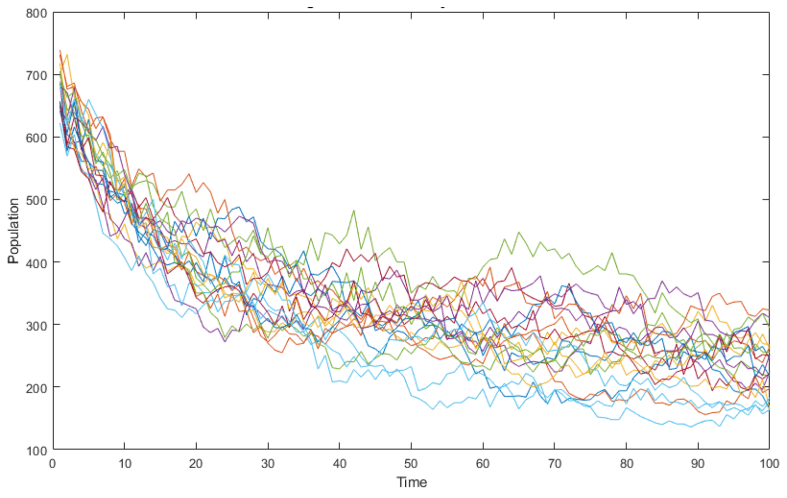

Figure 2: 20 simulations plot

With Figure 3 illustrates the average trend over twenty simulations, Figure 4 shows the average and the graph's standard deviations. For twenty simulations, the average beginning population is 680 while after 100 time steps, it is seen that only 250 survive. By looking at the red lines, the errors of the average plot, this is proven that towards the end the errors is getting higher.

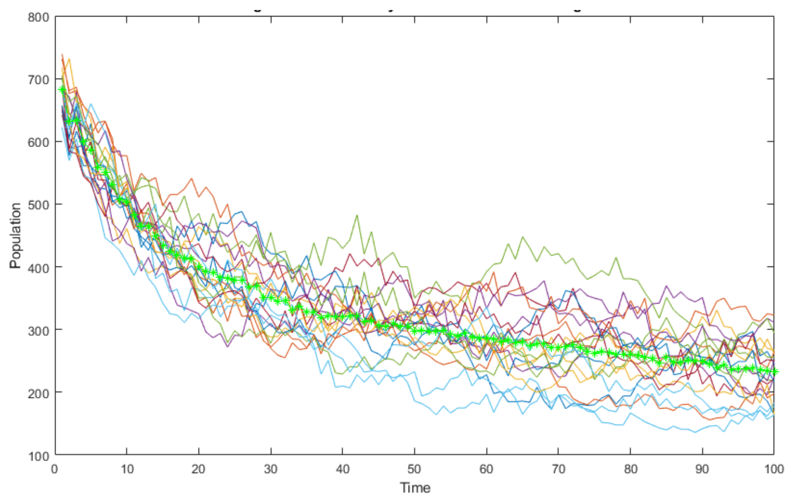

Figure 3: 20 simulations plot and mean population density

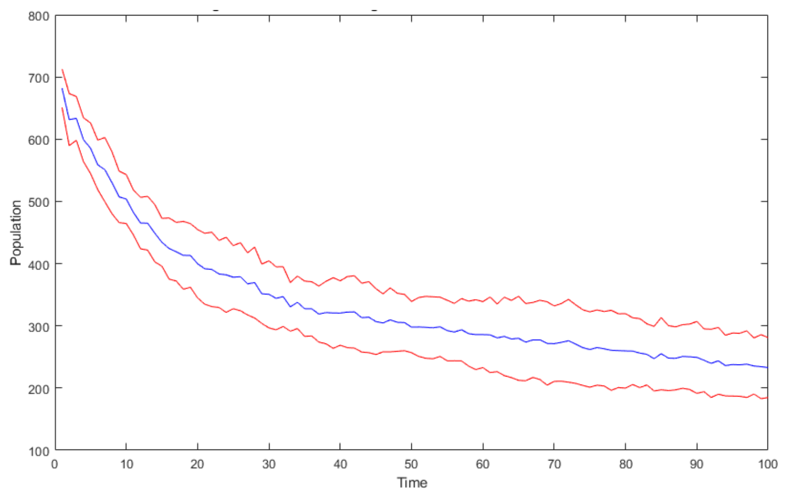

Figure 4: Mean population density and standard deviations

3.2 Matrix Size Alteration

Figures25, 26, 27, and 5 were created by increasing the size of the matrix n x m from 50x50 to 200x200 while the number of time steps kept unchanged with 100. As expected, the number of living cells of this species increases as the size of all twenty simulations increases. However, when comparing to a 50x50 simulating matrix, the average plot of population density over time decreases exponentially without much fluctuation. Moreover, by looking at Figure 5, its errors over twenty simulations are relatively smaller with a bigger matrix. Bigger matrix causes higher in population. Therefore, the population when changing the simulating matrix is higher throughout 100 time steps, where 3800 cells survive.

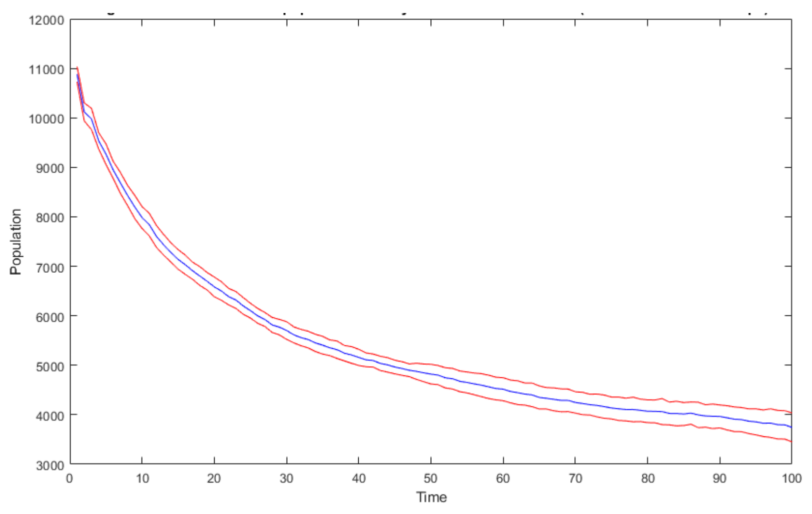

Figure 5: Mean and std deviation plot with matrix set to 200x200

3.3 Time Step Alteration

For Figures 6, 22, 23, and 24 which can be seen below and in Appendix B.1, the size of the matrix was kept at 50x50, while the time steps were increased from 100 to 300. Even though the graph of its average change in population density and errors looks similar to in Figure 4, but it fluctuates more throughout the simulations. Moreover, by looking at previous plot, due to the nature of population's decreasing over time, it is expected that the higher the time steps, the lower the number of living cells. This is proven when the average initial population for normal simulations and when increases the time steps are approximately 680 cells. However, for 100 time steps, only 250 cells of this species survive. while after 300 time steps, the population decreases to 150. Hence, the population will keep decreasing over time but the rate of change slows down as the number of time steps increases.

