CS 1112 Spring 2020 Project 3 due Wednesday, March. 4, at 11:00 PM

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 you own, seek help from the course staff.

Objectives

Completing this project will solidify your understanding of user-defined functions and vectors (1-d arrays) and allow you to continue exploring MATLAB graphics. You will perform *simulation* and *sensitivity analyses*, both of which are important concepts and tools in computational science and engineering.

Ground Rule

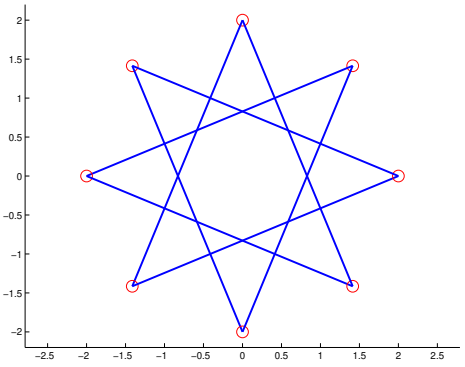
As usual, use only the functions and constructs learned so far in the course.

1 Spirograph stars

Implement the following function as specified:

```
function myStar(pts, inc, r)
% Draw a star with `pts` points that lie on a circle with radius `r`.
% The lines of the star connect the `i`th point with the `(i+inc)`th
% point (wrapping around).  The points of the star are highlighted
% with markers.
```

The function call `myStar(8,3,2)` should produce the figure shown on the right: an 8-pointed star with points that lie on a circle with radius 2 where each point is connected to the point 3 forward from it. The function call `myStar(5,2,1)` creates a traditional five-pointed star with points that lie on the unit circle. Choose marker and line colors as you wish. See Project 1 if you need a reminder on how to plot markers and lines, and you can also search MATLAB documentation (top-right search box) for clarification. The following graphics commands should be used to set up the figure window:




```
close all                                  % Close all figure windows
figure                                     % Open a new figure window
bound= 1.1*r;
axis([-bound bound -bound bound])          % Axis limits (center over origin)
axis equal                                 % Equal scaling for x and y axes
hold on                                    % Allow multiple plot cmds to stack
```

Use the command `hold off` at the end of your function in order to return MATLAB to its default graphics state.

Add an extra comment line to the header comments of function `myStar()` to answer this question: what shape is drawn if parameter `inc` has the value one?

As part of your solution, you must **implement** and make effective use of the following function in drawing the star:

```
function [x, y] = polarRad2xy(r,theta)
% (x,y) are the Cartesian coordinates of polar coordinate (r,theta).
% theta is in radians.
```

Note the different units assumed by this function vs. `polar2xy()` from lecture – comments are important!

2 Containing disease

In Project 2, you simulated how a disease spread throughout a susceptible population. While most people eventually recovered in our model, being sick is still no fun (and consequences can be far worse for more vulnerable members of society). In this problem, you'll explore strategies for limiting the spread of disease. By conducting a *sensitivity analysis*, you'll reveal how different parameters of the model impact the end results.

2.1 Functions and vectors

Your simulation code from Project 2 was a MATLAB *script* that prompted its user for a parameter value and conveyed its results via a plot and some printed lines of text. Performing a sensitivity analysis "by hand" with such a script would be tedious, but if instead the simulation were to exchange its inputs and outputs directly with other code, we could automate the process. Therefore, your first step will be to convert your script into a *function*.

Along the way, we'd like to make a few changes to the simulation's interface:

- One of the function's inputs will affect the initial number of *resistant* people R_0 in the population (previously this was assumed to be 0). The total population size remains the same ($N = 10000$), so think about how this will affect your initial state variables.
- The recovery probability γ should also be an input to the function (rather than being hard-coded to 25%).
- You no longer need to ensure that the number of contacts per day k is between 0 and 100 (a function can't ask for different input, so for now let's trust our callers to provide reasonable values).
- Instead of plotting and printing, the function should return two *vectors* of outputs storing the number of susceptible and infectious people on each day.

