Jared Rivera

804603106

CEE/MAE M20

February 28, 2017

Homework 7

1 The Game of Life

1.1 Introduction

The goal of this problem is to follow John Conway's rules for the game of life regarding cells in a Petri dish, but with automation and a periodic boundary condition. The cell population will be generated at random then run through a series of "generations", where their population dynamic is governed by the following rules: 1) A living cell with 2-3 living neighbors survives. 2) A living cell with less than 2 or more than 3 neighbors dies. 3) A dead cell with exactly 3 neighbors comes to life. The population will be initialized at a random spatial distribution of 10% living and 90% dead cells. The periodic boundary condition will allow for virtually infinite linear space in a finite geometry resembling a torus.

1.2 Model and Methods

The first step in this problem is to initialize the matrix that holds the cell statuses. This is done by making a matrix of random numbers, and using an if statement to say that if the random number is 0.1 or below the cell will be alive, and dead otherwise. A matrix of the same size is also preallocated with zeros to be overwritten later of when the cell population changes over generation. Next, the number of generations is specified in order to create a parameter for the outer for loop to run population change iterations.

Inside the generation loop, the neighboring cells have to be analyzed in order to determine the changes in population. This is done by iterating through each entry and assigning it a neighboring value. For central cells this is done intuitively, but for the cells on the boundary they are assigned according to the fact that there's a boundary condition, so the plane wraps around as shown below.

```
North=row-1;
 if North==0
  North=Ny;
     end
```

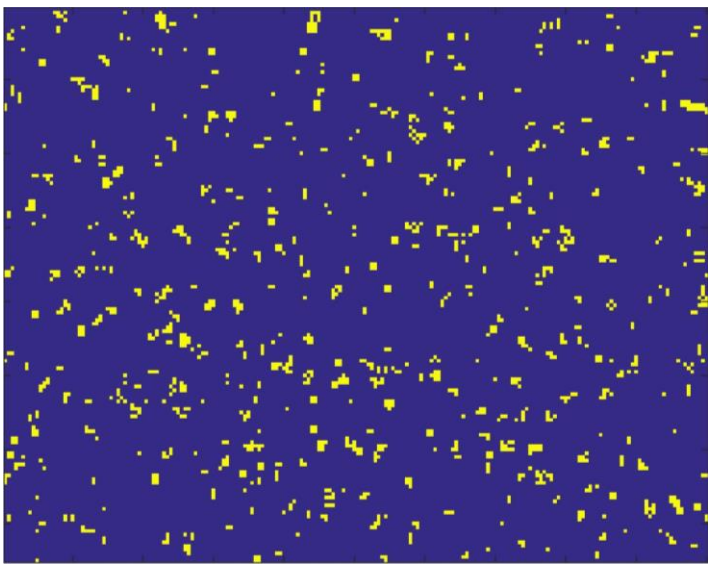
Once this is done for each cardinal direction, the sum of neighboring cell values is calculated to determine the number of living neighbors, and the values of the new matrix are entered based on the criteria set in the rules of the game, with the example of a dead cell coming to life shown below.

```
    %For dead cell
 if A(row,column)==0
   if neighbors==3
A_new(row,column)=1;
         end
         end
```

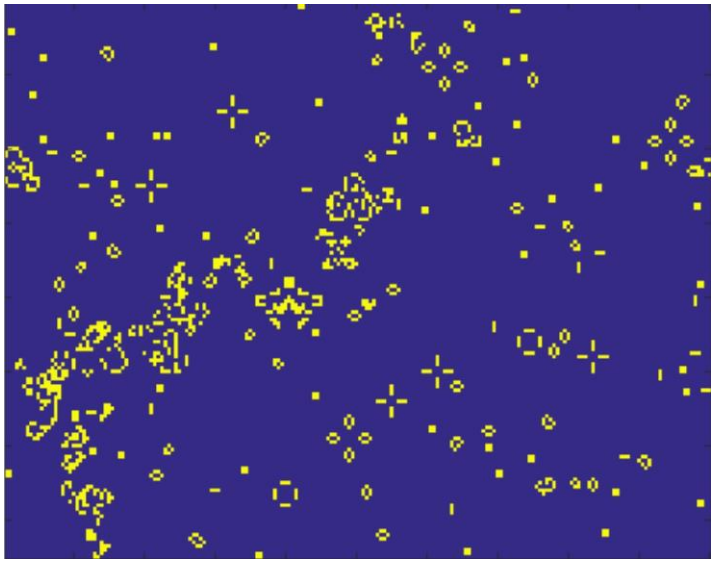
Finally, the value of the old matrix is reassigned to the new matrix and the animation is generated using the imagesc function and the drawnow command.

