CS166 - Problem Set 1

Minerva Schools at KGI

CS166 - Problem Set 1

Problem 1: Design a Model

LOs: #Modeling

HCs: #emergentproperties

Describe a real-world system of your choice that is made up of many interacting components.

A real-world system of my choice that is made up of many interacting components would be the spread of a contagious disease wherein the cellular automata would represent the population of a given place and the cells in the grid would represent human contagion with different states for each agent based on health.

At which scale do you choose to describe the microscopic components and what are those components? Briefly describe other scales (larger and/or smaller) at which you could have chosen to model the microscopic state and why your chosen scale is the most appropriate.

For this model, I choose to describe the microscopic components at the group level because that way one can assess the behavior that the components take on in clusters or groups of cells; this would include closely related cells that interact with one another. Other scales at which I could have chosen to model the microscopic state could be a smaller more individual scale for determining whether one single agent would be infected, or a larger scale for determining the total spread in a large but finite population. The chosen scale is the most appropriate because given the real-world implications, such model and considerations could provide insights to implementing policy and solutions to the problems and avoid further spread.

What states can the microscopic components take on?

The states that the microscopic components can take on include infected, not-infected, previously infected, healthy, and more depending on how rigorous the model and simulation can achieve to represent real-life scenario.

What are the relationships between the components?

The relationships between the components here is that they can change the state of each other based on rules of transmission or based on health status.

How do the states of the components change over time? Some of these changes may depend on interactions between the components while other changes may not.

The states of the components change over time through a few different rules such as transmission rates and interactions between nearby cells. The rules should model real-life as much as possible and other changes such as self-recovery or self-immunity.

Predict what kind of macroscopic behavior would arise if you ran a computational simulation of your model.

The kind of macroscopic behavior that would arise if a computational simulation of the model was ran would be a steady state after many time steps of either a certain level of infection with the population or healthy levels depending on the model assumptions.

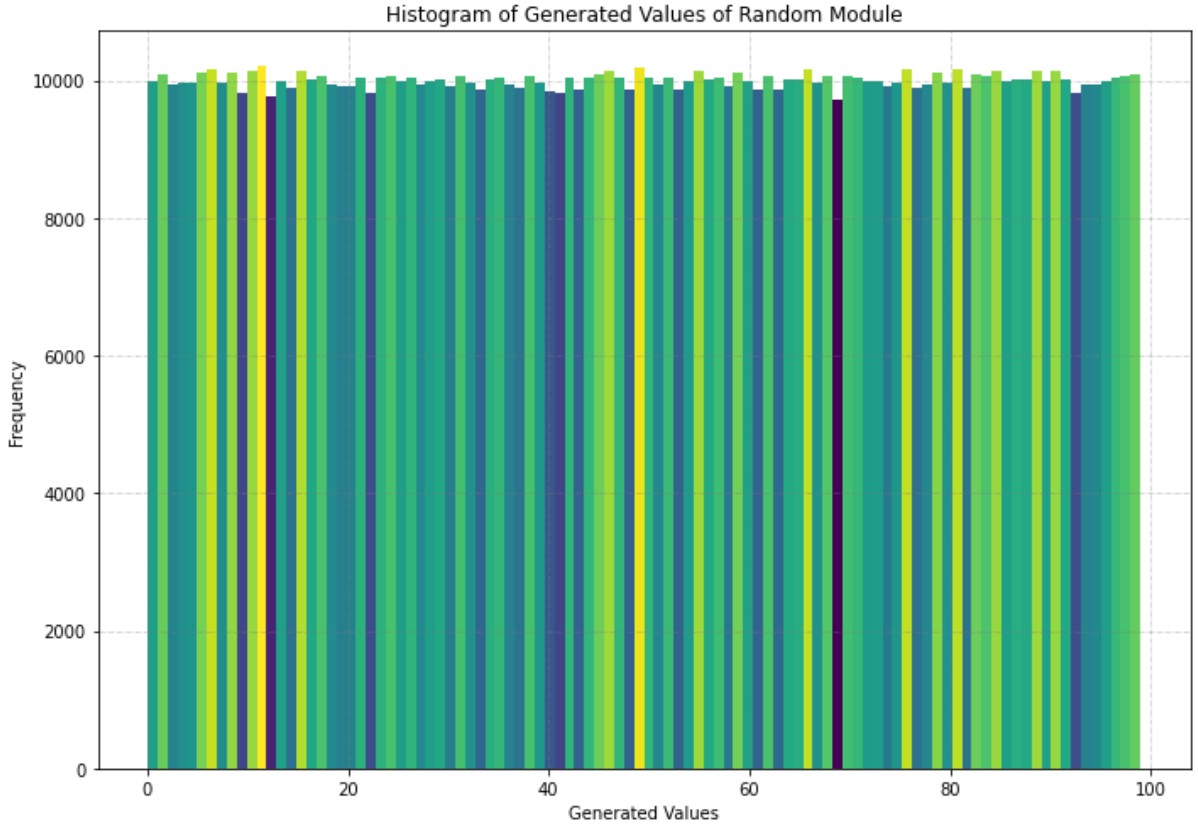
Problem 2: Is the random module random?

