rec12

April 30, 2022

1 RECITATION 12, ORIE 3120

1.1 Stochastic Optimization with Epidemic Models

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (10,10)
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
```

1.1.1 Background

In lecture, you've been learning about how mathematical models can be used to guide decisions about inventory. Models are useful for many kinds of decisions, not just for inventory. In this recitation, we'll see how models of epidemics can be used to guide decisions about how to respond to epidemics. We'll use a simulation to make predictions from the model, which is similar in many ways to simulations that can be built to make predictions for the performance of Q,R inventory policies.

SIR (*susceptible*, *infectious*, and *recovered*) models are one kind of epidemic model, and has been widely used to make decisions about COVID-19.

SIR models work by splitting the population into three categories: - *susceptible*, which refers to people who have not yet been infected - *infectious*, which refers to people who have been infected by the disease and are now able to infect susceptible individuals - *recovered*, which refers to people that are no longer able to transmit it to susceptible individuals.

In this recitation, you'll construct the basic components of an SIR model and see what happens to the course of the epidemic as you modify parameters within the model.

We can use a discrete time simulation to create the SIR model - what this means is that at each period of time in the simulation (which is a discrete period of time after the previous period), we simulate the changes that occur between the previous state and the current state, and use that to set the current state.

The three categories will be modeled at each discrete period of time, so they will be functions of time, t.

As such, the total number of people within each category are represented as the following functions: - $S(t)$ = total susceptible people - $I(t)$ = total infectious people - $R(t)$ = total recovered people

But we are also concerned about the proportion of the total population within each category. We model this with the following functions, which can be obtained by dividing the population in each category by the total number of people in the population, N. - $s(t) = \frac{S(t)}{N}$ = proportion of the population that is *susceptible* - $i(t) = \frac{I(t)}{N}$ = proportion of the population that is *infectious* - $r(t) = \frac{R(t)}{N}$ = proportion of the population that is *recovered*

1.2 Part 1: Constructing the SIR Simulator

1.2.1 Defining `step()`

Here, we define a function called `step()`, with parameters `state, beta, gamma`. It increments our simulation, calculating new values for the number of *susceptible*, *infectious* and *recovered* people over a single unit of time.

`state` is the current state of the simulation, which consists of three components: `state['susceptible']`, `state['infectious']` and `state['recovered']` (representing the proportions $s(t)$, $i(t)$ and $r(t)$, respectively).

Note that β (`beta`) represents the average number of contacts per person per period of time.

Likewise, γ (`gamma`) represents the transition rate, that is the percentage of infectious people that transition into the recovered state in any period of time.

Within our `step()` function, we need to generate a new `state` based on the previous `state`, as well as the `beta` and `gamma` parameters.

Hint:
- *Infectious*
To determine `state['infectious']` for the next time period, i.e. $i(t+1)$, we need to consider both the new infections over this time period and the recoveries over this time period. - New infections: Each infectious person comes into contact with `beta` people, which have a `state['susceptible']` chance of being susceptible. - This can be expressed as $\beta s(t) I(t)$ - Recoveries: Each infectious person has a `gamma` percent chance of recovering during this time period. - This can be expressed as $\gamma I(t)$ - Combined: The change in the number of infectious people between one state and the next is the number of new infections minus the number of recoveries. - This can be expressed as $\beta s(t) I(t) - \gamma I(t)$ - If you divide both sides of the equation by the total number of people, N, you'll arrive at the change in the proportion of the total population that is infectious - $\frac{\beta s(t) I(t) - \gamma I(t)}{N} = \beta s(t) i(t) - \gamma i(t)$ - Add this to the current value of `state['infectious']` to get `state['infectious']` for the next time period.

- *Susceptible*
  - Because susceptible people can only transition to infectious, this means that the number of susceptible people decreases by the number of new infections.
    - This can be expressed as $-\beta s(t) I(t)$
  - To get the change in the percent of the population that is susceptible, divide this number by the size of the population, N
    - In other words, $\frac{-\beta s(t) I(t)}{N} = -\beta s(t) i(t)$

        * Add this to the current value of `state['susceptible']` to get `state['susceptible']` for the next time period
- *Recovered.*
  - At each time period, the number of recovered people increase by the number of recoveries from the infectious state.
    * i.e. $\gamma I(t)$
  - Dividing this by N fives the percentage of the population that has recovered in this time period.
    * $\frac{\gamma I(t)}{N} = \gamma i(t)$
    * Add this to the current value of `state['recovered']` to get `state['recovered']` for the next time period

Question 1:
Complete `step(state, beta, gamma)` below, which will simulate the changes that occur between one time period and the next in the SIR model.

[ ]:
```
def step(state, beta, gamma):
  # your code here
  # new_infectious = COMPLETE THIS
  # new_recovered = COMPLETE THIS
  # state['infectious'] = COMPLETE THIS
  # state['susceptible'] = COMPLETE THIS
  # return state
```

Question 2:
Set $s(0) = 0.99, i(0) = 0.01, r(0) = 0, \beta = 0.5$ and $\gamma = 0.1$.
Run the simulation for 60 time periods and plot the percentage of the population in each category at each time period.

Your chart should be similar to the following:

[ ]:
```
# starting state
state = {'susceptible' : 0.99, 'infectious' : 0.01}