This suggests the following function specification:

```
function [Svec, Ivec] = simDisease(contacts_per_day, prob_recovery, r_frac)
% Run a discrete SIR model to simulate the spread of a disease.
% `contacts_per_day` is the number of contacts people have each day.
% `prob_recovery` is the probability that an infectious person recovers
% each day.  `r_frac` (between 0 and 1) is the initial fraction of the
% population that is already resistant to the disease.
% Returns two vectors containing the number of susceptible (`Svec`) and
% infectious (`Ivec`) people on each day.  Index 1 corresponds to day 0
% (initial conditions).
% The total population size is 10000, the initial number of infected people
```

```
% is 100, and the probability of infection during a contact is 30%.  The
% model is run for 30 days (so the output vectors will contain 31
% elements).
```

Using your Project 2 submission as a base, **implement** the above function. The header can be downloaded from the course website as `simDisease.m` so you don't need to re-type everything, but you still need to read the comments carefully. Be sure to remove any "side effects" from your old code that are not part of this function's specification.

Test your function by invoking it with `contacts_per_day=5`, `prob_recovery=0.25`, and `r_frac=0` and then plotting `Ivec`; the results should look similar to what you found in Project 2. Be prepared to show this test to a consultant or TA.

Tip: Once the solutions for Project 2 have been posted to CMS, compare their results with yours in order to check for errors in your model. You should fix any bugs you find this way, but remember not to copy the solution code directly into your own files.

2.2 Vaccination

Vaccinating people against a disease can be an effective way to limit its spread *if* enough of the population receives the vaccine (a phenomenon known as "herd immunity"). In our model, vaccinated people start out in the "resistant" group R, so the ratio R_0/N corresponds to the vaccination rate. We are interested in how the spread of the disease changes with different vaccination rates.

Write a script `diseaseSensitivity` that runs your simulation for ∼5–7 different vaccination rates between 70% and 94% and plots the infectious population vs. time for each case. For this study, the number of contacts per day should be 5, and the recovery probability should be 25%. Give the plot a title, label its axes, and provide a legend labeling the different rates (see below). Your script must call your `simDisease()` function to perform the simulation.

A simulated disease is said to become an *epidemic* if the infectious population increases from its initial size. With high enough vaccination rates, epidemics can be avoided. Based on your plot, what vaccination rate is required to prevent a disease with these parameters from becoming an epidemic? **Write a comment** in your script justifying your answer.

Hints

1. To plot multiple points as a single curve, form a vector of x-coordinates (e.g. a range of day numbers) and a vector of y-coordinates (e.g. counts of infected people) and pass those vectors as arguments to `plot()`. For example, `plot(xs, ys, 'r-')` will plot a red curve connecting the points (`xs(1)`,`ys(1)`), (`xs(2)`,`ys(2)`), ...

2. To plot multiple curves with different colors, simply use `hold on` and omit the color code from the line style argument. MATLAB will automatically assign a new color to each curve.

3. To label plotted curves, we pass two additional arguments to `plot()`: first the string `'DisplayName'`, then a string containing the label. To include the value of a variable in the label, look to our old friend `sprintf()`. For example, to label a curve by a variable `k`, use `plot(xs, ys, '-', 'DisplayName', sprintf('k=%d', k))`. To actually show a legend with the labels, use `legend show`.

2.3 Limiting contact

We expect that the more people are in contact with one another, the further a disease will spread. We might also expect that a disease from which people recover quickly won't spread as far. We'd like to show the effects of both of these parameters, but doing so on a 2D plot is a little tricky. First, we need to summarize a simulation with a single number (rather than showing data from all 30 days at once). Then

we'll plot that number on the y-axis, vary one parameter (say, recovery rate) along the x-axis, and plot multiple curves for different values of the remaining parameter (say, contact rate).