1.3 Calculations and Results

When the program is run, an animation comes on screen with the population per generation. Below are an example of the initial and final cell distributions, with yellow being living cells and blue being dead cell

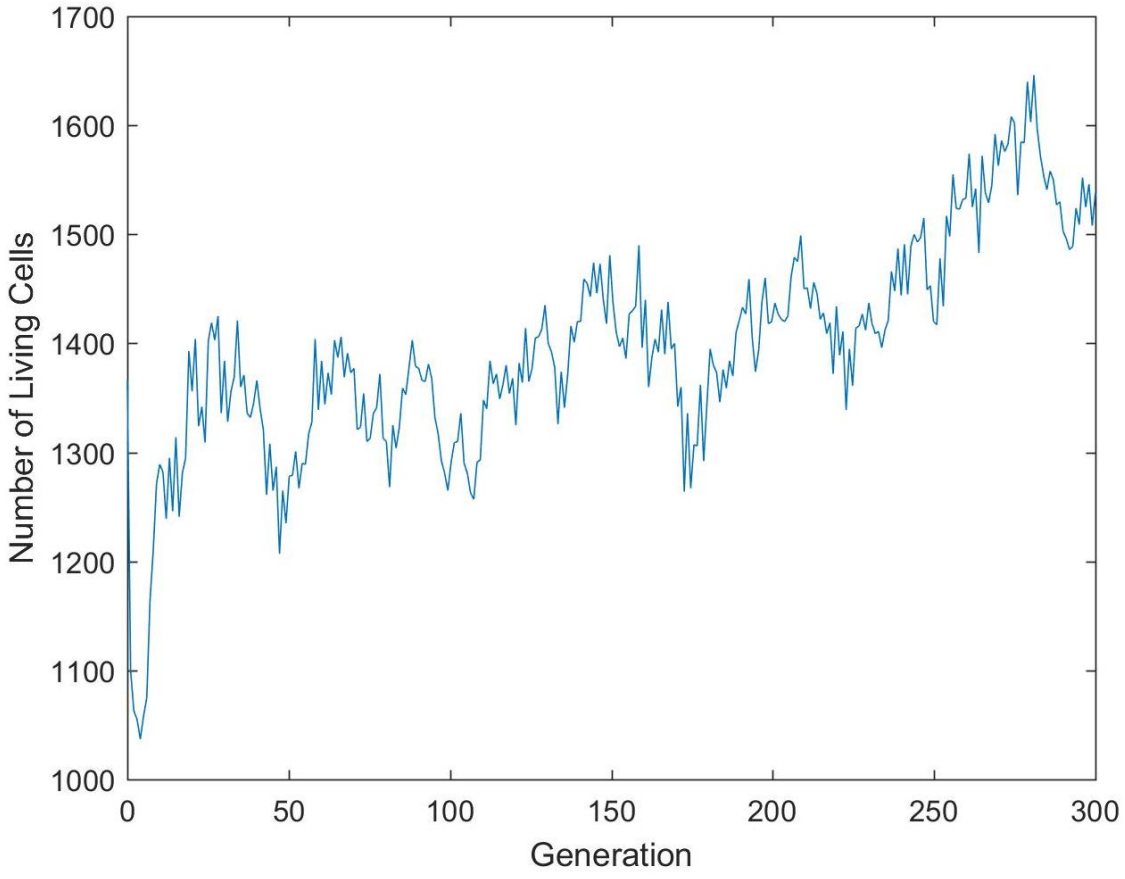


Initial Distribution



Final Distribution

Afterwards, the cell population over time is plotted as follows:



1.4 Discussion

Over the course of many generations, a few trends seem to arise. First is the initial competition in the population. Due to the even distribution, the cells begin to "fight" for space and massive circles grow and break apart as the cells look for space. This can be seen on the left side of the picture included above, where there's a trail of circular groups. Next is the groups in equilibrium. Common shapes that arise are a plus sign, a circle, and a line, and they oscillate and spin but don't translate anywhere and appear to keep a constant number of living cells. Finally, there are floaters. These are small Tetris-piece like shapes that move in a straight line across the plane.

Over the course of the generations, the population rises and falls dramatically but by inspection of the graph one can see an underlying increase in the total number of living cells over time. Of course, all these observations lie in the rules set forth in the problem statement. If, for example, the population is started at a much higher number (i.e. 40% living) there will be an exponential decrease over time, and a more stable equilibrium is found with new shapes like closed boxes. If the population is started much lower (i.e. 1%) there won't be enough neighbors to support life, and all cells will die. If the population is initialized at 10% and the number of neighbors allowed is raised to 7, the total number of cells rises drastically and reaches an equilibrium quickly.

One interesting possibility would be to place two populations in the plane, one that's a predator and one that's a prey. This would allow for a visualization of something similar to the Lotka-Volterra equations but in space rather than as just a graph. My attempt at this was to assign a certain number of cells the value 50, and have those be aggressive. They quickly dominated the neighboring cells but soon after died off due to their inability to live near each other, all during the first generation. I'd have to put in more stringent parameters to allow for a more realistic dynamic in the future.

