Modelling and Simulation of COVID-19 Vaccine Prioritization by Age Groups

By: **Youssef Elmougy**

1 Content

Vaccine Prioritizations using Cellular Automata

1 Introduction

The COVID-19 pandemic has caused a major crisis worldwide, with case transmissions between individuals still occurring on a wide scale. In an effort to decrease the case transmissions in the long term, COVID-19 vaccines are urgently required to be distributed to the population. However, a limited quantity of the COVID-19 vaccines is available to be administered. This raises concerns on how to prioritize the COVID-19 vaccine doses among the population. The goal of this part of the the project is to apply the Cellular Automata Model to achieve a well motivated and data-driven decision on which age group to prioritize in the vaccine distribution plan with the goal of slowing down infection and death rates to protect the general population from the effects of the COVID-19 virus.

This part analyzes COVID-19 vaccination prioritization using the Cellular Automata Model, where the cells in the model represent the demographics of the United States population with each cell having one of the following possible states: (1) Susceptible S, (2) Symptomatically Infected I_S, (3) Asymptomatically Infected I_A, (4) Recovered R, (5) Deceased D, or (6) Vaccinated V. With this configuration, we aim to display the spread of the COVID-19 virus simultaneously with the active distribution of the COVID-19 vaccine to the population while prioritizing each of the age groups under consideration. Using the data from this model, we discuss the effectiveness of prioritizing each age group according to reductions in deaths and infections.

2 Setting up the model

This section includes all the helper functions and parameters needed to run the Cellular Automata Model.
We import the following packages which will be used throughout the notebook:

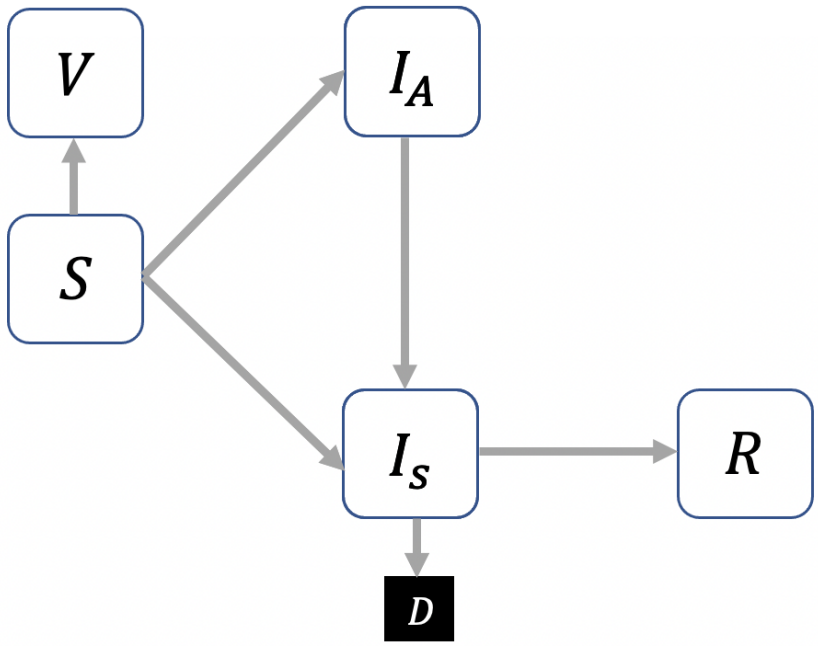
1. **NumPy**: Adds support for large and multi-dimentional arrays.
2. **MatPlotLib**: Powerful tool to generate data plots.
3. **Random**: Allows for generating random numbers and samples from various distributions.
4. **iPyWidgets**: Interactive sliders for plots to visualize changes in input parameters.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import random
from ipywidgets import interact
```

In our nxn grid $W = W(t)$, cells evolve over time. Where time t is measured in days. Every cell in W could be in one of the following possible states:

- **SUSCEPTIBLE (S):** The cell has not contracted the virus and can be infected if they are surrounded by infected cells. All cells are initially susceptible.
- **ASYMPTOMATICALLY INFECTED (I_a):** The cell is infected with the virus but is *not aware* of the infection. It has a *higher probability* of spreading the virus to neighboring cells.
- **SYMPTOMATICALLY INFECTED (I_s):** The cell is infected with the virus but is *aware* of the infection. It has a *lower probability* of spreading the virus due to quarantining measures, and can only infect immediate cells.
- **RECOVERED (R):** The cell has fully recovered from the virus. It cannot be reinfected or infect other cells.
- **DECEASED (D):** The cell has passed away from the virus.
- **VACCINATED (V):** The cell has been fully vaccinated, *yay*! It has full immunity and cannot be infected or infect other cells.

The transition between states is displayed by the following figure:



The associated integers for each state are:

[ ]:
```
EMPTY = -1
SUSCEPTIBLE = 0
ASYMP_INFECTED = 1
SYMP_INFECTED = 2
RECOVERED = 3
DECEASED = 4
VACCINATED = 5
```

The following function builds our 2D grid W and visualizes the 2D world. Each possible state is taken into consideration and plotted with a unique color on the spectrum:

[ ]:
```
def createGrid(world, vmin=EMPTY, vmax=VACCINATED, states="states"):
  assert states in ["states", "bool"]
  if (states == "states"):
    ticks = range(vmin, vmax+1)
    labels = ['Empty', 'Susceptible', 'Asymptomatic Infected', 'Symptomatic␣
↪Infected', 'Recovered', 'Deceased', 'Vaccinated']
  else:
    ticks = [0,1]
    labels = ['False (0)', 'True (1)']

