

CS 1112 Spring 2020 Project 2 Part A due Monday, Feb. 17, at 11:00 PM

(Part B will appear in a separate document. Both parts have the same submission deadline.)

You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. *Adhere to the Code of Academic Integrity.* For a group, “you” below refers to “your group.” You may discuss background issues and general strategies with others and seek help from the course staff, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not OK for you to see or hear another student’s code and it is certainly not OK to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on your own, seek help from the course staff.

Objectives

Completing this project will solidify your understanding of for-loops, while-loops, and nested loops (Chapters 2 and 3 in *Insight*) and give you practice with the accumulation pattern. You will also continue to explore MATLAB graphics.

Ground Rule

Since this is an introductory computing course, and in order to give you practice on specific programming constructs, some MATLAB functions/features are forbidden in assignments. Specifically, the use of arrays and the `break` and `continue` commands is *not* allowed in this project. Only the built-in functions that have been discussed in class and in the assigned reading may be used. For this project, do not write user-defined functions; we will do that later in Project 3.

1 Spreading disease

Epidemiologists study the propagation of infectious diseases using idealized mathematical models. These can predict how quickly a disease will spread, how much of the population will get infected, what vaccination rate is necessary to prevent an epidemic, etc. A common approach is to divide the population into three categories: *Susceptible*, *Infectious*, and *Recovered*. A susceptible person may contract the disease through contact with an infectious person, becoming infectious themselves. After some time, they will recover and become immune to the disease (no longer susceptible).

You will analyze this model by simulating contact, infection, and recovery over multiple days. You will need to translate our description of the model into mathematical operations, then code those formulae into Matlab and surround them with appropriate input, output, and control statements in order to satisfy the project requirements. The model depends on the following parameters:

- N : Total population size
- I_0 : Number of people initially infected
- k : Average number of close contacts a person makes with other people each day



- β : Probability that a susceptible person will become infected during a close contact with an infectious person, becoming infectious themselves the next day
- γ : Probability that an infectious person will recover at the end of any given day

(we have used single-letter parameter names to keep our formulae concise, but you can and should use more descriptive names in your code). For now, assume that everyone who is not initially infectious is susceptible.

The *state* of our model at any given time consists of three variables:

- d : How many days it has been since the disease was introduced
- S : The number of susceptible people in the population on day d
- I : The number of infectious people in the population on day d

(Note that the number of recovered/immune people R is not an independent state; you can compute it from N , S , and I .) Each day, some number of susceptible people become infected (move from S to I), and some number of already-infectious people recover (move from I to R). The expected number of new infections is given by:

$$S \left(1 - \left(1 - \beta \frac{I}{N} \right)^k \right)$$

(Derivation: the chance that one person in S becomes infected after k contacts is the complement of the chance that they don't become infected in each of k contacts. In each contact, I/N is the chance that the other person is infectious, and β is the chance that their infection spreads.)

The expected number of recoveries is simply $I\gamma$ (the number of infectious people times the chance that they recover). Newly-infected people are not eligible for recovery until the next day.

Write a script `diseaseSpread` that performs the following tasks:

1. Prompt the user for the expected number of close contacts a person makes in a day. Ensure that they enter a positive number less than 100.
2. Set the other parameter values to represent the following scenario: in a population of ten-thousand people, one-hundred of them are initially infectious. If an infectious person contacts a susceptible person, there is a 30% chance that the latter becomes infected. And each day, an infectious person has a 25% chance of recovering.
3. Simulate the model for 30 days.
4. Plot the number of infectious people (y -axis) vs. the number of days (x -axis) that have been simulated.
5. Print how many people are in each population category at the end of the simulation.

Run your script for a scenario in which each person makes 5 close contacts with others per day. Looking at your plot, determine the number of people who were infectious on the sickest day. **Write a comment** at the bottom of your script documenting your finding (use complete sentences). The number of people in each category may be fractional; think of them as averages over many trials.

Your plot should consist of a point for each day. You may choose which shape and color of marker to use. **Label** your axes (use `xlabel()` and `ylabel()`), and don't forget to include units. A note on timing: at the start of "day 0," I_0 people are infectious (include this point on your plot). In moving from day 0 to day 1, $I_0\gamma$ people have recovered, while some number of susceptible people have become infected; the point at $d = 1$ should reflect these changes to I .

In Project 3 we will continue to explore this model, so be kind to yourself and make sure your code is clearly written and well-documented.

2 Guessing game

2.1 Your turn to guess

You are going to make a simple guessing game in which the program generates a random integer from 1 to 200 and a human player gets 8 chances to guess the secret number. On each guess the program will tell the user whether their guess is too high, too low, or correct. As soon as the player guesses the number correctly or uses up their 8 guesses, the program will no longer accept guesses, and the output should be either “Congratulations!” or “Try again next time; the number was N ” (where N is the number the user failed to guess). **Write a script** `guessGame` based on the above specifications.

The output from an example run of the script is shown below (user input is shown in *italics*):

```
I am thinking of an integer between 1 and 200 (inclusive).
Guess the number (8 guesses left):  150
Your guess was too low.
Guess the number (7 guesses left):  200
Your guess was too high.
Guess the number (6 guesses left):  187
Congratulations!
```

Hints:

1. Use `ceil(200*rand())` to generate a random integer between 1 and 200. `ceil()` is the “ceiling” function, which rounds a number up to an integer. (Can you figure out why this works? What would happen if you used `floor()` or `round()` instead?)
2. Observe that the prompt for user input in the example above includes a variable value. Use the `sprintf()` function to format data into a string, which can be saved to a variable; that variable can then be used as the argument to the input function. See, e.g., `buttonClick` from Project 1 or the input validation example from Lecture 6.

2.2 Computer’s turn to guess

It’s time to turn the tables! In this next game, the human player will pick the number (again from 1 to 200), and your *program* must attempt to guess it within 8 tries. After each guess by the program, the user must indicate whether the guess was too high (H), too low (L) or correct (C). The program should stop after guessing correctly (reporting how many guesses it took) or after it runs out of guesses (printing “I give up.”). **Write a script** `numberGuesser` that tries to guess your secret number. You may implement any strategy for your program’s guesses so long as they are integers and consistent with the clues (that is, if one of its guesses is too high, it’s not allowed to guess the same or larger number in the future). Try to write a program that wins as often as possible (can you get it to mimic your own guessing strategy?)

The output from an example run of such a program is shown below (user input is shown in *italics*):

```
Think of your number between 1 and 200. Press Enter when ready.
Is 100 too [H]igh, too [L]ow, or [C]orrect? (8 guesses left):  L
Is 150 too [H]igh, too [L]ow, or [C]orrect? (7 guesses left):  H
Is 125 too [H]igh, too [L]ow, or [C]orrect? (6 guesses left):  C
I won in 3 guesses.
```

Note that the human player does not type in their chosen number in this game. We assume that the human player remembers their own chosen number and appropriately types L, H, or C after each guess made by the program.

Hints:

1. In order for the input function to accept text input (instead of numeric input) from a user, you need to use the `input()` function with a second argument, 's' (for “string”). For example:

```
letter = input('Give me a letter: ', 's')
```

2. Relational operators can be used on characters, e.g., `var == 'L'` evaluates to true if variable `var` stores the character L and evaluates to false if `var` stores anything else.

Submission

Submit your files `diseaseSpread.m`, `guessGame.m`, and `numberGuesser.m` on CMS (after registering your group).

Part **B** will appear in a separate document. Parts A and B have the same due date.