2 Euler-Bernoulli Beam Bending

2.1 Introduction

The goal of this problem is to calculate the deflection in a simply supported beam subjected to a point load by using the second order relationship between the load and the deflection. A second order Euler method will be derived and used from the methods used previously, and the beam will be broken up by method of sections and the deflection in each section will be calculated. The calculated value will then be compared to the theoretical value found from the equation given in the problem statement.

2.2 Model and Methods

The first step to this problem is to initialize (and calculate in the case of moment of inertia) the known variables, these being the geometric and physical properties of the beam and its loading. Next set the parameters surrounding the division of the beam into sections, these being the number of sections, the array holding the x-values at each division, and the spacing between each division.

Now the system of equations for the Euler method must be set up. It will be solved using matrix algebra, so the matrix is preallocated with as many rows and columns as there are subdivisions of the beam. Knowing the form if the Euler method to be as follows,

$$\frac{f(x+dx)-2f(x)+f(x-dx)}{dx^2} \text{ with error } O(\Delta x^2)$$

the matrix is then put through a loop to assign it values of 0, 1, or -2 based on row, as shown below

```
for k=1:N-2
A(k+1,k)=1;
A(k+1, k+1)=-2;
A(k+1, k+2)=1;
end
```

Next, the vector containing the deflection second derivative values is initialized, with the boundary conditions known and set to zero, and the rest all filled by the following iteration. By determining the x-position of the segment, the proper moment value is calculated using if statements, then all the needed information is plugged into the second order Euler equation seen below.

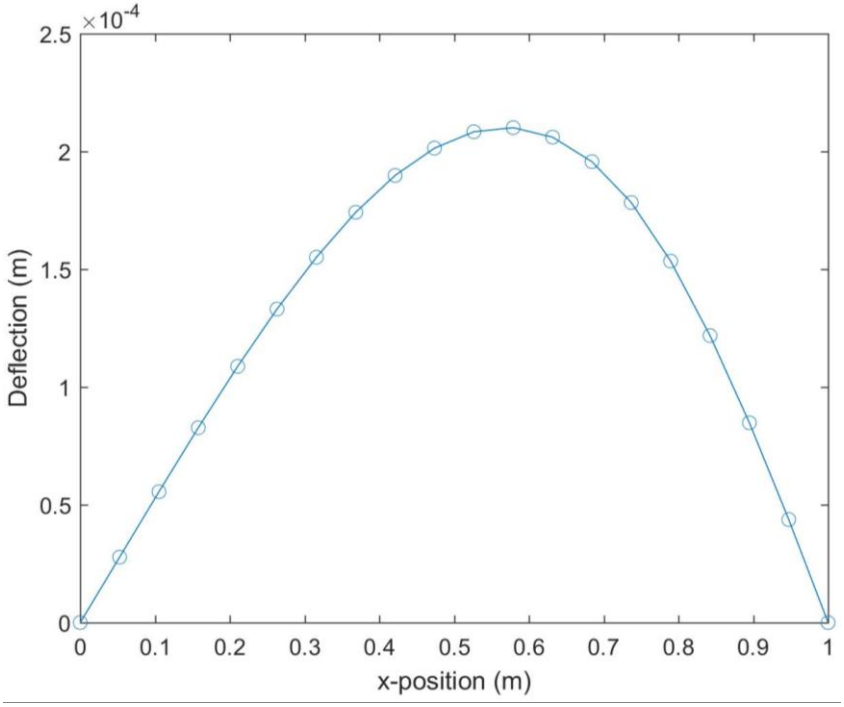
```
b(j)=((dx^2)*M)/(E*I);
```

Where j is the current iteration. Next, using the inverse command (`y=A\b;`), the solution to the system of equations is found, and that solution contains the y-positions of the beam for every section. The maximum value is then calculated and compared to the maximum value found through the theoretical equation from the problem statement and the error between the two values is calculated. Finally, deflection is plotted as a function of x and the error between the calculated and theoretical values is printed to the command window.

2.3 Calculations and Results

When the program is run, and the number of divisions is set to 20, the following is printed to the command window, and the plot below it is displayed.

```
The calculated and theoretical values differ by 4.91 percent
```



2.4 Discussion

As seen in the results section, the maximum displacement found by the Euler method is 2.1×10^{-4}m, which only differs from the theoretical value by less than 5%. That's a good estimate for only 20 subdivisions of the beam. Interestingly enough, the maximum displacement occurs closer to 0.6m in the x-direction, while the load is placed at 0.7m. This discrepancy may seem like it is likely due to the finite number of subdivisions and will improve as more are taken, but at 100 subdivisions the maximum deflection comes closer to 0.6m along the x axis, and the error in value drops below a single percent. This shows that the maximum deflection doesn't necessarily occur exactly beneath the load.

However, the maximum displacement does match the position where the load is applied if it is at the direct center of the beam. The error is actually smallest (0.04%) at that point as well, likely due to lack of roundoff error because of the clean numbers involved. After experimentation with the placement of the load, it appears that the maximum deflection occurs between x=0.4m and x=0.7m, and is actually pushed furthest to those extremities by values such as d=0.25m and d=0.8m.