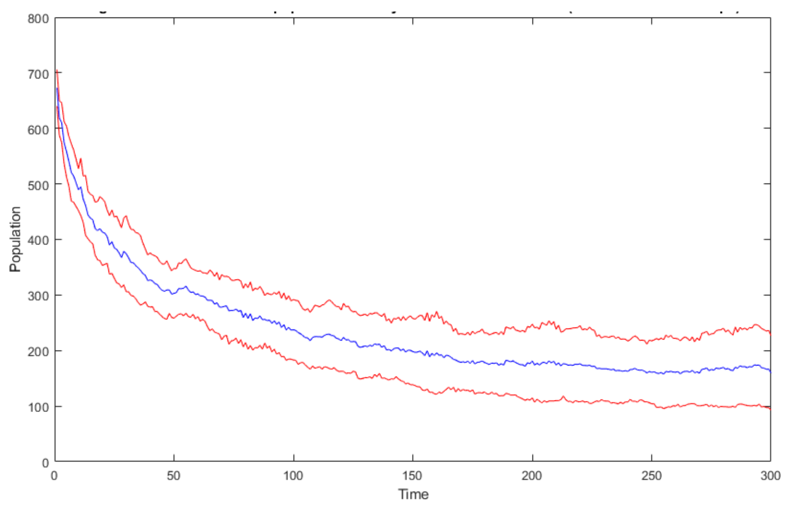

Figure 6: Mean and standard deviations for time increased to 300 time steps

3.4 Removing Periodic Boundaries

As can be seen in the pseudo-code and its explanation in Part 2, it is defined that whenever the cell locates at the corners of the matrix or on the outer sides of the simulating matrix, its neighbours is the one next to it but on the opposite sides. This makes the interacting space for this species looks like the shape of a doughnut. However, when removing these periodic boundaries by rewriting the code to listings 3 , the neighbours are just the one next to it which makes it simulating in a two dimensional space. Figures 28 and 29 in Appendix B.3 describes single and twenty simulations of the species' change in population density over 100 time steps for a 50x50 matrix, respectively. Figure 7 below illustrates the average plot over twenty simulations and its error. Comparing with Figure 4, these two plots look identical with the same number of surviving cells, 250. However, when observing their behaviours in each time step, for cells located at the corners of the matrix, it is harder for them to becomes a live cell when they died. It takes a lot more time for these cells to reincarnate and evolve to the next generation. Moreover, it makes their living environment become an in cohesive space and the cells outer of the matrix becomes weak links.

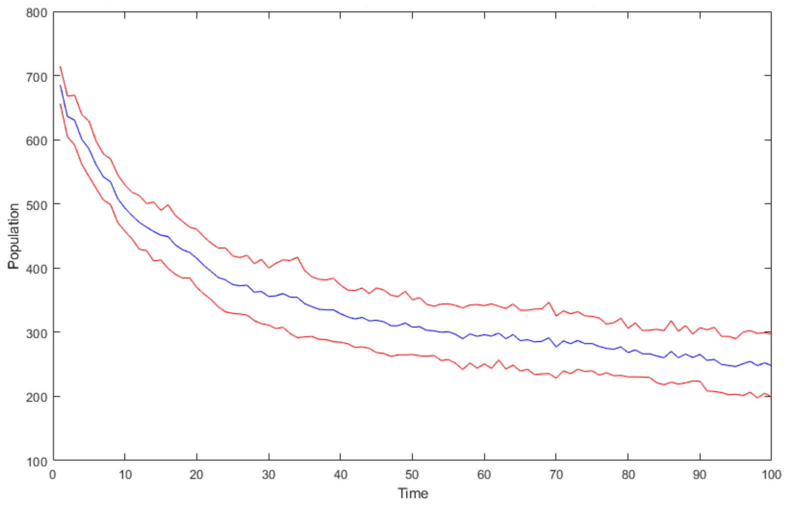

Figure 7: Mean and standard deviation with the boundaries removed

3.5 Rules Alteration

In Conway's Game of Life, the rules are very strict. The cell will die if it has more than three neighbours or less than two neighbours. A dead cell can becomes alive if they have exactly three live neighbours. By changing the rules, it can be seen that how these rules are finite.

Firstly, the rules are changed so that the cells will live if there only have one or two neighbours and it will becomes alive if there are three living neighbours. Figure 8 below describes the population of this species for twenty simulations when the matrix is set as 50x50 over 100 time steps. For the first five time steps, the population graph plummets to 0 from approximately 680 and flattens out until it reaches 100 time steps . Over twenty simulations, even though the number of living cells sometimes remains at 10, it remains unchanged. This is because there are not enough neighbours around them to dies or comes back to life.

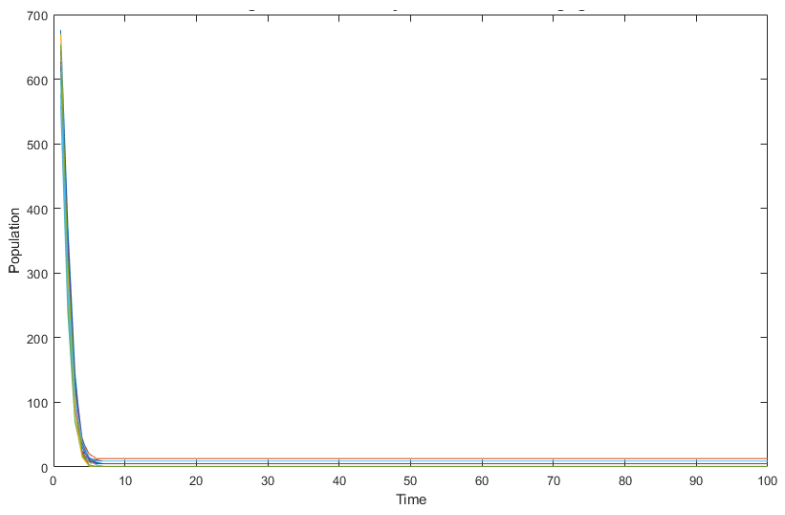

Figure 8: 20 simulations plot with the rules setting lower

Secondly, the rules are now changed by setting it higher than what Conway did as can be seen in Appendix A.5. In this case, the cells will survive if there are three to four neighbours around them and becomes alive if there are five live neighbours. Similar to the previous part where the rules were set lower, the plot in Figure 9 looks identical. As the rules change, only a small number of cells satisfy the rules which causes the drastically decreases in its population. The plot decreases from 680 to nearly 0 living cells in the first five time steps and remains unchanged.

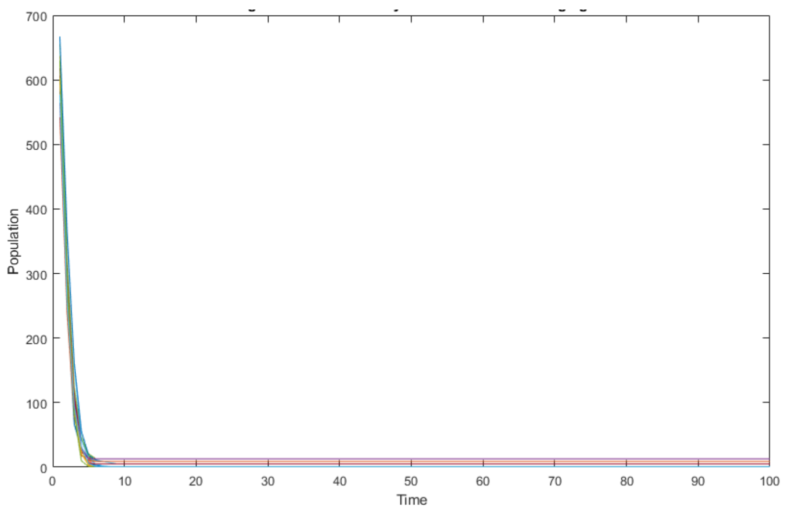

Figure 9: 20 simulations plot with the rules setting higher

To conclude, it is proven that only the original rules make the Game of Life works. By altering it, either set it higher or lower, causes the whole systems collapse because every cells died instantly. Therefore, in order to study their behaviours for a specific period of time, it is important to keep the rules unchanged.

3.6 Sources and Sinks

For a normal Game of Life simulation, the state of each cell is determined by its interaction with others in an environment and the rules set up. However, when change its state, either to make it always alive or dead, their neighbours behaviours alter altogether. Sources are defined as the cells that always alive regarding the rules and sinks are the cells that always dead. By setting up the code as in Appendix A.6, the source is set up at the point (5, 5) of the 50x50 matrix while the sink is at (45, 45) of the same matrix. When the the program is ran twenty simulations for 100 time steps, it can easily observed that the source is on the top left corner and the sink is on the bottom right corner of the matrix. As each time steps, it is witnessed that the source's neighbours either always alive or becomes a live cell in a short time. This area always consists of living cells. Conversely, at the bottom right corner, where exists sink, it becomes a dead area after approximately 50 time steps. Due to the normal population decreases over each time step and the sink cell, its neighbours have a harder time to staying alive or comes back to life. Therefore, the area of sources live while cells near sinks die.

