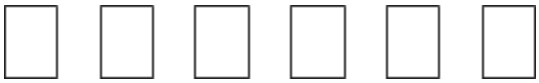
Modeling the Spread of an Infection in 1-D

(*Adapted from Scientific Computing with Case Studies*)

"When faced with a spreading infection, public health workers would like to predict its path and severity in order to make decisions about vaccination strategies, quarantine policy, and use of public health resources. This is true whether dispersal of the pathogen is natural (as in SARS or the 1918 flu) or deliberate (for example, through terrorism). Effective mathematical models form a way to test the potential outcome of a public health policy and arrive at an effective response."

Imagine the following scenario: a hospital ward has one row of patients in beds as shown below:



Suppose that one day (we'll call it Day 1), one of the patients gets sick. We'll show this by shading one of the beds.

Now suppose that because of the closeness of the beds, there is a 100% chance that the infected patient will pass on the illness to the neighbors in the beds on the left and right. We'll limit the speed of the spreading illness by saying that neighbors can only be infected every 24 hours. On Day 2, the hospital ward would look like this:

Suppose also that this disease only lasts 2 days, and after that, the patient recovers and is immune to the disease.

1) On your own paper, draw the spread of the disease by making daily diagrams like the ones above. Remember that the disease only lasts 2 days and after that the patient recovers and cannot catch the disease again.
   Check this.

2) How many days will it take for all the patients to become immune?
   6 days

All kinds of questions arise from this simple model, and we can answer many of them by using Monte Carlo simulations. Each patient in this model can be identified as 1) **infected**, 2)

susceptible to infection, or 3) **recovered** and immune from further infection. Here are some of the questions we might wonder:

- What proportion of people in the hospital ward are infected, susceptible, and recovered as time passes?
- How does this change if we increase or decrease the length of the illness?
- How does this change if we decrease the probability that the neighbors of an infected patient will become infected? (For example, what if there was only a 50% chance the neighbors would get the disease?)

These are questions about numbers, or *quantitative*, questions. We might also wonder:

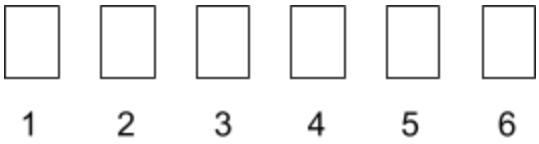
- If we were to visualize the disease spreading (as in our diagrams), what would that spread look like?

This is a question about something descriptive that cannot be measured with numbers, or a *qualitative*, question.

We want to try to program this situation into Mathematica and use Monte Carlo simulations to try to answer all these questions.

This seems pretty complicated at first glance, so let's think about how to do it. First we need to put in place a system for storing information about each of the patients. And, we need to think about how to communicate information about the patients to Mathematica because it won't understand the words "infected," "recovered," or "susceptible."

The best way to do this is by using numbers! First we'll number the beds like this:



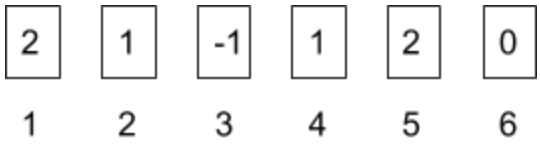
This is called *indexing*, and the number that tells us the placement of each bed is called an *index*.

Then we'll describe each of the "states" with a number as well. If a person is **recovered** (and immune), they will be described by a **-1**. If a person is **susceptible** to infection, they will be described by a **0**.

The tricky description is for people who are infected. Remember that an illness can last more than one day, so we don't want to assign just one number for someone who is infected. We might want to know whether they have been sick for 1 day, 2 days, 3 days, etc. *In fact*...maybe we want to put a countdown on a person who is sick, so we know when they'll recover. For

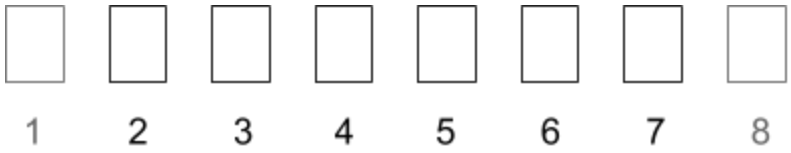
example, suppose an illness lasts 4 days. If a person gets infected, we would assign them a 4, the next day they would be decreased to a 3, meaning they have 3 days of sickness left. The next day, they would be decreased to a 2, etc. So, a person will be **infected** if they are assigned a number **greater than 0**, and that number will be the number of sick days they have left.

3) Using the system outlined above, describe in words the information shown in the diagram below.