  plt.pcolor(world, vmin=vmin, vmax=vmax, edgecolor='black')
  colorBar = plt.colorbar()
  colorBar.set_ticks(ticks)
  colorBar.set_ticklabels(labels)
  plt.axis('square')
  h = world.shape[0]-2
  w = world.shape[1]-2
  plt.axis([0, h+2, 0, w+2])
```

The following functions help us in extracting useful information for the cells on the grid W. Each possible state has its own helper function for more effective use throughout the notebook. Each state helper function provides the following information:

1. Locations of all cells in the grid W that are of that state.
2. Number of all cells in the grid W that are of that state.
3. Ratio of the cells of that state with respect to the total number of cells in the grid W.

[ ]:
```
def susceptible(grid):
  locationsSusceptible = (grid == SUSCEPTIBLE).astype(int)
  numberSusceptible = len(np.where(grid == SUSCEPTIBLE)[0])
  ratioSusceptible = numberSusceptible / ((grid.shape[0]-2) * (grid.shape[1]-2))
  return locationsSusceptible, numberSusceptible, ratioSusceptible

def asympInfected(grid):
  locationsAsympInfected = (grid == ASYMP_INFECTED).astype(int)
  numberAsympInfected = len(np.where(grid == ASYMP_INFECTED)[0])
  ratioAsympInfected = numberAsympInfected / ((grid.shape[0]-2) * (grid.
↪shape[1]-2))
  return locationsAsympInfected, numberAsympInfected, ratioAsympInfected

def sympInfected(grid):
  locationsSympInfected = (grid == SYMP_INFECTED).astype(int)
  numberSympInfected = len(np.where(grid == SYMP_INFECTED)[0])
  ratioSympInfected = numberSympInfected / ((grid.shape[0]-2) * (grid.
↪shape[1]-2))
  return locationsSympInfected, numberSympInfected, ratioSympInfected
```