4 Population and Evolution

After the program is modified as seen in Appendix A.7, there would exists another species, red, interacting with each other and with the previous one, blue. After running the program, the behaviours of red and blue and how they interacts with each other can be seen. Similar to the blue species in previous part, the population of red starting to decreases over each time step. However, when these two species are put on the same plot, or in the same environment, they coexist and overlap in some areas. This can be seen in Figure 10 and Figure 11 where it illustrates the average population density of red and blue species, respectively for 50x50 matrix and over 100 time steps. After twenty simulations, these two average plots are nearly identical which means that they coexist and does not affect each other.

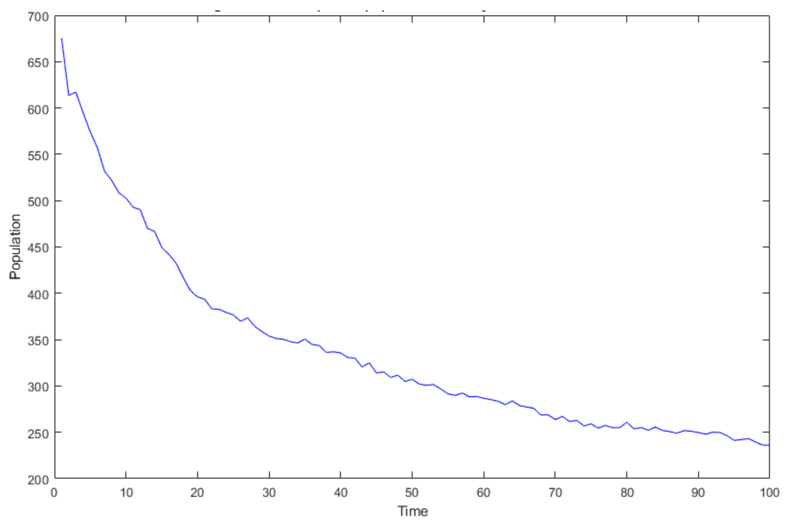

Figure 10: Blue species population density mean over 20 simulations

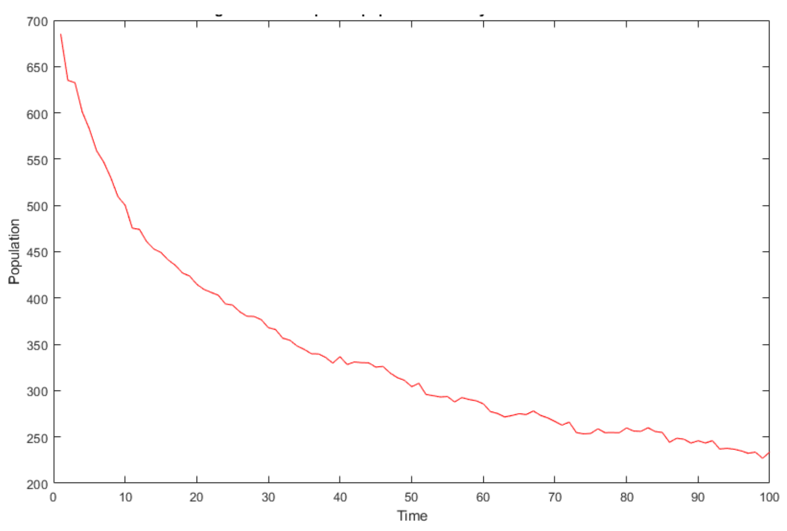

Figure 11: Red species population density mean over 20 simulations

However, when the red species was changed so that it becomes more aggressive as described in Appendix A.8, their behaviours witnessed a change. Their interactions changed so that whenever they encounters, or on the same area, the red species will eat blue. In a 50x50 matrix and over 100 time steps, their average population density change over time for twenty simulations can be seen as in Figure 13 and Figure 12. Compared to the previous two figures, it can be seen that blue species population plummeted drastically in the first ten time steps and decreases gradually as its asymptote is y=0.

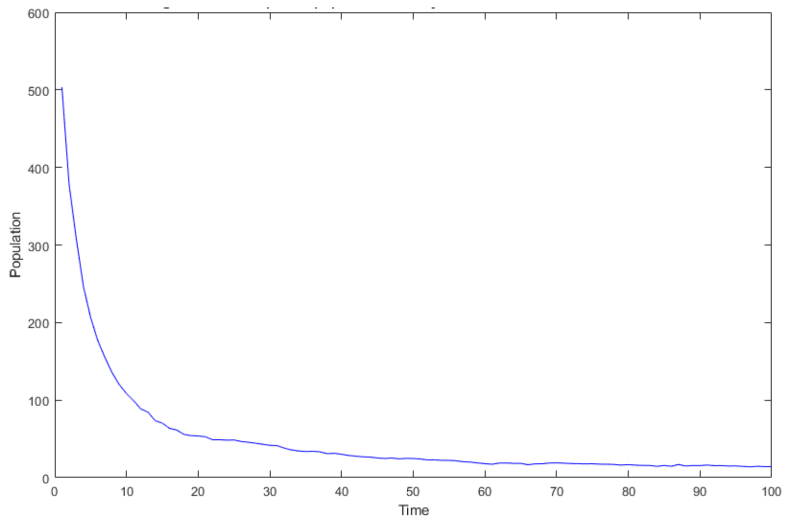

Figure 12: Blue species population density mean over 20 simulations when red eats blue

On the other hand, after eating blue whenever they encounter, the average population density graph of the red species fluctuates slightly and decreases gradually as in previous figure where this two species coexists. Therefore, as an aggressive type, red's population over 100 time steps remained unchanged from when two species are the same type.

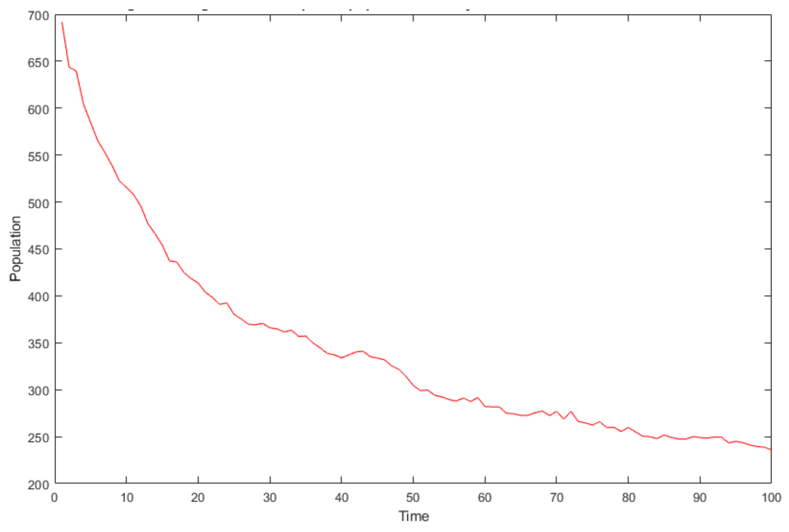

Figure 13: Red species population density mean over 20 simulations when red eats blue

Moreover, by altering the red species aggressiveness while keeping all the implementations the same, the changes in their population can be examined. When changing the aggressiveness, the n by m matrix for simulations was set up as 50x50 over 100 time steps. Firstly, the dominant species was set up with the code in Appendix **??** by replacing rand () < 0.7 to rand () < 0.1, so that whenever they encounter, there would only be 10 per cent that the inferior species get eaten. By determining that red is the dominant, the blue species is the one that die after the interaction. Figure 14 describes the average population density of blue species over twenty simulations. Even though this plot and the population's of this species in Figure 10 looks relatively similar, the number of living cells after 100 time steps has a huge difference. With both species are not aggressive, its average surviving cells are recorded as approximately 250. While with red as a dominant species that eats blue 10 per cent of all encounters, the average population of blue at the end of simulations is 120.