# run simulation and plot graph
# HINT: use a for loop and lists that you can append current values to
```

Question 3:

In the previous run of the model, at what value of `state[susceptible]` does the number of infectious people stay the same or decrease between time t and time $t + 1$?

In the graph, this is the point where the number of people and proportion of the population that is infectious begins to fall.

This is known as *herd immunity.*

YOUR ANSWER HERE

Question 4:

Make plots similar to what you made for Question 3, but change β (i.e. the number of contacts an infectious person has during a time interval).
What happens to the progression of the epidemic if β is made larger or smaller?

YOUR ANSWER HERE

Question 5:

Now, try varying the value of γ (i.e. the recovery rate for an infectious person in a single unit of time).

Make plots similar to what you made for Question 3, but change γ .
What happens to the progression of the epidemic if γ is made larger or smaller?

YOUR ANSWER HERE

Question 6:
Now, let's extend the simulation. Instead of having deterministic steps, let's create a function called `random_step()` (similar to the previous `step()` function) where the number of new infections and new recoveries is not a deterministic function, but is instead generated from a binomial distribution.

Thus, the number of new infections is $binom(n = S(t), p = \beta i(t))$. - Recall that $S(t)$ is equal to $(N * $ `state['susceptible']`), and that the change in `state['infectious']` is the number of new infections divided by N.

Additionally, the number of new recoveries is $binom(n = I(t), p = \gamma)$. - Recall that $I(t)$ is equal to $(N * $ `state['infectious']`), and that the change in `state['recovered']` is the number of new recoveries divided by N.

For the binomial distribution, use `np.random.binomial()`

[ ]:
```
def random_step(state, beta, gamma, popsize):
  # YOUR CODE HERE
  pass
```

Question 7:

Remake some of your plots from earlier but with `random_step()` instead of `step()`. Are there any differences?

Additionally, try varying the population size. Does this cause the result of `random_step()` to be more or less similar to the plot obtained with `step()`?

YOUR ANSWER HERE

1.3 Part 2: SIR With Asymptomatic Screening

Part 2: Now we are going to study what happens when there is asymptomatic screening. We adjust our step function so that it now takes in a `fraction_tested` argument. We note that when we conduct asymptomatic screening and detect a positive, we can quarantine them, effectively removing them from the infectious pool of people. You will make this change below:

Question 8: Similar to when we defined `step`, adjust to account for asymptomatic screening for the function `asympto_step`. This can simply be done by adjusting the `new_recovered_or_quarantined = (gamma + fraction_tested) * state['infectious']`

[ ]:
```
# fraction_tested is the fraction of the population being tested every day
def asympto_step(state, beta, gamma, fraction_tested):
  # YOUR CODE HERE
  pass
```

We will now find the appropriate amount of asympotmatic testing we need to conduct to ensure that infectious on day t+1 are smaller than on day t. This is the "safe" level of testing.

Question 9: Simulate across multiple `fraction_tested` from a range of {0.1, 0.2, 0.3, 0.4, 0.5}. Moreover, set $s(0) = 0.95, i(0) = 0.05, r(0) = 0, \beta = 0.5$ and $\gamma = 0.1$. Run the simulation for 60 time periods and plot just the infections across each `fraction_tested` on a single plot

[ ]:
```
frac_tests = [0.1, 0.2, 0.3, 0.4, 0.5]
# YOUR CODE HERE
```

From the graph you generated, you should see that as you increase `fraction_tested`, the number of people infected at any given time is smaller. Because there is a cost associated with tested excessively, we can't conduct asymptomatic screening on the entire population, but we still want to find the testing level where the number of infections is a monotonically decreasing function.

NOTE: monotonically decreasing simply means the function or vector has successive values that are either the same or smaller.

Question 10: From the graph you generated, what is the smallest `fraction_tested` you can select that maintains the property that infections at time t+1 is either the same or smaller than at time t?

YOUR ANSWER HERE

Since we don't actually know β, we want to set our `fraction_tested` to be high enough so that it accounts for many values of β. We will find the required `fraction_tested` for a given β value and make a plot of `fraction_tested` vs β

Question 11: Simulate across multiple β from a range of {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}. Moreover, set $s(0) = 0.95, i(0) = 0.05, r(0) = 0$ and $\gamma = 0.1$. For each β, you will need to find the smallest `fraction_tested` that results in a monotonically decreasing `infectious` vector. For this, test `fraction_tested` over {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1} and find `fraction_tested` that results in a decreasing infections vector. We have provided a function to help you with this. Run the simulation for 60 time periods and plot `fraction_tested` vs. β

[ ]:
```
betas = fracs = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]

def monotonically_decreasing(lst):
    return all(x>=y for x, y in zip(lst, lst[1:]))

# YOUR CODE HERE
```

[ ]:
```
# ANSWER:
# GENERATE YOUR PLOT HERE
```

[ ]:
```

```