LOs: #EmpiricalAnalysis

HCs: #confidenceintervals

We check if NumPy's random module generates values that appear to be random. If the model is random, we should get integers selected from a range at random approximately equally often. Use the code below to generate an array of 1,000,000 random integers from 0 to 99.

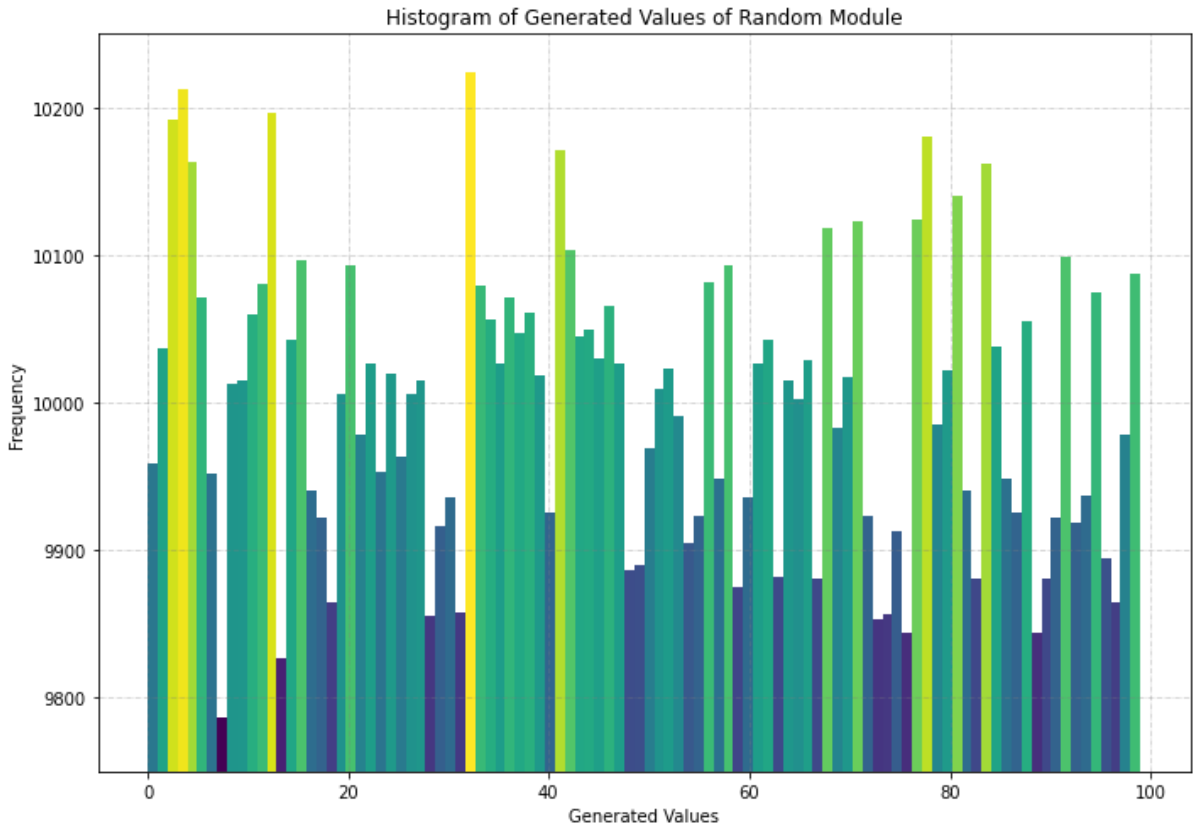
1. Make a properly formatted histogram of the number of times each value appears.



2. Discuss in detail how often you expect each value to appear, how often each value actually appeared, and whether the distribution over the results matches what you would expect according to the Central Limit Theorem.

Here we run a NumPy random module to generate (given the code provided) one million random integers from 0 to 99. Provided this, one would expect each value to appear equally often, so given that there are 100 values, each one should appear 10,000 times. After making a properly formatted histogram of the number of times each value appears, we notice that there is approximately half of values that have a frequency of less than expected and the other half have a frequency of more than expected. According to the Central Limit Theorem, taking sufficiently large random samples then we would expect some distribution over the results; in

general, the central limit theorem talks about the normal distribution; here we expect the values to be uniformly distributed. So, that means that generating a sufficiently large number of values between 0 and 99 we would expect that there is almost an equal number of each. However, the nature of the randomness itself in the distribution is not deterministic in that that will happen.



Problem 3: Forest fire mean-field approximation

LOs: #TheoreticalAnalysis

Consider the following forest fire model described in Drossel, B., & Schwabl, F. (1994). Formation of space-time structure in a forest-fire model. Each site of a 2-dimensional grid is occupied by a tree, a fire (a burning tree), or it is empty. During each discrete update, the following rules are applied.