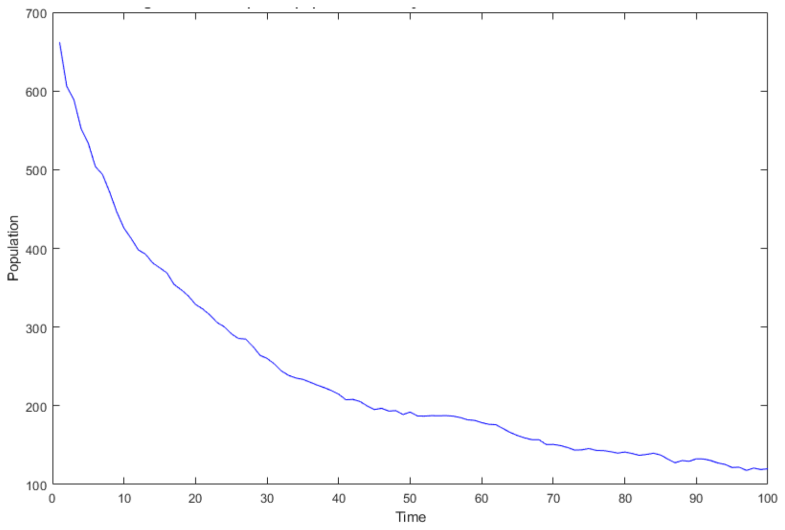

Figure 14: Blue species population density mean when red's aggression set to 10 perent

As an aggressive type, the plot of the red species remain unchanged as can be seen Figure 15. The overall trend and plot are the same as when two species coexisting on a same area. Also, the average population at the beginning and the end of twenty simulations are relatively similar at nearly 700 and 250, respectively.

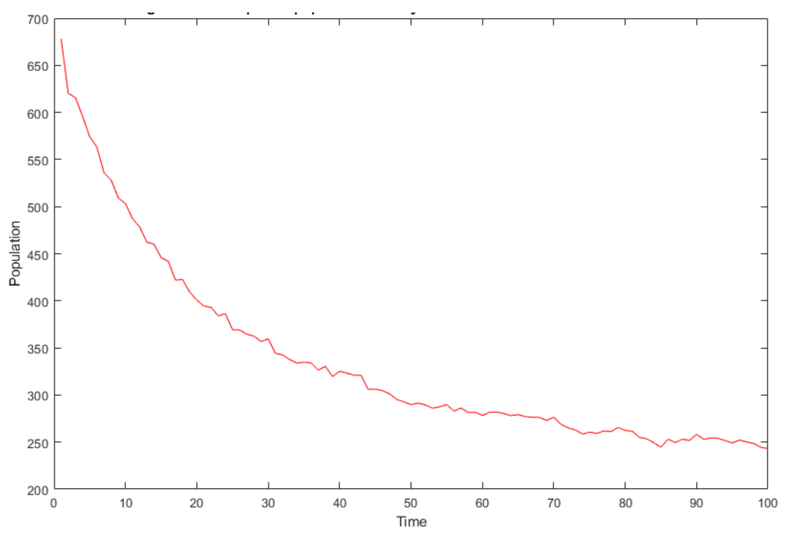

Figure 15: Red species population density mean when red's aggression set to 10 percent

Secondly, by changing the code to the one in Appendix **??**, the aggressiveness of the dominant species was increased so that for every 100 time red and blue encounter, blue will get eaten 70 per cent of the time. By looking at Figure 16, it can be seen that the red species is unaffected by the interaction with the plot and the average population density unchanged.

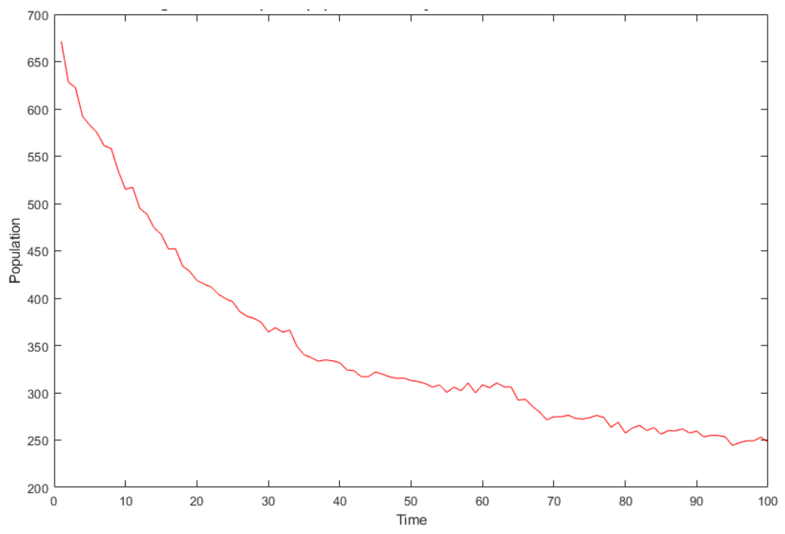

Figure 16: Red species population density mean when red's aggression set to 70 percent

However, as the aggressiveness increases, the chance of the inferior species getting killed also increases which can be witnessed in Figure 17. Similarly to Figure 13 where the blue species get eaten by the dominant every time they encounter, the plot is exponentially decreases and approaches to 0 as the time steps increase. By being eaten, its population decreases drastically from approximately 550 to 100 in the first 10 time steps.

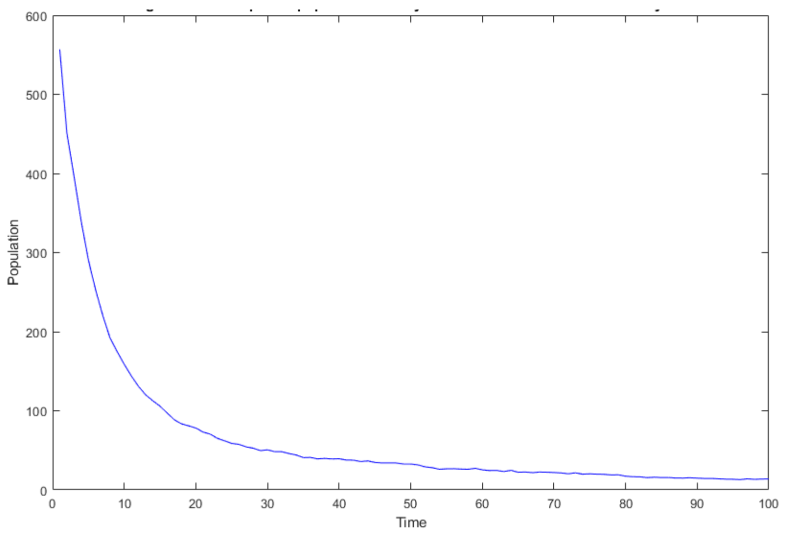

Figure 17: Blue species population density mean when red's aggression set to 70 percent

Overall, by making the red species more aggressive does not alter its overall trend of population over time. As the red species becomes more aggressive, the inferior dies out faster in the first ten time steps. However, the blue species does not die completely in 100 time step but slowly decreasing. It can be predicted that as the time steps increase, the dominated species will eventually disappeared due to the red species random interaction.

Moreover, by slightly altering the rules as seen in Appendix A.9, it is possible to observe two species original rules of growth. As the size of the matrix still set as 50x50 over 100 time steps for twenty simulations, this can be tested. When a cell of the red species is alive and their neighbours is less than the neighbours of the blue species, this cell will die. For this case, the number of living cells of the dominant is set as 5. As can be seen in Figure 18, there is no changes in its average decreasing in the number of living cells with the initial population as 680 and the final is approximately 250.