There is another tricky thing that we need to think about before starting to write a program in Mathematica. We know that patients in beds to the right and left of an infected patient will also become infected. It is possible to write this as a rule, but we want to make sure we can use this rule for each bed. So what happens for beds 1 and 6? Bed 1 does not have a left neighbor, and bed 6 does not have a right neighbor. Mathematica will not be able to ignore this and will come up with an error. These end beds are exceptions to the rule, and they will make the program more difficult to write.

BUT...we can get rid of these exceptions by adding "ghost beds" to the left of Bed 1 and to the right of Bed 6. We can renumber the beds, apply our original rule to the middle 6 beds (now Beds 2 - 7), and only pay attention to the results in those middle beds. Our new diagram would look like this:



4) So, keeping all these things in mind, let's try to program this!
   a) Open Mathematica and create a new notebook.

   b) Save the notebook as "Infection1D." (We're going to try a 2D version later!)

   c) First we want to define some variables: the length of the illness (call this **k**), the probability of neighbors getting the disease (call this **p**), the number of days we want to observe our hospital ward (call this **tmax**), and the number of hospital beds (call this **m**). Type:

      k = 2;
      p = 1;
      tmax = 10;

m = 6;

We are randomly choosing a 10 days for now. We can go back and change any of these variables later to see how they affect the outcome. It is possible we may need more days.

d) For now, we want to create a table of zeros to store all our patient information. It needs to have space for all **m** beds (and our two ghost beds!) for each of the **tmax** days, so

H = Table[0, {i, 1, m + 2}, {j, 1, tmax}];

You might want to run this statement without the semicolon to see what you get.

e) Now we want to introduce a sick person on Day 1. We will try to put the person in the middle of the beds since it will be more interesting if they are surrounded by about the same number of people on each side. Remember that to access a space in a table, you have to use the table name and double brackets "**[[]]**". Type:

H[[(m + 2)/2, 1]] = k;

Here we are finding the middlemost bed on the first day and giving it a value of **k** to show that this person just became sick. We find the middlemost bed by taking our **m** beds and our two ghost beds, and dividing by 2. The **1** indicates the day.

At this point, you might want to type **H**, and run your statements to see how **H** has changed.

f) Here comes the difficult part: now we are going to write an *algorithm*, some rules for Mathematica to follow many times over, to get our results. First we want to tell Mathematica to perform these rules **tmax** times, so that we will know what happens on each of the 10 days.

To do this, we use a "for" loop:

For[t = 1, t < tmax, t++,
(*stuff we fill in later*)
]

Mathematica will go through the whole numbers starting from 1 all the way to **tmax** (or 10), and it will insert that number wherever it sees a **t** in the rules we write. The **t++** in the for loop tells Mathematica to increase the **t** values by 1 after

it runs through all the rules. The **t = 1** tells Mathematica the starting value for **t**, and the **t < tmax** tells Mathematica to go all the way to the biggest whole number that is still less than **tmax**.

What would that number be for our **tmax** value of 10?
9

g) We'll separate the inside of the "for" loop into two sections: i) we want to update the countdown for any sick people and ii) we want to add any new infected people.

i) To update the countdown for sick people, we add code inside the "for" loop as shown below. Let's break down the code. **H[[All, t + 1]]** means we are assigning the values for **All** the beds on the next day (**t + 1** means the current day plus another). To do that, we create a table which has **m + 2** spaces for the **m + 2** beds.

Then we say how to tell what each space in the table will be.

We know that sick people will be assigned a number greater than 0. If the number in the current table is greater than 1, then we just need to decrease that number by 1. If the current number is equal to 1, then the person is on their last day of sickness and needs to be switched to recovered (-1) for the next day. If neither of these are true, then the person should keep their current value. Add the highlighted lines to your code:

```
For[t = 1, t < tmax, t++,
    H[[All, t + 1]] =  Table[
    If[H[[i, t]] > 1,
    H[[i, t]] - 1,
    If[H[[i, t]] == 1,
    -1,
    H[[i, t]]
    ]
    ],
    {i, 1, m + 2}];
    (more stuff)
]
```

Discuss the code with your group. Do you see how the statements above correspond to the written description?
This is open-ended.

ii) To add new infected people, we want to go through each of the current values for each person. In order to infect someone, we need three things to be true: 1) they need to be susceptible (their current value has to be 0), 2) we need to see if the probability determines that they get sick, and 3) they need to be the neighbor of someone who is currently sick.