- fire → empty, with probability 1.
- tree → fire, with probability 1 - g if at least one neighbor in its Von Neumann neighborhood is a fire. (Think of g as the immunity of a tree to catching fire.)
- tree → fire, with probability (1 - g) f if no neighbor is burning. (Think of f as the probability of a lightning strike or some other cause of a spontaneous fire – a small probability quite close to 0.)
- empty → tree, with probability p.
- Otherwise, the state of the cell remains the same.

Prove the equations in the paper in (2.1) for the change of the mean-field densities during one timestep.

$$\Delta\rho_e = \rho_f - p\rho_e$$

$$\Delta\rho_t = p\rho_e - \rho_t(1-g)\left(f + (1-f)\left(1 - \left(1-\rho_f\right)^{2d}\right)\right)$$

$$\Delta\rho_f = -\rho_f + \rho_t(1-g)\left(f + (1-f)\left(1 - \left(1-\rho_f\right)^{2d}\right)\right)$$

Note that $d = 2$ is the dimension of the grid.

In the forest fire mean-field approximation calculations, provided with the equations of the mean density of empty sites, mean density of trees, and mean density of burning trees in the steady state; we expect the change Δ of the mean-field densities to change from one timestep to the other.

Naturally, given this model, there are three states, empty, tree, burning, so the sum of their densities should be equal to 1.

$\rho_e + \rho_t + \rho_f = 1$ and $\rho_f = p\rho_e$

In the steady state, the mean of burning trees ρ_f is equal to the mean number of growing trees, so given that there is an empty site after a burning tree and a probability of a tree in an empty site the first equation shows that the change in the mean field density would be equal to the burning tree minus the growing trees (in the empty sites). The mean density of trees and burning trees are given by the equations.

Problem 4: Wireworld

LOs: #PythonImplementation

HCs: #algorithms

The Wireworld model is a cellular automaton that can be used to simulate simple electrical circuits. Even though these circuits are simple, the Wireworld CA is Turing-complete. This means you can use it to simulate a general-purpose computer like your laptop – assuming you have enough storage, computation, and patience to run the CA.

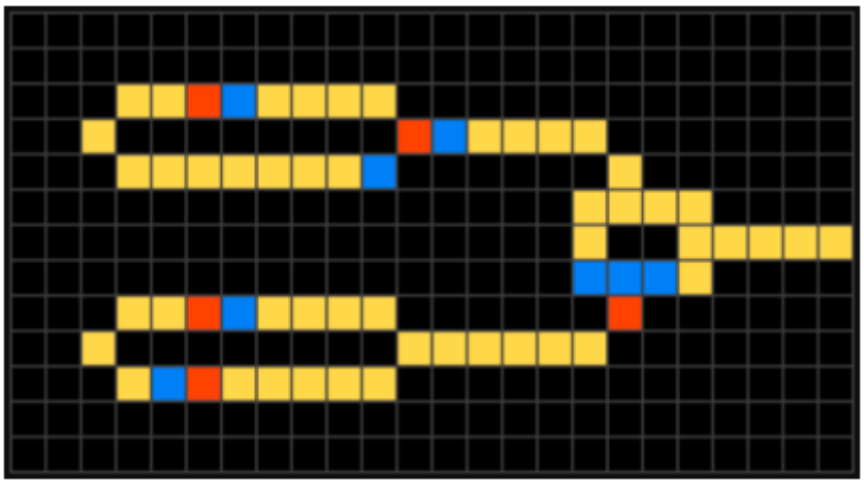


Figure 1. An example of a Wireworld state. Black cells represent the background, yellow cells represent wires, and blue and red cells represent electrons (head and tail, respectively).

Each cell is in 1 of 4 states–

0. background
1. electron head
2. electron tail
3. wire

In each discrete time step, the update rules are as follows.

- Background remains background.
- Electron head changes to electron tail.
- Electron tail changes to wire.

- Wire changes to electron head if and only if one or two of its neighbors are electron head. Otherwise, it remains wire. Use a Moore neighborhood to count the number of neighbors.

Implement this simulation in Python and use your implementation to reproduce the animation in Figure 1 above. Include the animation in your Python notebook.

In the Wireworld simulation, black cells, yellow cells, blue, and red cells are the 4 different states that they can be on, and on each time step the update rules perform a change in state.

```
38      def step(self):
39          new_world = WireworldSimulator.__new_world()
40
41          for x, y in self.world[Color.yellow]:
42              n = sum((x+dx, y+dy) in self.world[Color.blue] for dx, dy in self.dd)
43              if n == 1 or n == 2:
44                  color = Color.blue
45              else:
46                  color = Color.yellow
47              new_world[color].append((x, y))
48
49          new_world[Color.red] = self.world[Color.blue]       # head -> tail
50          new_world[Color.yellow] += self.world[Color.red]    # tail -> conductor
51
52          self.world = new_world
53          return self.world
54
```

The simulation for the Wireworld simulation can be found at https://github.com/cesar-ca/cs166-problem-set and shows a video of the simulation running as a YouTube link. https://www.youtube.com/watch?v=Xg_MEStWTrU