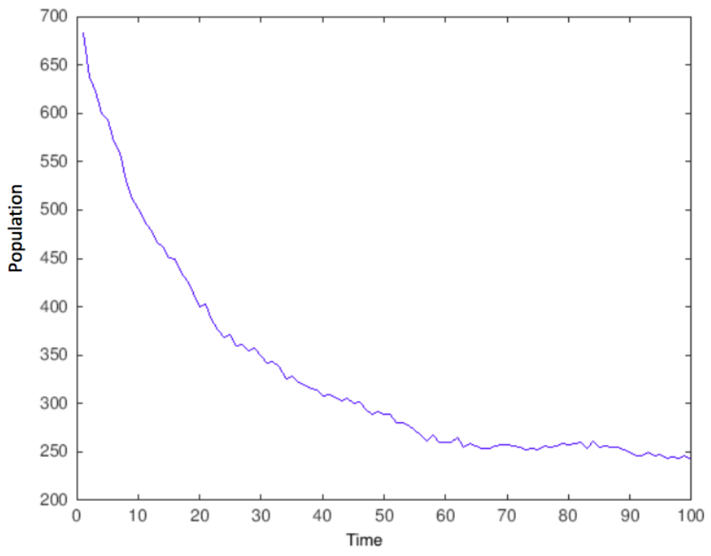

Figure 18: Blue species population density mean when changing the neighbours rules

Figure 19 illustrates the inferior species can be seen as below. Comparing to the plot of the dominant species, the differences are not extremely drastic since the final population at 100 time steps is recorded as about 230 cells. Another thing can be seen that, while for the blue species, the graph nearly flattens out from 60 time steps until the end, this plot exponentially decreases over time with no sign of stopping. However, overall, by changing the rules, there is not much of a difference between their behaviours.

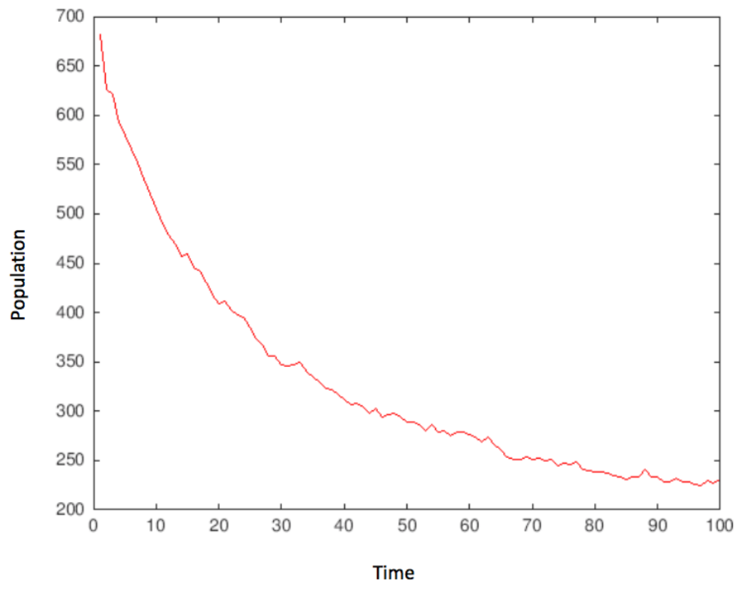

Figure 19: Red species population density mean when changing the neighbours rules

On the other hand, when we alter the rules so that the red species will die if their neighbours is simply less than the number of neighbours of the other species without specifically defining how many, the code can be seen as in Appendix A.10. Figure 20 describes the population changes over 100 time steps of the blue species in a 50x50 matrix. There is no changes in the decreasing trend of the dominant species when altering the rules as described when the plot looks exactly identical to the original graphs in Figure 4.

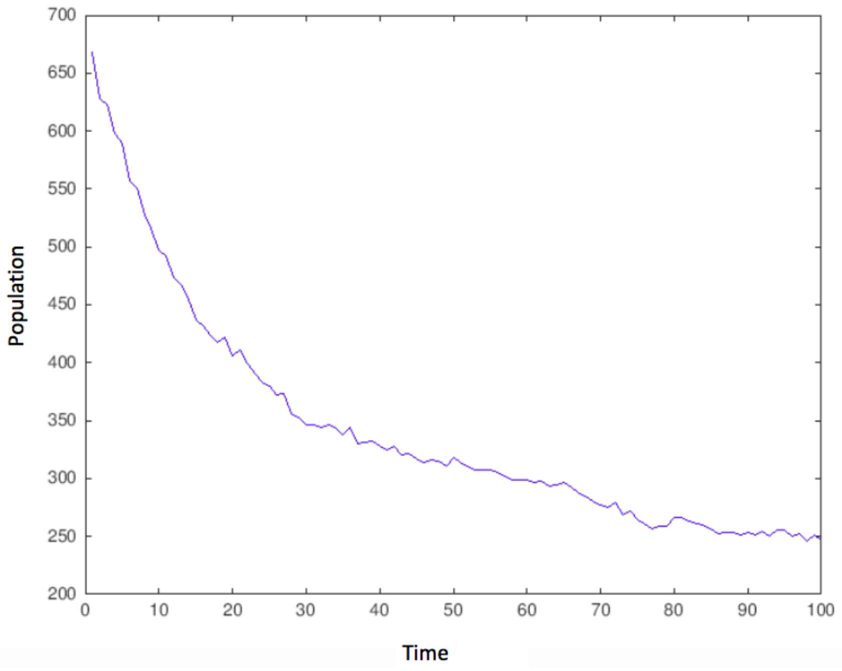

Figure 20: Blue species population density mean when alter the neighbours rules

Furthermore, by observing Figure 21 below, which it presents the population of the red species, there is no differences with the dominant species' graph. Even though there is a slightly lower in the average final population of this inferior species, it does not affect the result that these graphs looks like when this two species coexist in an environment.

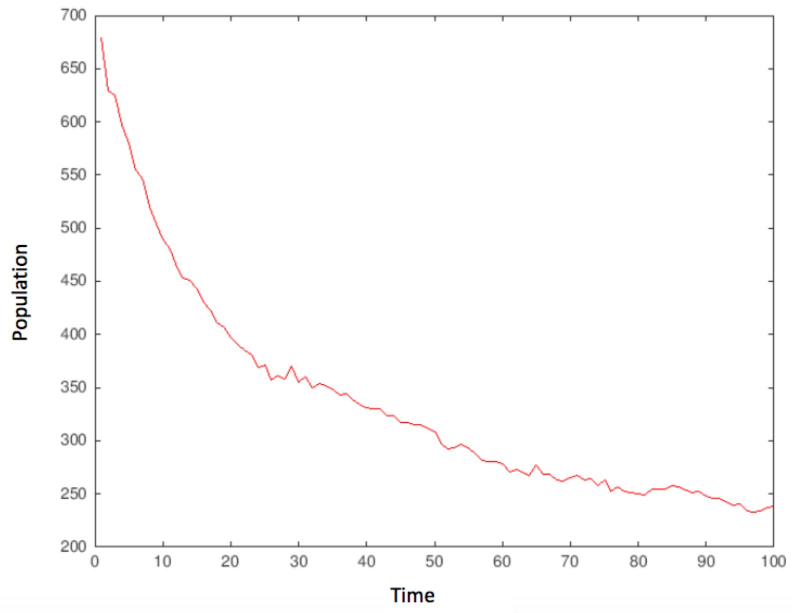

Figure 21: Red species population density mean when alter the neighbours rules

To summarize, even when the rules are changed so that a cell of the inferior species will die due to its lower in number of neighbours compared to the dominant's, these two species' plot of change in population over time are not affected. Due to this two small experiments, it can be seen that the original rules of growth is finite. As the dominant is not too aggressive, even when the red cell dies, its neighbours help them to come back to life. Also, because their interactions are random, the number of dead cells in this case is very small, which does not affect their changes in population density over time. Their populations decreases exponentially due to overpopulated and underpopulated causes by each time steps' interaction. As the number of living cells decreases, the chances for the dead cells to come back to live also decrease, hence, the cells eventually dies out.

5 Conclusion

In conclusion through studying and modifying the Conway's Game of Life code for various situations multiple population statistics were obtained along with a greater understanding of using matrices, vectors and a plethora of helpful data analysis functions in the Matlab software. Combining the results, taking the Means and standard deviation of the multiple simulations it can be seen that when parts of the code like the rules and the boundaries are changed the simulation doesn't give correct results, whereas when the matrix size and time steps are increased the results are still valid and can be extrapolated from the original plots.

Lastly the addition of another species brought interesting data, by modifying the code we could make them coexist or have one of the species aggression vary this brought about more plots and data that show the difference in population between the two species at various levels of aggression. Moreover, when we testing the original rules of growth, it was observed that the species population decreases exponentially over time. Nevertheless, the data and model aren't accurately depicting real world scenarios but with more rules and extra factors coded in, there is the potential to simulate an actual ecosystem.