For this section, we can ignore the ghost beds, so we add a second "for" loop inside the original "for" loop. Add the highlighted lines to your code:

```
For[t = 1, t < tmax, t++,
    H[[All, t + 1]] =  Table[
    If[H[[i, t]] > 1,
    H[[i, t]] - 1,
    If[H[[i, t]] == 1,
    -1,
    H[[i, t]]
    ]
    ],
    {i, 1, m + 2}];

    For[i = 2, i <= m + 1, i++,
    If[RandomReal[] <= p && H[[i, t + 1]] == 0,

    anyNeighborSick = (H[[i - 1, t]] > 0 || H[[i + 1, t]] > 0);

    If[anyNeighborSick,
    H[[i, t + 1]] = k
    ]
    ]
    ];
]
```

How do we know this inside "for" loop is ignoring the ghost bed positions? Because it starts at 2 and ends at **m + 1**, ignoring the first and last positions.

In Mathematica **&&** means "and" and **||** means "or." **RandomReal[]** will create one random decimal number between 0 and 1. Discuss with your group members what you think each of the following parts of the code means:

RandomReal[] <= p

H[[i, t + 1]] == 0

anyNeighborSick = (H[[i - 1, t]] > 0 || H[[i + 1, t]] > 0)

H[[i, t + 1]] = k

h) The giant "for" loop takes care of figuring out all the information for each day. (You might be curious to see what it looks like, so type **H** and run everything to see what you get.) Next we would like to try to visualize everything. Enter the following statement:

```
Manipulate[ArrayPlot[{H[[2 ;; m + 1, t]]}, Mesh -> True,
ColorRules -> {-1 -> Gray, 0 -> White, _ -> Black}], {t, 1, tmax, 1}
]
```

Manipulate[] will allow us to basically create a movie of our infection spreading. **H[[2 ;; m + 1, t]]** means only the middle beds (and not the two end ghost beds) will appear. The **ColorRules** section assigns different colored squares to recovered (gray), susceptible (white), and infected (black). Run the statement. You should get a box with a bar where you can manipulate the value of **t** and see what happens as time passes.

i) Remember we also had some questions about the proportion of infected, susceptible, and recovered people. Add the following lines to count the number of each type of person on each day and plot the information:

```
Sus = Table[Count[H[[2 ;; m + 1, i]], 0]/m, {i, 1, tmax}];
Rec = Table[Count[H[[2 ;; m + 1, i]], -1]/m, {i, 1, tmax}];
Inf = Table[1 - Sus[[i]] - Rec[[i]], {i, 1, tmax}];
ListLinePlot[{Inf, Rec, Sus},
PlotLegends -> LineLegend[{"Infected", "Recovered", "Susceptible"}],
AxesLabel -> {"Days", "Proportion of Population"}]
```

What do the first three lines do? Discuss with your group members. You may have to look up the **Count[]** function. Run the statements and see what you get. The first two lines count the number of susceptible and recovered people for each day and divides them by the total number of beds, yielding the proportion of susceptible and recovered people for each day. Since we know that each person must be in one of the 3 states, we can subtract those proportions from 1 to get the proportion of infected people. That is what the third line does.

5) Check with your team leader to make sure your code is correct. Now you have two tools to qualitatively and quantitatively observe how the infection spreads. Do some

investigating! Try some of the following things and write down any observations you make:

a) Change your code so that there are 50 beds and the simulation runs for 100 days.

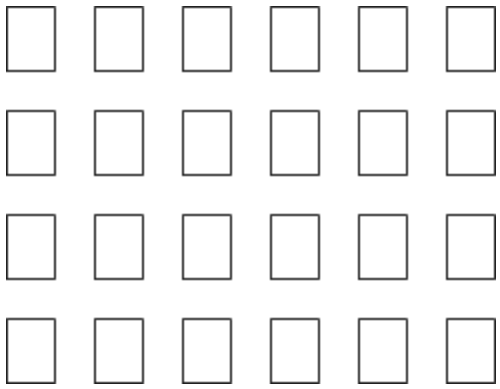
b) Try longer and shorter illness lengths by changing the value of **k**. What do you notice in the spread and the plot?

c) Try decreasing the probability that the neighbors become infected. Try benchmark numbers like 0.25, 0.5, 0.75. Try other numbers if you're curious. Observe what happens and write down your observations. Try different lengths of the illness with these new probabilities as well. What do you notice?
This section is all open-ended; however, the students should notice that when the probability is 1, the infection travels in two sections, each with a length equivalent to the length of the illness.

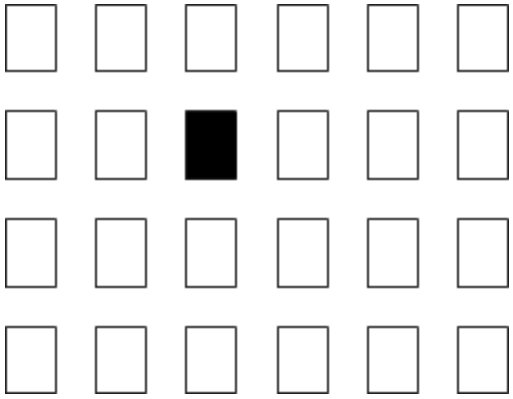
They should notice that for lower probabilities it is possible that the infection dies out very quickly and that not everyone is immune afterwards.