```
def recovered(grid):
  locationsRecovered = (grid == RECOVERED).astype(int)
  numberRecovered = len(np.where(grid == RECOVERED)[0])
  ratioRecovered = numberRecovered / ((grid.shape[0]-2) * (grid.shape[1]-2))
  return locationsRecovered, numberRecovered, ratioRecovered

def deceased(grid):
  locationsDead = (grid == DECEASED).astype(int)
  numberDead = len(np.where(grid == DECEASED)[0])
  ratioDead = numberDead / ((grid.shape[0]-2) * (grid.shape[1]-2))
  return locationsDead, numberDead, ratioDead

def vaccinated(grid):
  locationsVacc = (grid == VACCINATED).astype(int)
  numberVacc = len(np.where(grid == VACCINATED)[0])
  ratioVacc = numberVacc / ((grid.shape[0]-2) * (grid.shape[1]-2))
  return locationsVacc, numberVacc, ratioVacc
```

The virus spreads among individuals symptomatically and asymptomatically. Likewise, at each time step in our simulation, the cells can either become symptomatically infected or asymptomatically infected according to the following conditions:

Asymptomatic Spread: The cell unknowingly spreads the virus to surrounding cells, so there is a higher probability of spread. The asymptomatic spread of the virus is indirectly proportional to the distance between the cell and its neighbors. The following quantifies asymptomatic spread:

- Cells **1** block away are infected with a probability of $asympRate\%$
- Cells **2** blocks away are infected with a probability of $asympRate^2\%$
- Cells **3** blocks away are infected with a probability of $asympRate^3\%$

Symptomatic Spread: Cells that are symptomatic will immediately quarantine safely away from others, so they only have the ability to infect cells that are 1 block away with a probability of $sympRate\%$.

The only cells that have the ability to be infected are ones in the state ***susceptible***. All vaccinated, recovered, and deceased cells have full protection from infection and spread.

The following function calculates the probability that the susceptible cells in the grid W become either symptomatically and asymptomatically infected:

[ ]:
```
def infectionSpreadProbability(grid, asympRate, sympRate):
  suscep = susceptible(grid)[0]
  asympInfect = asympInfected(grid)[0]
  sympInfect = sympInfected(grid)[0]
  asymp1 = np.zeros(shape=grid.shape, dtype=int)
  asymp2 = np.zeros(shape=grid.shape, dtype=int)
  asymp3 = np.zeros(shape=grid.shape, dtype=int)
  symp = np.zeros(shape=grid.shape, dtype=int)
```

```
  #Asymptomatic spread
  asymp1[3:-3, 3:-3] = asympInfect[2:-4, 3:-3] | asympInfect[3:-3, 4:-2] |␣
↪asympInfect[4:-2, 3:-3] | asympInfect[3:-3, 2:-4]
  asymp1 = asymp1 & suscep
  asympProbability1 = asympRate * asymp1

  asymp2[3:-3, 3:-3] = asympInfect[1:-5, 3:-3] | asympInfect[3:-3, 5:-1] |␣
↪asympInfect[5:-1, 3:-3] | asympInfect[3:-3, 1:-5]
  asymp2 = asymp2 & suscep
  asympProbability2 = (asympRate**2) * asymp2

  asymp3[3:-3, 3:-3] = asympInfect[0:-6, 3:-3] | asympInfect[3:-3, 6:] |␣
↪asympInfect[6:, 3:-3] | asympInfect[3:-3, 0:-6]
  asymp3 = asymp3 & suscep
  asympProbability3 = (asympRate**3) * asymp3

  #Symptomatic spread
  symp[3:-3, 3:-3] = sympInfect[2:-4, 3:-3] | sympInfect[3:-3, 4:-2] |␣
↪sympInfect[4:-2, 3:-3] | sympInfect[3:-3, 2:-4]
  symp = symp & suscep
  sympProbability = sympRate * symp

  totalProbability = asympProbability1 + asympProbability2 + asympProbability3 +␣
↪sympProbability

  return totalProbability
```

If a cell is asymptomatic, there is a latent period of **3 days**, after which the cell becomes symptomatically infectious. This function keeps track of all the *asymptomatically infectious* cells and changes their state to the *symptomatically infectious* state after the latent period:

[ ]:
```
def updateAsympInfected(grid, asympCells):
  asympInfect = asympInfected(grid)[0]
  asympCells += asympInfect

