Cellular Automata

Amanda Rojas

Fall 2020

Abstract

This lab introduces the concept of cellular automata (CA) and how to use it to generate complex patterns. A simple example of this is that a pattern is generated by basing one cell's outcome from the specific pattern of 3 adjacent cells. This is expanded upon with discrete rules for each step and is then iterated over and over to produce complex structures.

1 Introduction

Cellular automata (CA) are relevent for generating complex patterns from very simple rules. There are a possible 2^8(256)number of possible outcomes. This is because there are only 2 responses for 8 rules.
It begins by creating a grid with cells that are either colored or not colored. Then there are rules to determine if the following cells will be colored or not. The example that we looked at was Rule 90, which is outlined below in figure 1.

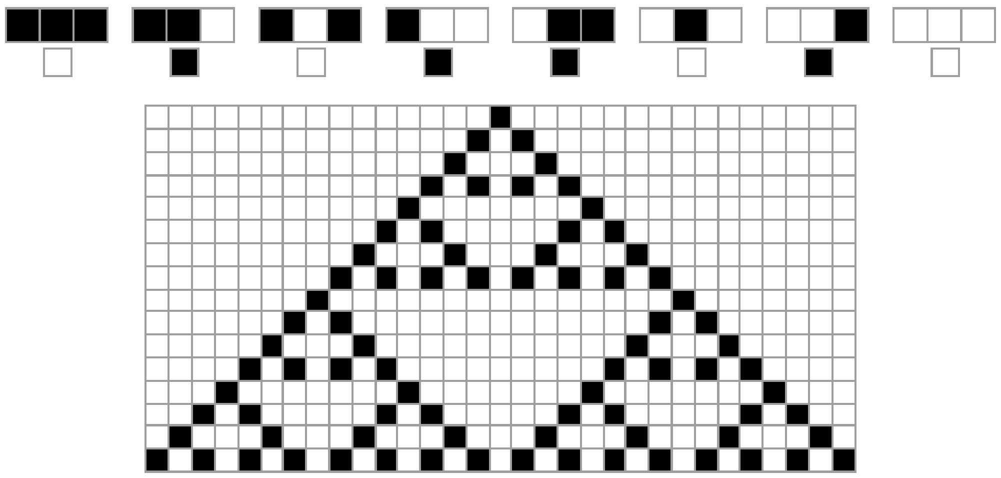


Figure 1: Rule 90, with the rules for each cell are indicated above.

2 Theory

In order to generate the pattern visualized in figure 1 the rules must be followed by the algorithm.

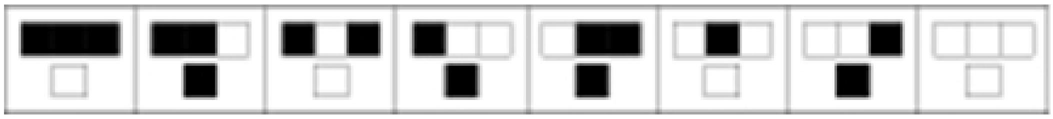


Figure 2: Rule 90

The cell in the bottom center is determined by the three above it. By manipulating what pattern generates a colored or blank cell is how you come to have the 256 possible outcomes of cellular automata. Above in figure 2 are the rules for Rule 90.

3 Procedures

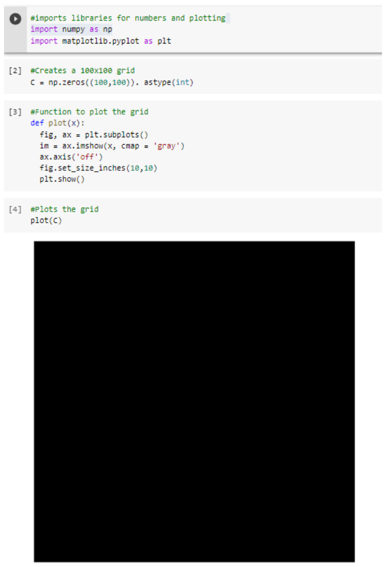
In this lab we learned 3 different ways to generate this automata.

First for all the methods you want to import your python libraries:

```
import numpy as np

x = np.random.random((2,2))
y = x + 3
```