References

”Conway’s Game of Life”, *Wikimedia Foundation*, Wikipedia 2019. Available: https://en.wikipedia.org/wiki/ConwaysGameofLife. [Accessed: Feb. 24, 2019]

Appendices

A Matlab Codes

A.1 Population against time for twenty simulations

For this code the matrix was kept at 50x50 with 100 time steps and over 20 simulations

```
function Game(n,m,t,nSims)

pop = zeros(nSims,t);
for k=1:nSims
    B = round(rand(n,m));
    for time=1:t
        A = B;
        for i=1:n
            for j=1:m
                cola = [i-1,i,i+1];
                cola(cola<1) = n;
                cola(cola>n) = 1;
                colb = [j-1,j,j+1];
                colb(colb<1) = m;
                colb(colb>m) = 1;
                R=sum(sum(A(cola,colb)));

                B(i,j) = ((R==3 || R==4) && A(i,j)) || (R==3
&& ~B(i,j));

            end
        end
        pop(k,time)=sum(sum(B));

   end
   Figure (2)
    plot(pop(k,:))
    hold on;

    Figure(1)
    plot(pop(1,:))

end
```

A.2 Multiple plots with mean and standard deviation using the Matlab functions mean() and std().

```
function Game(n,m,t,nSims)

pop = zeros(nSims,t);
for k=1:nSims
    B = round(rand(n,m));
    for time=1:t
        A = B;
        for i=1:n
            for j=1:m
                cola = [i-1,i,i+1];
                cola(cola<1) = n;
                cola(cola>n) = 1;
                colb = [j-1,j,j+1];
                colb(colb<1) = m;
                colb(colb>m) = 1;
                R=sum(sum(A(cola,colb)));

                 B(i,j) = ((R==3 || R==4) && A(i,j)) || ( R==3 &&
~B(i,j));

            end
        end
        spy(B);
        pause(0.05);
        pop(k,time)=sum(sum(B));

       meanvar= mean(pop);
       meanstd=std(pop);
    end
    figure(2)
    plot(pop(k,:))
    hold on;

    figure(1)
    plot(pop(1,:))
    figure(3)
    plot(pop(k,:));
    hold on;

end
figure(3)
```

```
    plot(1:t,meanvar,'-.*g');
    hold on;
figure(4)
plot(1:t,meanvar,'-.*g');
hold on ;
figure(4)
plot(1:t,(meanvar+meanstd),'-.*c');
hold on;
figure(4)
plot(1:t,(meanvar-meanstd),'-.*c');
hold on;
```

A.3 Population against time for periodic boundaries removed

```
function Game(n,m,t,nSims)

pop = zeros(nSims,t);
for k=1:nSims
    B = round(rand(n,m));
    for time=1:t
        A = B;
        for i=1:n
            for j=1:m
                if j==1
                    colb=[j,j+1];
                else if j==m
                        colb=[j-1,j];
                    else
                        colb = [j-1,j,j+1];
                    end
                    if i==1
                        cola=[i,i+1];
                    else if i==n
                            cola=[i-1,i];
                        else
                            cola = [i-1,i,i+1];
                        end

                        R=sum(sum(A(cola,colb)));
                        B(i,j) = ((R==3 || R==4) && A(i,j)) || (R==3 &&
~B(i,j));
                    end
                end
                figure (1)
                pop(k,time)=sum(sum(B));

                meanvar= mean(pop);
                meanstd=std(pop);
            end
            figure(2)
            plot(pop(k,:))
            hold on;

            figure(1)
            plot(pop(1,:))
            figure(3)
            plot(pop(k,:));
            hold on;

        end
        figure(3)
        plot(1:t,meanvar,'-.*g');
        hold on;
        figure(4)
        plot(1:t,meanvar,'b');
        hold on ;
        figure(4)
        plot(1:t,(meanvar+meanstd),'r');
        hold on;
        figure(4)
        plot(1:t,(meanvar-meanstd),'r');
        hold on;
```

A.4 Population against time for setting the rules lower

```
function Game(n,m,t,nSims)

pop = zeros(nSims,t);
for k=1:nSims
    B = round(rand(n,m));
    for time=1:t
        A = B;
        for i=1:n
            for j=1:m
                        cola = [i-1,i,i+1];
                        cola(cola<1) = n;
                        cola(cola>n) = 1;
                        colb = [j-1,j,j+1];
```

```
                    colb(colb<1) = m;
                    colb(colb>m) = 1;
                    R=sum(sum(A(cola,colb)));

                    B(i,j) = ((R==2 || R==3) && A(i,j)) || (R==3 &&
~B(i,j));

            end
            spy(B);
            pause(0.05);
             figure (1)
             pop(k,time)=sum(sum(B));

             meanvar= mean(pop);
             meanstd=std(pop);
        end
        figure(2)
        plot(pop(k,:))
        hold on;

        figure(1)
        plot(pop(1,:))
        figure(3)
        plot(pop(k,:));
        hold on;

    end
    figure(3)
    plot(1:t,meanvar,'-.*g');
    hold on;
    figure(4)
    plot(1:t,meanvar,'b');
    hold on ;
    figure(4)
    plot(1:t,(meanvar+meanstd),'r');
    hold on;
    figure(4)
    plot(1:t,(meanvar-meanstd),'r');
    hold on;
```

A.5 Population against time for setting the rules higher

```
function Game(n,m,t,nSims)

pop = zeros(nSims,t);
```

```
for k=1:nSims
    B = round(rand(n,m));
    for time=1:t
        A = B;
        for i=1:n
            for j=1:m
                 cola = [i-1,i,i+1];
                 cola(cola<1) = n;
                 cola(cola>n) = 1;
                 colb = [j-1,j,j+1];
                 colb(colb<1) = m;
                 colb(colb>m) = 1;
                 R=sum(sum(A(cola,colb)));

                 B(i,j) = ((R==4 || R==5) && A(i,j)) || (R==5
&& ~B(i,j));

                    end
                end
               spy(B);
               pause(0.05);
                figure (1)
                pop(k,time)=sum(sum(B));

                meanvar= mean(pop);
                meanstd=std(pop);
            end
            figure(2)
            plot(pop(k,:))
            hold on;

            figure(1)
            plot(pop(1,:))
            figure(3)
            plot(pop(k,:));
            hold on;

        end
        figure(3)
        plot(1:t,meanvar,'-.*g');
        hold on;
        figure(4)
        plot(1:t,meanvar,'b');
        hold on ;
        figure(4)
        plot(1:t,(meanvar+meanstd),'r');
        hold on;
        figure(4)
        plot(1:t,(meanvar-meanstd),'r');
        hold on;
```

A.6 Sources and Sinks

```
function Game(n,m,t,nSims)

pop = zeros(nSims,t);
for k=1:nSims
    B = round(rand(n,m));
    for time=1:t
        A = B;
        for i=1:n
            for j=1:m
                cola = [i-1,i,i+1];
                cola(cola<1) = n;
                cola(cola>n) = 1;
                colb = [j-1,j,j+1];
                colb(colb<1) = m;
                colb(colb>m) = 1;
                R=sum(sum(A(cola,colb)));

                B(i,j) = ((R==3 || R==4) && A(i,j)) || (R==3
&& ~B(i,j));

                 if (i == 5 && j == 5)
                      B(5,5) = 1;
                  end
                  if (i == 45 && j == 45)
                          B(45,45) = 0;
                   end
              end
             spy(B);
             pause(0.05);
         end
     end
```