  becomeSymptomatic = np.where(asympCells > 3)
  asympCells[becomeSymptomatic] = 0
  grid[becomeSymptomatic] = SYMP_INFECTED
```

Following an infectious period of **5 days**, a symptomatically infectious cell either turns into the *recovered* state with a probability of *recoveryRate*% or into the *deceased* state with a probability of *deathRate*%:

[ ]:
```
def updateSympInfected(grid, sympCells, recoveryRate, deathRate):
  sympInfect = sympInfected(grid)[0]
  sympCells += sympInfect
  symp = np.where((sympCells) & (sympCells % 5 == 0))
```

```
  for cell in zip(*symp):
    number = random.random()

    if (number < recoveryRate):
      grid[cell] = RECOVERED
      sympCells[cell] = 0
    elif (number < recoveryRate + deathRate):
      grid[cell] = DECEASED
      sympCells[cell] = 0
```

This function controls the vaccine distribution among cells in the grid *W*. At each time step (each day), we have to vaccinate a total amount of (*vaccineRate* x *total cells*) cells in the grid *W*. The only cells with the ability to get vaccinated are susceptible cells. The function uses *random.choices* to get a random selection of susceptible cells from grid *W* to vaccinate:

[ ]:
```
def updateVaccinated(grid, vaccineRate, n):
  cellsToVaccinate = int(vaccineRate * (n*n))
  cellsVaccinated = 0

  susceptibleCells = susceptible(grid)[0]
  sus = list(zip(*np.where(susceptibleCells)))
  cells = random.choices(sus, k=cellsToVaccinate)

  for i in cells:
    grid[i] = VACCINATED
    cellsVaccinated += 1
```

The initial configuration of the world is made up of susceptible cells with a small fraction, *initialPercent*%, of asymptomatically infected cells selected at random, representing the ongoing spread of the virus at the beginning stage of vaccination. The world *W* is an *nxn* grid of cells. The function also initializes the *asympCells* and *sympCells* arrays which track the asymptomatically infected and symptomatically infected cells consecutively:

[ ]:
```
def createWorld(n, initialPercent):
  world = EMPTY * np.ones((n+6, n+6))
  asympCells = np.zeros(world.shape, dtype=int)
  sympCells = np.zeros(world.shape, dtype=int)

  world[3:-3, 3:-3] = SUSCEPTIBLE

  numInitialInfected = int((initialPercent/100) * (n*n))
  for _ in range(numInitialInfected):
    position = int(random.random() * n)
    world[position+3, position+3] = ASYMP_INFECTED

  return world, asympCells, sympCells
```

To simulate the real world as accurately as possible, we have to do the following at each time step:

1. **Vaccinate cells:** In order to successfully simulate the vaccine prioritization by age group, the function does the following:
   - It vaccinates an $ageGroupwantVaccine\%$ of the selected age group for prioritization until the full number of individuals/cells in that age group have been vaccinated.
   - Once everyone in the prioritized age group has been vaccinated, all other individuals/cells have an equal chance% to get vaccinated next.
2. **Update asymptomatically infected cells:** This calls the $updateAsympInfected$ function with the current $asympCells$ array in order to update all asymptomatic cells.
3. **Update symptomatically infected cells:** This calls the $updateSympInfected$ function with the current $sympCells$ array in order to update all symptomatic cells.
4. **Update susceptible cells:** Each susceptible cell's probability of infection is calculated using the $infectionSpreadProbability$ function. It then iterates through the susceptible cells to infect them based on the calculated infection probability.

[ ]:
```
def stepEvolution(grid, ageGroupPopulation, ageGroupwantVaccine,␣
↪vaccineRolloutRate, asympCells, sympCells):