Modeling the Spread of an Infection in 2-D

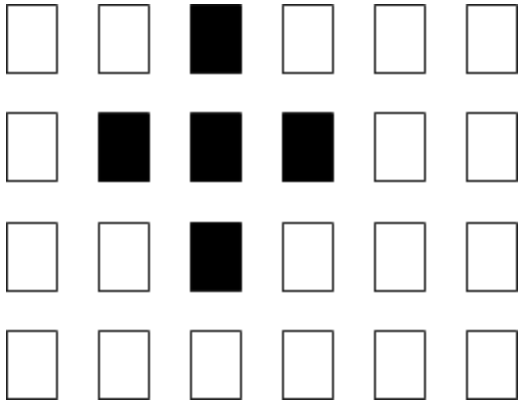
Now let's try doing the same thing in two dimensions! Now the hospital ward has patients in beds positioned in rows and columns as shown below:



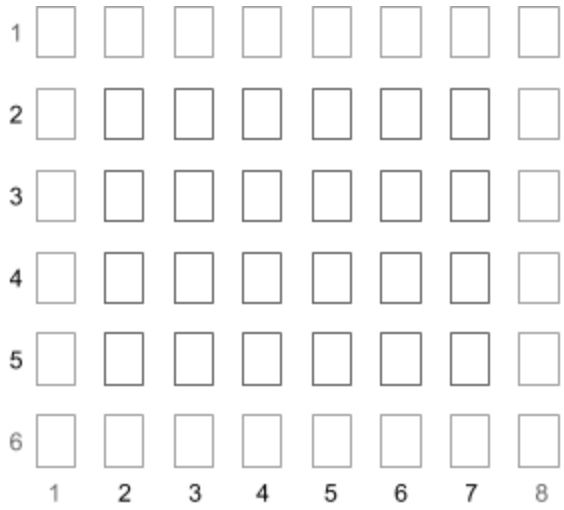
Suppose that one day, one of the patients gets sick.



This time, there is a 100% chance that the neighbors in the beds on the left, right, above, and below will get the infection. The next day, the hospital ward would look like this:



How can we revise our 1-D code to model this 2-D situation? It's going to be very similar. We just need to add an extra dimension. Since the beds are placed in rows and columns, we will need an index set for the rows and an index set for the columns. We'll also need to have ghost rows and ghost columns:



1) Let's try modifying the code:
    a) Copy your Infection1D notebook and save it as Infection2D.

    b) Set
    k = 2;
    p = 1;
    tmax = 50;
    m = 4;
    n = 6;

    The last row introduces a new variable, **n**, that represents the number of columns. Now in 2-D, **m** represents the number of rows of beds.

    c) Before we made **H** a 2-D table, now it will be a 3-D table:

    H = Table[0, {i, 1, m + 2}, {j, 1, n + 2}, {k, 1, tmax}];

    d) And when we define our sick person, we have to use the 3-index system:

    H[[(m + 2)/2, (n + 2)/2, 1]] = k;

    e) Discuss with your group members what changes you think will need to be made to the rest of the code. Here are some hints:

i) Any statement that has **H[[ ]]** will need to be changed so that the 3-index system is used inside the brackets.

ii) Any **Table[]** statements will need to be changed to include an extra dimension. You will have to decide what that dimension will be.

iii) The inner "for" loop that determines if a patient becomes sick will now have to have an additional "for" loop inside of it (and inside of that will be the rules). In the end, you will have 3 "for" loops. The outermost goes through the days, represented by **t**. The middle loop goes through the rows of the hospital beds represented by **i**. The innermost loop goes through the columns of hospital beds represented by **j**.

iv) The definition for **anyNeighborSick** will need to be changed to include statements for the neighbors above and below.

v) These statements apply to the **Manipulate[]** and plot sections of the code too. Those will have some changes as well.

f) When you think you've arrived at a working code, run your statements. Double check with your team leader to make sure everything is correct.

2) Once your 2-D code is working, start experimenting!
   a) Set **m** and **n** to be 50 and **tmax** to be 100.

   b) Run your code with **p = 1;** and try changing the length of the illness, **k**. See what patterns emerge.

   c) Now try decreasing **p**. What do you observe?

   d) Try different values of **p** and **k**. Are there values of **p** and **k** for which some people do not become immune? (For some of these experiments, you may have to increase your **tmax** value.)

This section is open-ended but they should notice similar things as in the 1-D case. There are some really great animations particularly for longer illness times (like 10 days) with low probability (around 0.1).