What number best summarizes the results of an entire simulation? Ideally, we want very few people to ever get sick at all. If we assume this is a new disease for which no vaccine exists, then the size of the susceptible group S at the end of the simulation tells us how many people were never infected (bigger is better). We'll use this to judge the effect of limiting contact.

To plot a single curve for multiple probabilities of recovery γ, you'll need to collect the summarized results of multiple simulation runs into a vector. Since you'll need to do this multiple times (once for each different contact rate k), it's a great candidate for making a function! Since the utility of this function may be limited beyond this analysis, let's skip the work of making a new m-file and instead declare it as a subfunction in your analysis script.

At the bottom of `diseaseSensitivity`, **implement** the following function (don't neglect the comments):

```
function [Ss, recovery_probs] = sweepRecoveryRates(contacts_per_day)
% Simulate disease propagation for a range of recovery rates.
% `contacts_per_day` is the number of contacts people have each day (this
% value is used by each simulation in the sweep).  No one is initially
% resistant.  Returns `recovery_probs`, a vector of the different
% recovery probabilities used in the sweep (covering 0 to 1 in 10%
% increments), as well as `Ss`, a vector of final suscecptible population
% sizes corresponding to the recovery rates in `recovery_probs`.
```

Note that, in a script, subfunctions *must* go at the bottom of the script, and each subfunction *must* end with the keyword `end`.

Next, just above your subfunction, **add code** to your script that performs several sweeps over recovery probabilities for different contact rates k spanning the range from 1 to 7 contacts per day. In a new figure (with title and axis labels), plot a curve for each sweep. As before, label the curves with a legend. Based on your plot, assuming each person only has 3 close contacts per day, what daily recovery probability would be required to ensure that at least 3/4 of the population never becomes infected? **Write a comment** justifying your answer.

3 Collatz's hailstones

The Collatz conjecture, named after mathematician Lothar Collatz, states:

> Start with any natural number n. If n is even then divide it by two; otherwise multiply it by three and add one. Repeat the process indefinitely and you will eventually get 1.

The conjecture (proposed in 1937) and the sequence of numbers involved (often called the "hailstone sequence") have intrigued many. The conjecture had been tested for all numbers up to 10^{20} and 1 was always reached. Every few years there would be rumbling of a successful proof of the conjecture, but in fact the conjecture is still unproven and remains an open question! Here are the first few sequences and their lengths:

n	Sequence	Sequence Length
2	2 1	2
3	3 10 5 16 8 4 2 1	8
4	4 2 1	3

3.1 Hailstones

Write a function `myCollatz()` that has an input parameter `n` and returns a vector storing the hailstone sequence starting with `n`, following Collatz's conjecture. As discussed in lecture, you must write a concise function comment describing the function and its parameters.

Test your function! You should call your function to check the cases shown above. You can also do a quick visualization to see the "hailstone behavior," referring to how hailstones go up and down in a cloud before crashing to Earth. Here's an example:

```
sequence= myCollatz(3);
plot(1:length(sequence), sequence)
```

3.2 Is there a pattern to the sequence length?

Implement the following function:

```
function nStar= collatzLengths(nMax)
% Plot the lengths of the Collatz sequences for starting values from
% 2 to `nMax`.  Assume `nMax` is a positive integer greater than 2.
% Returns `nStar`: the starting value for the longest sequence found.
% If multiple sequences have the longest length, `nStar` is the largest
% of those starting values.
```

Notes on program requirements:

- Make effective use of function `myCollatz()`.
- Plot the length of the sequence versus the starting value (i.e., n on the x-axis). Use a marker of your choice to indicate each length; do not connect the markers with lines. Be sure to label both axes. The title of your plot should indicate the maximum sequence length and the associated starting value `nStar`.

Submission

Submit your files `polarRad2xy.m`, `myStar.m`, `simDisease.m`, `diseaseSensitivity.m`, `myCollatz.m`, and `collatzLengths.m` on CMS (after registering your group).