  numVaccinated = vaccinated(grid)[1]
  if (numVaccinated <= int(ageGroupPopulation*ageGroupwantVaccine*n*n)):
    updateVaccinated(grid, vaccineRolloutRate*ageGroupwantVaccine, n)
  else:
    updateVaccinated(grid, vaccineRolloutRate*averageWantVaccine, n)

  updateAsympInfected(grid, asympCells)
  updateSympInfected(grid, sympCells, recoveryRate, deathRate)

  infectionProbability = infectionSpreadProbability(grid, asympRate, sympRate)

  for (x,y) in zip(*np.where(infectionProbability > 0)):
    number = random.random()

    if (number < infectionProbability[x,y]):
      grid[x,y] = ASYMP_INFECTED
```

The following are our model parameters:

1. **timeSteps:** The number of time steps through which the model evolves. Each time step is equal to 1 day.
2. **n:** The size of the nxn grid W.
3. **asympRate:** The asymptomatic infection rate which is the probability that a susceptible cell is infected one cell ($asympRate$), two cells ($asympRate^2$), or three cells away ($asympRate^3$) which are asymptomatic.
4. **sympRate:** The symptomatic infection rate which is the probability that a susceptible cell is infected only by direct neighbors which are symptomatic.
5. **recoveryRate:** The probability that a cell recovers from the virus.

6. **deathRate:** The probability that a cell passes away from the virus.
7. **vaccineRolloutRate:** The percentage of the population that is vaccinated per day.
8. **populationAgeGroups:** This is an array of size 5, where each index represents the percentage of population of an age group:
    - **Under 20**: 24.87%
    - **20-39 years**: 27.20%
    - **40-59 years**: 25.19%
    - **60-79 years**: 18.80%
    - **80+ years**: 3.94%
9. **wantVaccineAgeGroups:** This is an array of size 5, where each index represents the percentage of each age group that accepts and is not hesitant to take the vaccine :
    - **Under 20**: 75%
    - **20-39 years**: 65%
    - **40-59 years**: 55%
    - **60-79 years**: 75%
    - **80+ years**: 90%
10. **averageWantVaccine:** This is the average acceptance of the vaccine across all age groups. It is 72%.

It is important to note that all the preceding data was gathered from the most up to date data for the United States.

[ ]:
```
#timeSteps = 550
n = 100
asympRate = 0.0274
sympRate = 0.064
recoveryRate = 0.3
deathRate = 0.0181
vaccineRolloutRate = 0.002