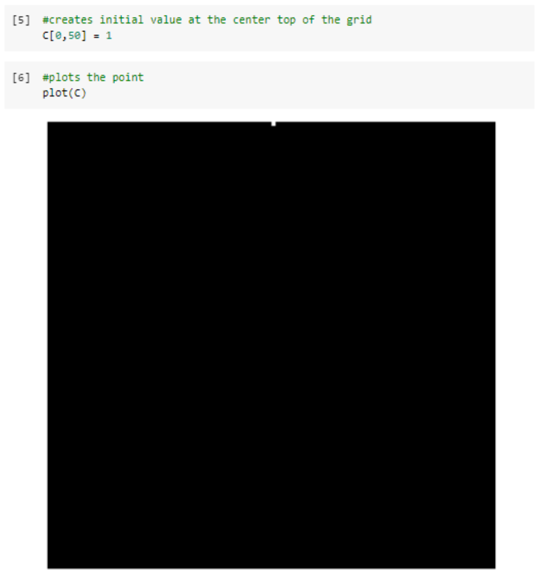
1. Excel Coded Method

   - Creating a grid



```
#imports libraries for numbers and plotting
import numpy as np
import matplotlib.pyplot as plt

[2] #Creates a 100x100 grid
    C = np.zeros((100,100)). astype(int)

[3] #Function to plot the grid
    def plot(x):
      fig, ax = plt.subplots()
      im = ax.imshow(x, cmap = 'gray')
      ax.axis('off')
      fig.set_size_inches(10,10)
      plt.show()

[4] #Plots the grid
    plot(C)
```

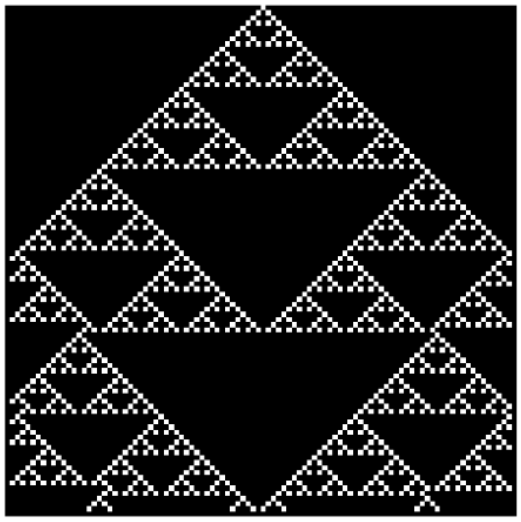
- Plotting the initial cell in the center



```
[5] #creates initial value at the center top of the grid
    C[0,50] = 1
```

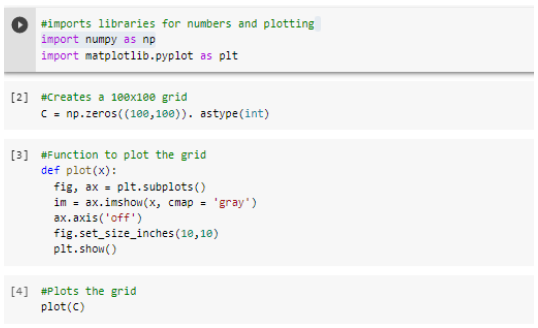
```
[6] #plots the point
    plot(C)
```

- Following the rules in Excel by coding it to do so.



2. Using a Bin in Python to Extract the Rule Set.

- Creating a grid



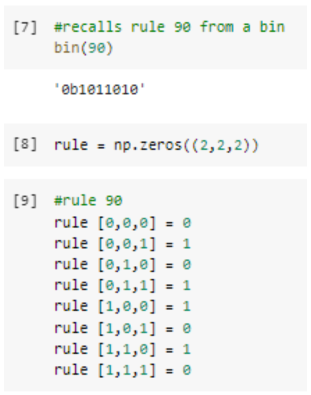
```
#imports libraries for numbers and plotting
import numpy as np
import matplotlib.pyplot as plt
```

```
[2] #Creates a 100x100 grid
    C = np.zeros((100,100)). astype(int)
```

```
[3] #Function to plot the grid
    def plot(x):
      fig, ax = plt.subplots()
      im = ax.imshow(x, cmap = 'gray')
      ax.axis('off')
      fig.set_size_inches(10,10)
      plt.show()
```

```
[4] #Plots the grid
    plot(C)
```

- Recalling Rule from a bin.

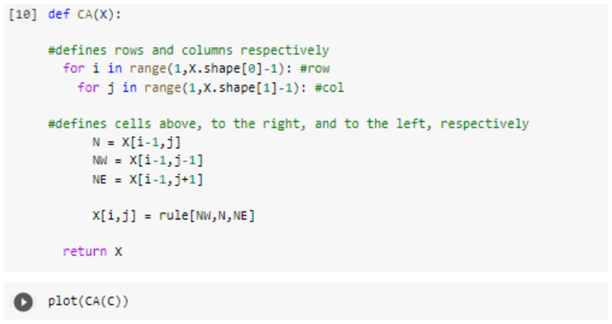


```
[7] #recalls rule 90 from a bin
    bin(90)

    '0b1011010'
```

```
[8] rule = np.zeros((2,2,2))
```