A.7 Population against time for two coexisting species

```
function gameoflife(n,m,t,nSims)

for k=1:nSims
```

```
    B = round(rand(n,m));
    D = round(rand(n,m));

    for time=1:t
        A = B;
        C=D;
        for i=1:n
            for j=1:m
                cola = [i-1,i,i+1];
                cola(cola<1) = n;
                cola(cola>n) = 1;
                colb = [j-1,j,j+1];
                colb(colb<1) = m;
                colb(colb>m) = 1;
               R=sum(sum(A(cola,colb)));

               B(i,j) = ((R==3 || R==4) && A(i,j)) || ( R==3 && ~B(i,j));

               S=sum(sum(C(cola,colb)));

               D(i,j) = ((S==3 || S==4) && C(i,j)) || ( S==3 && ~D(i,j));

            end
        end
         subplot(1,3,1)
         spy(D,'r');
         subplot(1,3,2)
         spy(B);
         subplot(1,3,3)
         hold off
         spy(D,'r'); hold on
         spy(B,'b');
         spy(B&D,'y')
         pause(0.05);

        pop(k,time)=sum(sum(B)); meanpop= mean(pop);
        popu(k,time)=sum(sum(D)); meanpopu= mean(popu);
        figure (1)
        plot(1:t,meanpop,'r');
        figure(2)
        plot(1:t,meanpopu,'b');
    end
end
```

A.8 Population against time of both species when Red eats Blue when encounter

```
function gameoflife(n,m,t,nSims)

for k=1:nSims
    B = round(rand(n,m));
    D = round(rand(n,m));

    for time=1:t
        A = B;
        C = D;
        for i=1:n
            for j=1:m
                cola = [i-1,i,i+1];
                cola(cola<1) = n;
                cola(cola>n) = 1;
                colb = [j-1,j,j+1];
                colb(colb<1) = m;
                colb(colb>m) = 1;
               R=sum(sum(A(cola,colb)));

               B(i,j) = ((R==3 || R==4) && A(i,j)) || ( R==3 && ~B(i,j));

               S=sum(sum(C(cola,colb)));

               D(i,j) = ((S==3 || S==4) && C(i,j)) || ( S==3 && ~D(i,j));

                 if (B(i,j) && D(i,j))
                     B(i,j) = 1;
                     D(i,j) = 0;
               end
           end
       end
        subplot(1,3,1)
        spy(D,'r');
        subplot(1,3,2)
        spy(B);
        subplot(1,3,3)
        hold off
        spy(D,'r'); hold on
        spy(B,'b');
        spy(B&D,'y')
        pause(0.05);
        pop(k,time)=sum(sum(B)); meanpop= mean(pop);
        popu(k,time)=sum(sum(D)); meanpopu= mean(popu);
        figure (1)
        plot(1:t,meanpop,'r');
        figure(2)
        plot(1:t,meanpopu,'b');
    end
end
```

A.9 Population against time for changing rules 1

```
function game(n,m,t,nSims)

for k=1:nSims
    B = round(rand(n,m));
    D = round(rand(n,m));

    for time=1:t
        A = B;
        C=D;
        for i=1:n
            for j=1:m
                cola = [i-1,i,i+1];
                cola(cola<1) = n;
                cola(cola>n) = 1;
                colb = [j-1,j,j+1];
                colb(colb<1) = m;
                colb(colb>m) = 1;
                R=sum(sum(A(cola,colb)));

                B(i,j) = ((R==3 || R==4) && A(i,j)) || ( R==3 && ~B(i,j));
                S=sum(sum(C(cola,colb)));

                D(i,j) = ((S==3 || S==4) && C(i,j)) || ( S==3 && ~D(i,j));

                if (A(i,j)==1) && (C(i,j)==1)
                        if (S==5) && (R <= S)
                        (B(i,j)==0) && (D(i,j) == 1);
                        end
                end

            end
        end
```

```
          pop(k,time)=sum(sum(B)); meanpop= mean(pop);
          popu(k,time)=sum(sum(D)); meanpopu= mean(popu);
          figure (1)
          plot(1:t,meanpop,'r');

          figure(2)
          plot(1:t,meanpopu,'b');
     end

end
```

A.10 Population against time when changing rules 2

```
function game(n,m,t,nSims)
for k=1:nSims
     B = round(rand(n,m));
     D = round(rand(n,m));

     for time=1:t
          A = B;
          C=D;
          for i=1:n
               for j=1:m
                    cola = [i-1,i,i+1];
                    cola(cola<1) = n;
                    cola(cola>n) = 1;
                    colb = [j-1,j,j+1];
                    colb(colb<1) = m;
                    colb(colb>m) = 1;
                    R=sum(sum(A(cola,colb)));

                    B(i,j) = ((R==3 || R==4) && A(i,j)) || ( R==3 && ~B(i,j));
                    S=sum(sum(C(cola,colb)));

                    D(i,j) = ((S==3 || S==4) && C(i,j)) || ( S==3 && ~D(i,j));

                    if (A(i,j)==1) && (C(i,j)==1)
                              if (R<=S)
                              (B(i,j)==0) && (D(i,j) == 1);
                              end
                    end
               end
          end

          pop(k,time)=sum(sum(B)); meanpop= mean(pop);
```

```
        popu(k,time)=sum(sum(D)); meanpopu= mean(popu);
        figure (1)
        plot(1:t,meanpop,'r');

        figure(2)
        plot(1:t,meanpopu,'b');
    end


end
```

B Plots

Below are some of the plots referenced in previous discussions.