numAgeGroups = 5
populationAgeGroups = [0.2487, 0.272, 0.2519, 0.188, 0.0394]
wantVaccineAgeGroups = [0.75, 0.65, 0.55, 0.75, 0.9]
averageWantVaccine = 0
for x in wantVaccineAgeGroups:
  averageWantVaccine += x
averageWantVaccine = averageWantVaccine / numAgeGroups
```

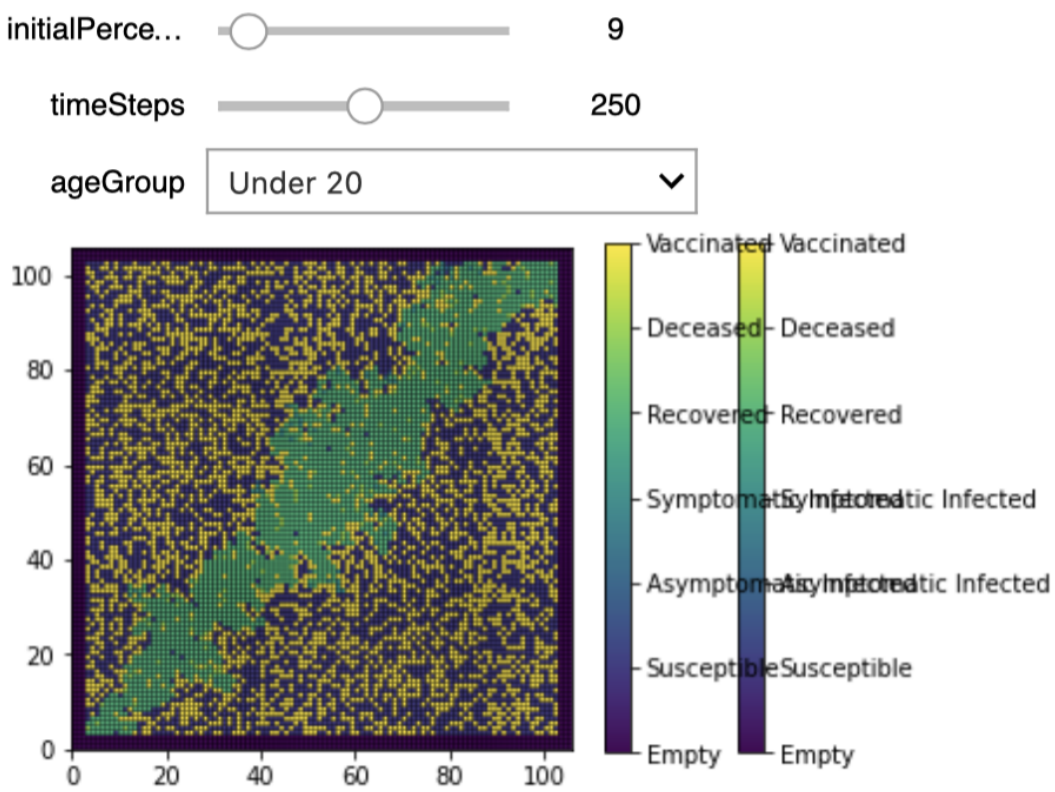
3 Cellular Automata Model

This is the main function of our model. It brings together all the previous functions to create a complete Cellular Automata Model that models both the symptomatic and asymptomatic spread of COVID-19, along with the vaccine rollout, prioritizing a single age group over others. The goal of the model is to apply the Cellular Automata model to achieve a well motivated and data-driven decision on which age group to prioritize in the vaccine distribution, with the goal of slowing down infection and death rates. Hence, the main function takes in, as parameters, the *initialPercentage* (percentage of the whole population that is initially infected), the *timeSteps*, and the *ageGroup* (the age group to prioritize for vaccine distribution). Using those parameters, the function creates a new world and iterates through a number of *timeSteps*, with each time step making a call to the appropriate *stepEvolution* function to prioritize the given age group.

The inputs to the model, *initialPercentage*, *timeSteps*, and *ageGroup*, can be toggled through the sliders available. This helps to better visualize the effects of adjusting the different parameters by interacting with the plot. Feel free to adjust the input parameters to see their effect on the COVID-19 virus spread and vaccine distirbution.

[ ]:
```
susceptibleList = []
asympInfectedList = []
sympInfectedList = []
recoveredList = []
deceasedList = []
vaccinatedList = []

def main(initialPercentage, timeSteps, ageGroup):
  world, asympCells, sympCells = createWorld(n, int(initialPercentage))

  createGrid(world)

  for i in range(timeSteps):
    suscep = susceptible(world)[1]
    susceptibleList.append(suscep/n)
    asympI = asympInfected(world)[1]
    asympInfectedList.append(asympI/n)
    sympI = sympInfected(world)[1]
    sympInfectedList.append(sympI/n)
    recover = recovered(world)[1]
    recoveredList.append(recover/n)
    dead = deceased(world)[1]
    deceasedList.append(dead/n)
    vacc = vaccinated(world)[1]
    vaccinatedList.append(vacc/n)

    if (ageGroup == 1):
      stepEvolution(world, populationAgeGroups[0], wantVaccineAgeGroups[0],␣
↪vaccineRolloutRate, asympCells, sympCells)
    elif (ageGroup == 2):
```

