Honors TA Lab Proposal – Population Dynamics

Outlined below is a proposal for an Engineering Models lab, to be completed during the Engineering Models II course. This proposal serves as an artifact for the Honors TA experience. The MATLAB code to implement this lab has been created and is fully functional. Interested individuals should contact Alex Muir at muiram@mail.uc.edu.

Introduction to Discreet Time Analysis

In every field of engineering, it is often necessary to analyze the behavior of extremely complex systems. Examples of such systems include the cooling of a metal rod in a pool of liquid, the transfer of materials throughout a chemical plant, and the flow of electricity in a circuit. If a strict mathematical analysis of such systems is performed, a set of high-order differential equations often results. However, it is usually extremely difficult (or even impossible) to solve such systems in a timely manner, if at all. This lab demonstrates a way to simplify such problems through the use of *discreet time analysis*. This method ignores the exact solutions to differential equations, and instead estimates the system's behavior. This estimate is usually 'good enough' for preliminary engineering purposes.

Consider a model that involves the cooling of a metal rod. To solve the problem exactly would require the solution of cylindrical heat transfer equations (Bessel functions, yikes!). To avoid these arduous solutions, we can analyze the system from moment-to-moment. At the starting point in time, t = 0, the rod is at a given temperature, T_o, and a simple set of algebraic equations (the energy balance) govern the system. Solving these equations is trivial, and they yield answers that tell how much heat is transferred from the rod *when the rod is at temperature* T_o. For example, this number may be 10 Joules/second. Now, imagine that $1/10^{th}$ of a second goes by. According to our estimate, 1 Joule would be transferred from the cylinder. We can then use the fact that 1J was transferred from the cylinder in 0.1 seconds to calculate a new temperature at t = 0.1. We repeat the analysis for as many seconds as we wish to predict.

The discreet time analysis method is essentially a numerical solution to a differential equation. MATLAB has built in functions to do this, so it is natural to wonder why discreet time analysis is necessary at all. The answer is that the underlying differential equations *do not have to be known* in order to perform discreet time analysis! By knowing only a few fundamental equations, we can model the system quite efficiently!

The example in this lab considers population dynamics. The differential equations describing the system have a solution, but we desire a simpler analysis method. Therefore, instead of solving the system of differential equations, the population will be analyzed at discreet steps in time.

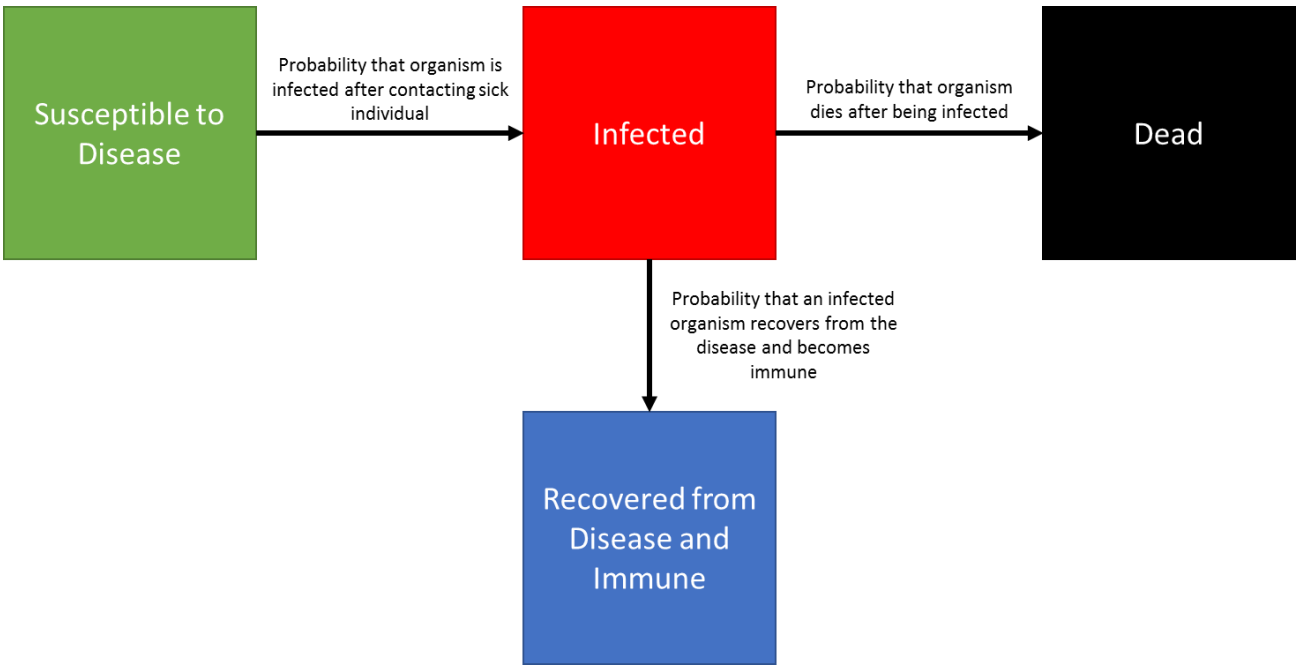
Consider a population that involves two groups of people – susceptible (average joes, healthy) and infected (sick with a disease that causes cravings for coffee). Assume that we know that 20% of the susceptible are infected every day. This is great for coffee companies, and so we complete the analysis below to predict how much coffee to produce. To obtain the estimated infected values at day two, we calculate $Infected\ (Day\ 2) = Infected\ (Day\ 1) + 0.2 * Susceptible(Day\ 1)$. We then complete the same calculation for 5 days. For comparative purposes, the exact solution is also shown.