B.1 Time alteration plots

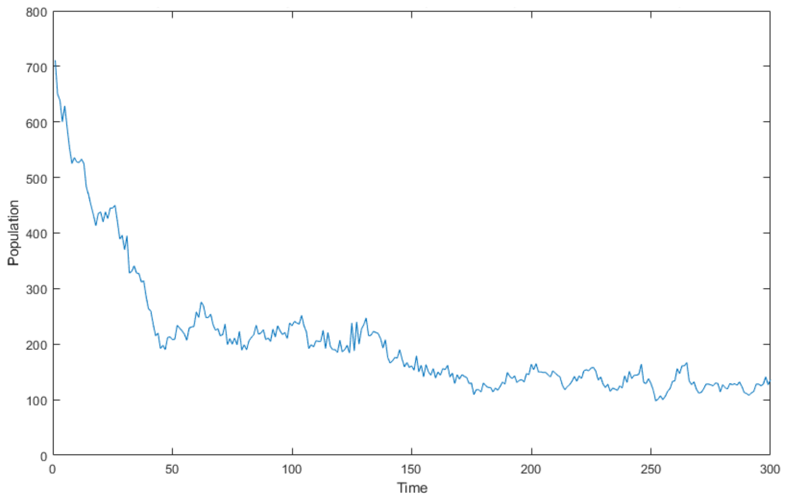

Figure 22: Single simulation plot for the time increased to 300 steps

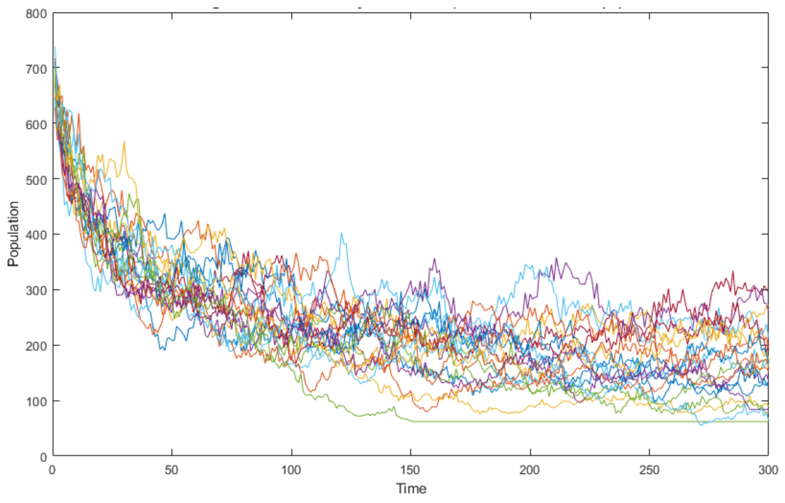

Figure 23: 20 simulation plot for the time increased to 300 steps

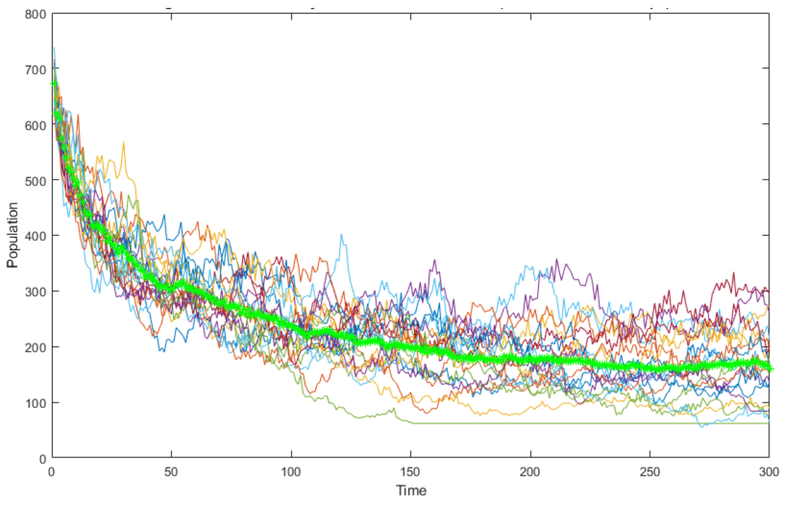

Figure 24: 20 simulation with mean plot for the time increased to 300 steps

B.2 Matrix alteration plots

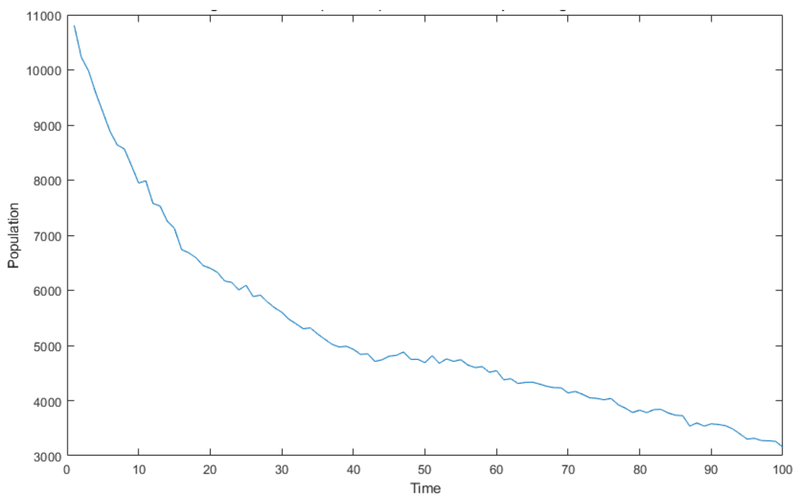

Figure 25: Single simulation with matrix increased to 200x200

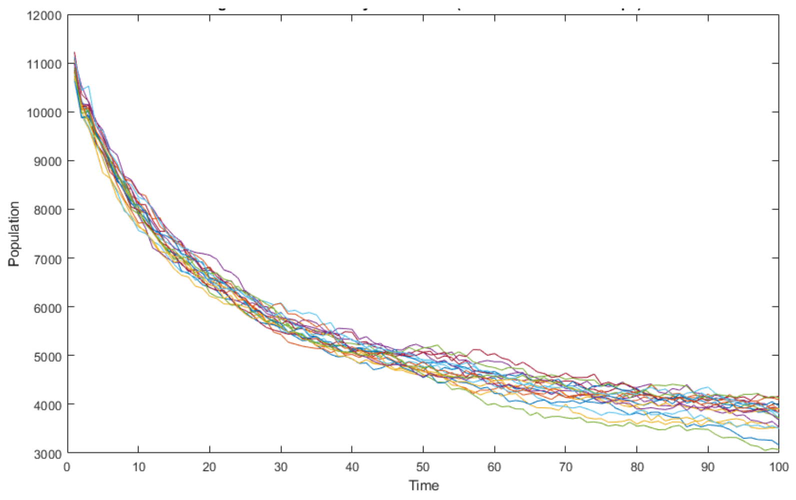

Figure 26: 20 simulation plot with matrix set to 200x200

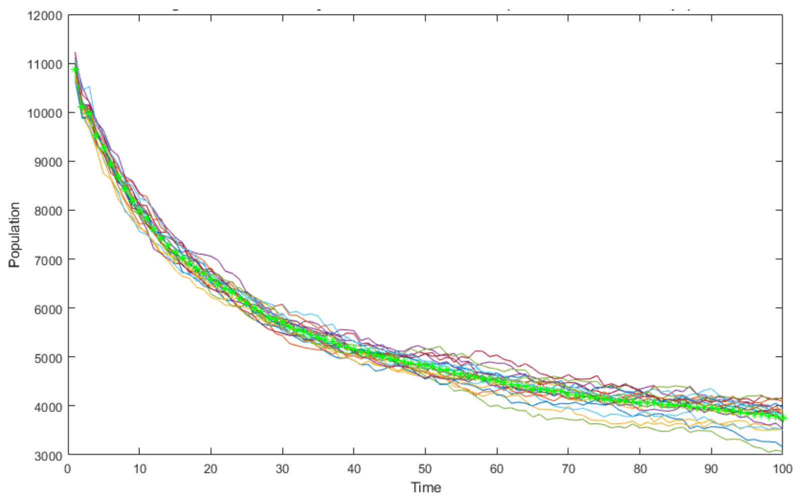

Figure 27: 20 simulations and the mean with matrix set to 200x200

B.3 Periodic boundaries removed

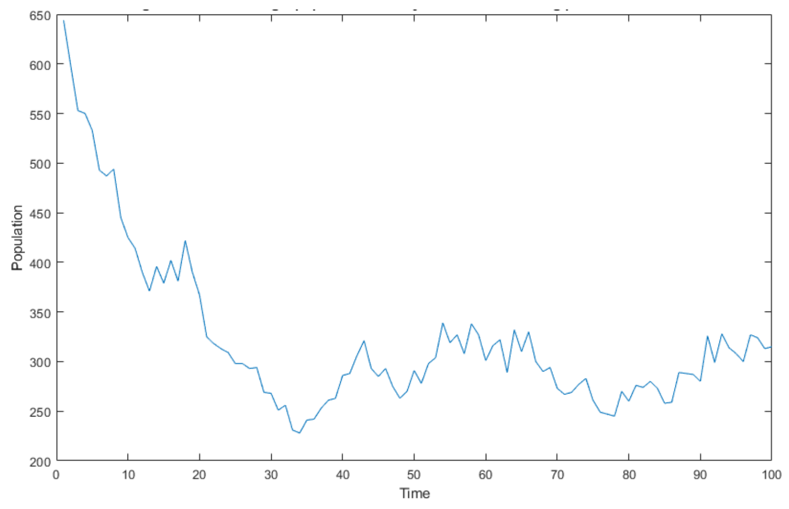

Figure 28: Single simulation plot with the boundaries changed

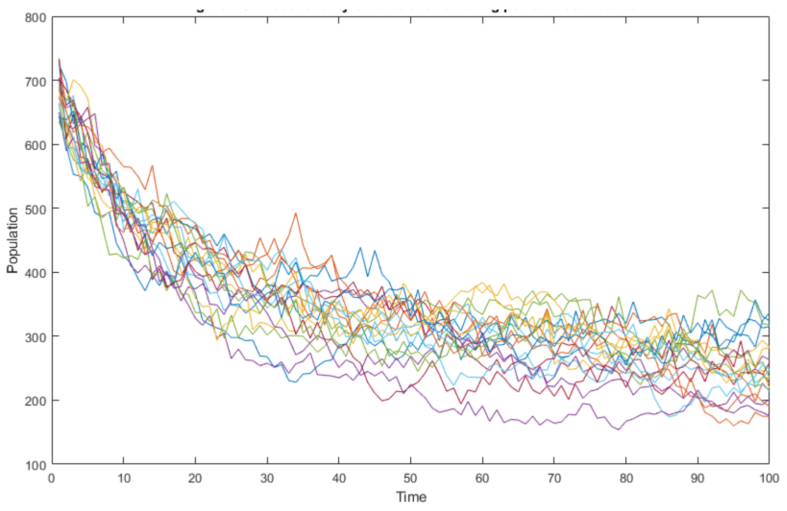

Figure 29: 20 simulation plot with the boundaries removed