```
      stepEvolution(world, populationAgeGroups[1], wantVaccineAgeGroups[1],␣
↪vaccineRolloutRate, asympCells, sympCells)
    elif (ageGroup == 3):
      stepEvolution(world, populationAgeGroups[2], wantVaccineAgeGroups[2],␣
↪vaccineRolloutRate, asympCells, sympCells)
    elif (ageGroup == 4):
      stepEvolution(world, populationAgeGroups[3], wantVaccineAgeGroups[3],␣
↪vaccineRolloutRate, asympCells, sympCells)
    elif (ageGroup == 5):
      stepEvolution(world, populationAgeGroups[4], wantVaccineAgeGroups[4],␣
↪vaccineRolloutRate, asympCells, sympCells)

  createGrid(world)

interact(main,
         initialPercentage = (0, 100, 1),
         timeSteps = (0, 500, 1),
         ageGroup = [('Under 20', 1), ('20-39 years', 2), ('40-59 years', 3),␣
↪('60-79 years', 4), ('80+ years', 5)])
```



4 Cellular Automata simulations prioritizing each age group

This includes runs of the scenarios of the Cellular Automata model for each age group.

Setting up additional model parameters:

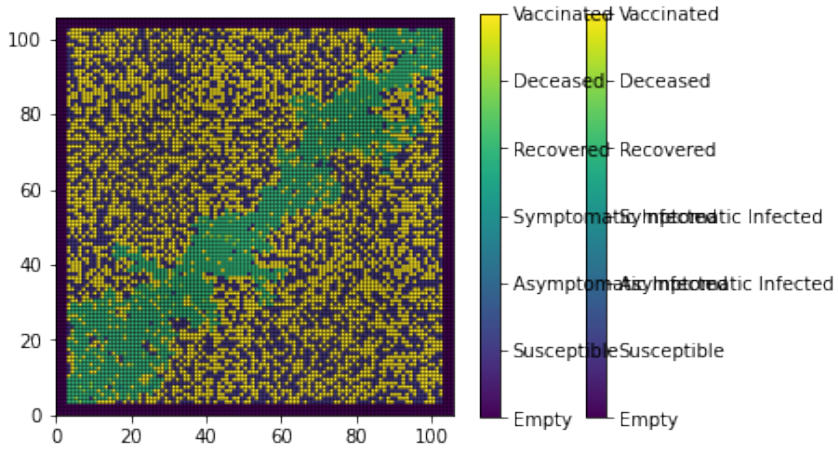
- **initialPercentage:** This is the percentage of the population that is initially infected. This value was chosen as 6%, considering the most up to date data for the US at the time the first vaccines were administered.
- **timeSteps:** The simulations will be run for a period of 300 days.

[ ]:
```
initialPercentage = 6
timeSteps = 300
```

Simulation run for *ageGroup* = **Under 20**. This prioritizes the *Under 20* age group for the vaccine distribution.

[ ]:
```
susceptibleList1 = []
asympInfectedList1 = []
sympInfectedList1 = []
recoveredList1 = []
deceasedList1 = []
vaccinatedList1 = []


def main(initialPercentage, timeSteps):
  world, asympCells, sympCells = createWorld(n, int(initialPercentage))

  createGrid(world)

  for i in range(timeSteps):
    suscep = susceptible(world)[1]
    susceptibleList1.append(suscep/n)

    asympI = asympInfected(world)[1]
    asympInfectedList1.append(asympI/n)

    sympI = sympInfected(world)[1]
    sympInfectedList1.append(sympI/n)

    recover = recovered(world)[1]
    recoveredList1.append(recover/n)

    dead = deceased(world)[1]
    deceasedList1.append(dead/n)