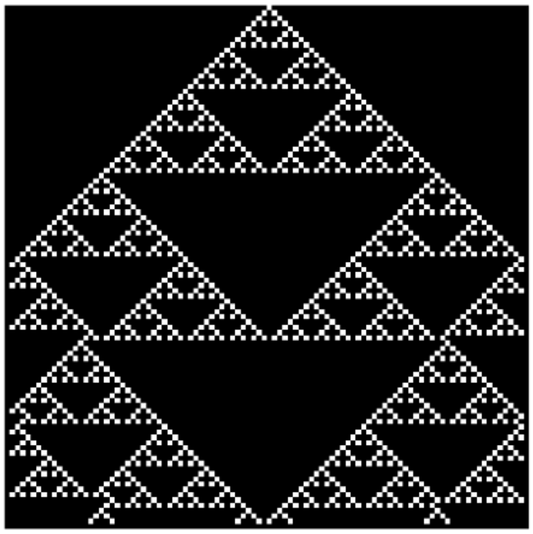
```
[9] #rule 90
    rule [0,0,0] = 0
    rule [0,0,1] = 1
    rule [0,1,0] = 0
    rule [0,1,1] = 1
    rule [1,0,0] = 1
    rule [1,0,1] = 0
    rule [1,1,0] = 1
    rule [1,1,1] = 0
```

- Execute function to define CA and follow the rules outlined.



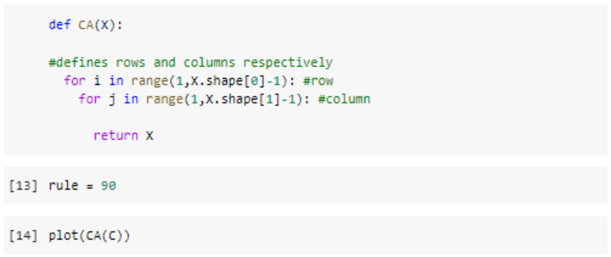
```
[10] def CA(X):

     #defines rows and columns respectively
       for i in range(1,X.shape[0]-1): #row
         for j in range(1,X.shape[1]-1): #col

     #defines cells above, to the right, and to the left, respectively
           N = X[i-1,j]
           NW = X[i-1,j-1]
           NE = X[i-1,j+1]

           X[i,j] = rule[NW,N,NE]

       return X
```

```
plot(CA(C))
```



3. Using a formula that defines a cell based on it's neighbors to arrive at the automata.

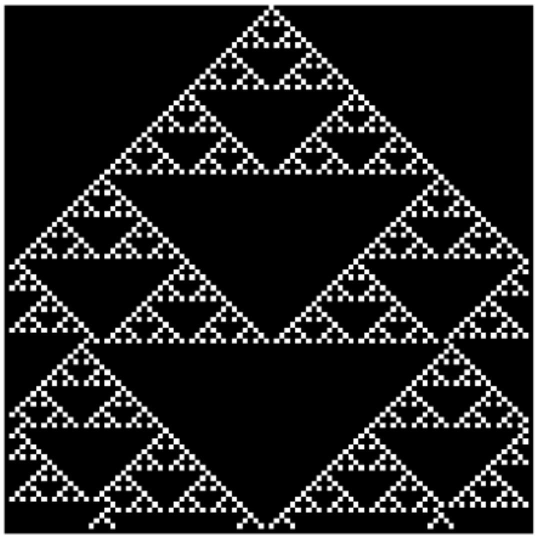
$$X[i,j] = (rule/(2**(4*X[i-1,j-1] + 2*X[i-1,j] + X[i-1,j+1]))) \qquad (1)$$



```
def CA(X):

#defines rows and columns respectively
  for i in range(1,X.shape[0]-1): #row
    for j in range(1,X.shape[1]-1): #column

      return X
```

```
[13] rule = 90
```

```
[14] plot(CA(C))
```



4 Conclusions

Cellular automata is another method that generates patterns/organization of excess data using simple tools.

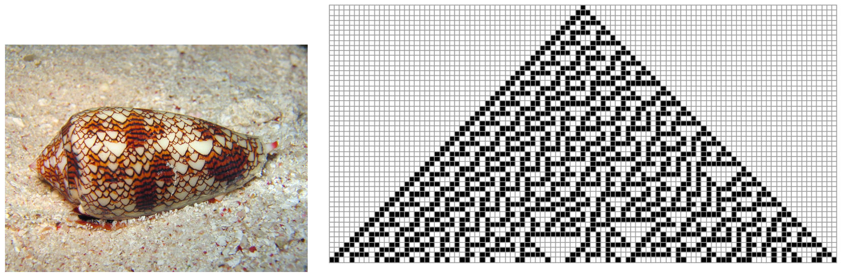


Figure 3: Rule 30 has been seen in nature, for example the pattern on this sea shell.

Similar to the chaos game, this can be appreciated in nature (see figure 3), which insinuates that data sets that are seemingly "random" have a method of organization that just can't be appreciated without further study of these nonlinear systems.