	Day 1	***Day 2***	***Day 3***	***Day 4***	***Day 5***
Susceptible	10	8	6.4	5.12	4.1
Infected (Estimate)	0	2	3.6	4.9	5.9
Infected (Exact)	0	1.8	3.3	4.5	5.5

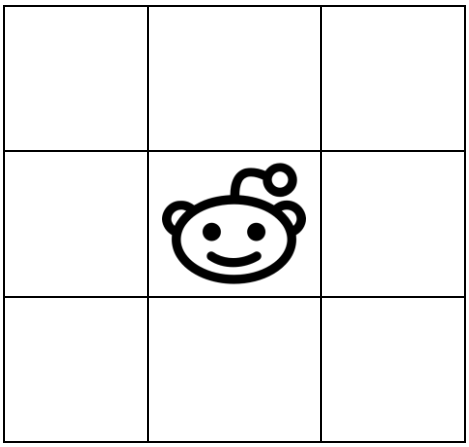
As is shown above, we obtained a pretty good estimate without solving any differential equations! If we make the time difference even smaller (1/2 day instead of 1 day), the estimates will get even better.

Model Overview

The model that we'll be using in this lab is slightly more complicated than the example above, and uses an adapted version of the SIR population model[1]. It specifies 4 color-coded categories, and these categories are shown in the graphic below. Organisms live in each colored box, and move between different boxes depending on their surroundings.



Our model will show the progression of a disease through a population. It assumes that all members are stationary at all times (probably a horrible assumption, by the way. If you feel like taking this lab one step further, try making healthy organisms flee from infected areas). To explain how this will work, imagine an organism surrounded by 8 other organisms:



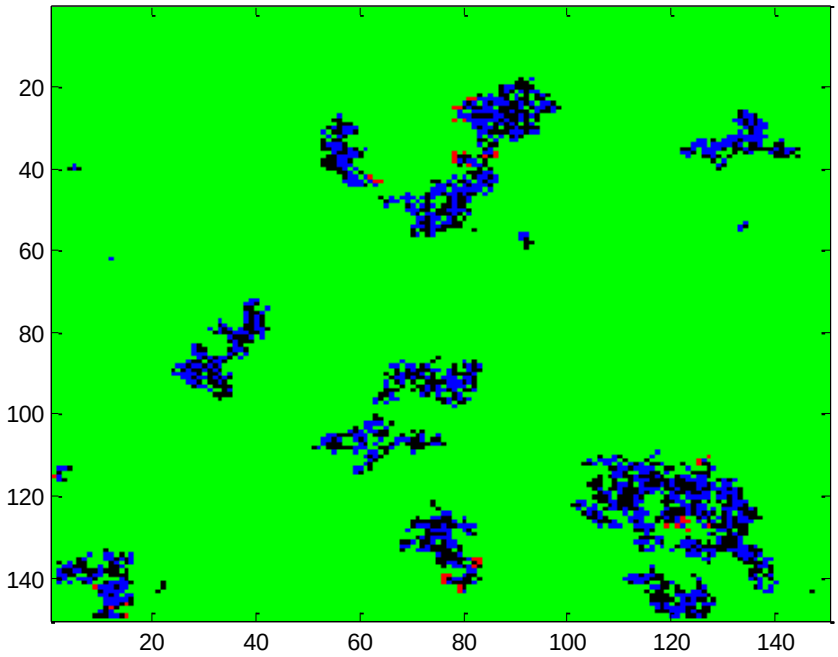
Via the laws of the universe (e.g., neglecting heredity and other acts of nature) the fate of Tiny in round 2 (the alien in the center of the square) depends on who is around Tiny in round one. A brief brainstorming session reveals that the following situations exist:

1. If Tiny is healthy in round 1 (henceforth R1), and Tiny is surrounded by all healthy aliens, there's no way he can get sick! So, he's healthy in round 2 (R2).

[1] https://services.math.duke.edu/education/ccp/materials/diffcalc/sir/sir1.html

2. If Tiny is healthy in R1 but surrounded by infected individuals, there's some chance that he'll be sick in R2. This is controlled by how *aggressive* the disease is.
3. If Tiny is infected in R1, he's either going to be recovered and immune in R2, or dead. This is governed by how *deadly* the disease is.
4. If Tiny is dead in R1, he's dead in R2 (another interesting twist: change 'dead' to 'undead', and run a zombie simulation).
5. If Tiny is immune in R1, then he can never be infected. So, he's immune in R2

By using the 5 simple rules above, we can develop a 2D simulation of a population of Tinys. The final results may look similar to the plot below:



Each pixel on the screen is an individual, and the animation will show how the disease moves through the population with time. The programmer is responsible for setting the initial number of sick individuals (placed randomly), the aggression of the disease, the deadliness of the disease, and the width of the screen (the above example is 150x150, for a total of 22,500 Tinys).

The method used to create the aforementioned animation is where discreet time analysis comes in. To see how this works, replace Round 1 and Round 2 (described above) with Round `N` and Round `N+1`. At *any* point in time, we can generate a new map of the population (round `N+1`) by analyzing the way that the population is in the present time (round `N`). Then, when we apply the five rules above to many, many Tinys instead of only one, we generate the full-fledged population, and are able to watch a disease spread through it.

Benefits and Programming Concepts

Discreet time analysis forces students to consider time-dependent problems from a different light than they are likely used to. Instead of conceptualizing the entire system for all values of time at once (visualizing a graph of the system response, for example), students will be introduced to problem solving methods that analyze the process one step at a time. This way of thinking removes the mathematical heavy lifting that surrounds most engineering problems, and instead allows students to focus on what is fundamentally happening in the system. In accordance with the example above, this is the difference between understanding Bessel functions and visualizing the heat map of a cylinder through time, and understanding that the cylinder is transferring heat to the water, which heats up the water, which makes the cylinder transfer heat at a slower rate because the temperature difference between the metal and the water is smaller now. Both solutions yield the same result, but one is based on mathematical formalism, while the other is based on a fundamental understanding of the system.

With respect to programming, this lab focuses on the use of loops, if-statements, and random numbers. The use of loops is obvious – the program must analyze the population repeatedly, and the number of generations (loops) varies depending on population parameters, so students are not able to use a simple for-loop. The tricky part is deciding when to stop looping, which (in my initial, crude solution), involves subtracting the previous generation's matrix from the current generation's and performing a double sum. If the double sum is zero (i.e., if the population did not change at all in the last generation), then I assume that the simulation is complete. The use of if-statements is also intuitive, as seen in the Model Overview section.

Random numbers are what allow the simulation to behave differently given the same inputs (disease deadliness, etc), resulting in a unique simulation each time that the program is run. Consider a case where a sick organism has a 75% chance of dying. To decide the fate of the organism in the next round, we first generate a random number between 0 and 1. If the number is truly random, then there is a 75% chance that the number will be at or below the value of 0.75, and a 25% chance that the number will be greater than 0.75. Thus, if the random number generated is greater than the probability that the organism will die, the organism gets to live, and vice versa. This exposes students to the use of random numbers to simulate physical situations based on chance.

Finally, this lab introduces students to the concept of mapping. By completing this lab, the students will recognize that the population is not stored on the computer as it is displayed on screen. That is, to say, the population is not stored as an image. Rather, the population is stored as a matrix of numbers 0-3 (one for each possible organism condition), and this matrix is mapped onto the display. This is a powerful technique that often saves greatly on processing time and computational simplicity in larger programs, especially for computer-based games.