    vacc = vaccinated(world)[1]
    vaccinatedList1.append(vacc/n)
```

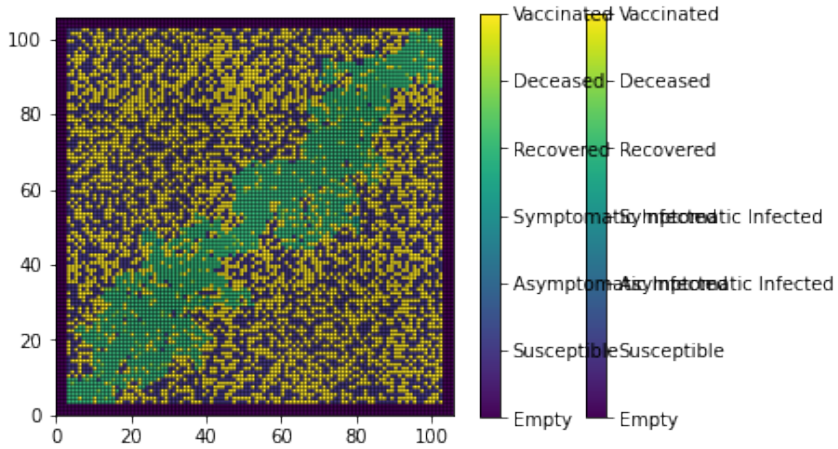
```
    stepEvolution(world, populationAgeGroups[0], wantVaccineAgeGroups[0],␣
 ↪vaccineRolloutRate, asympCells, sympCells)

  createGrid(world)
main(initialPercentage, timeSteps)

tot_inf1 = [sum(x) for x in zip(asympInfectedList1, sympInfectedList1)]
```



Simulation run for *ageGroup* = **20-39 years**. This prioritizes the *20-39 years* age group for the vaccine distribution.

[ ]:
```
susceptibleList2 = []
asympInfectedList2 = []
sympInfectedList2 = []
recoveredList2 = []
deceasedList2 = []
vaccinatedList2 = []


def main(initialPercentage, timeSteps):
  world, asympCells, sympCells = createWorld(n, int(initialPercentage))

  createGrid(world)

  for i in range(timeSteps):
    suscep = susceptible(world)[1]
    susceptibleList2.append(suscep/n)

    asympI = asympInfected(world)[1]
    asympInfectedList2.append(asympI/n)

    sympI = sympInfected(world)[1]
```

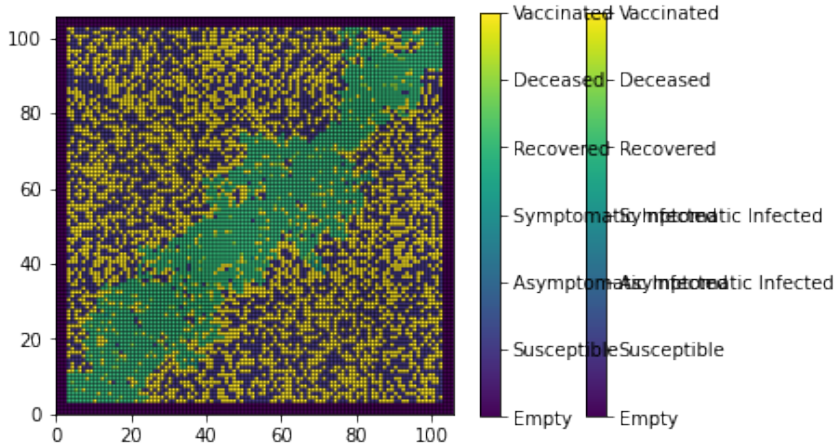
```
    sympInfectedList2.append(sympI/n)

    recover = recovered(world)[1]
    recoveredList2.append(recover/n)

    dead = deceased(world)[1]
    deceasedList2.append(dead/n)

    vacc = vaccinated(world)[1]
    vaccinatedList2.append(vacc/n)

    stepEvolution(world, populationAgeGroups[1], wantVaccineAgeGroups[1],␣
 ↪vaccineRolloutRate, asympCells, sympCells)

  createGrid(world)
main(initialPercentage, timeSteps)

tot_inf2 = [sum(x) for x in zip(asympInfectedList2, sympInfectedList2)]
```



Simulation run for *ageGroup* = **40-59 years**. This prioritizes the *40-59 years* age group for the vaccine distribution.

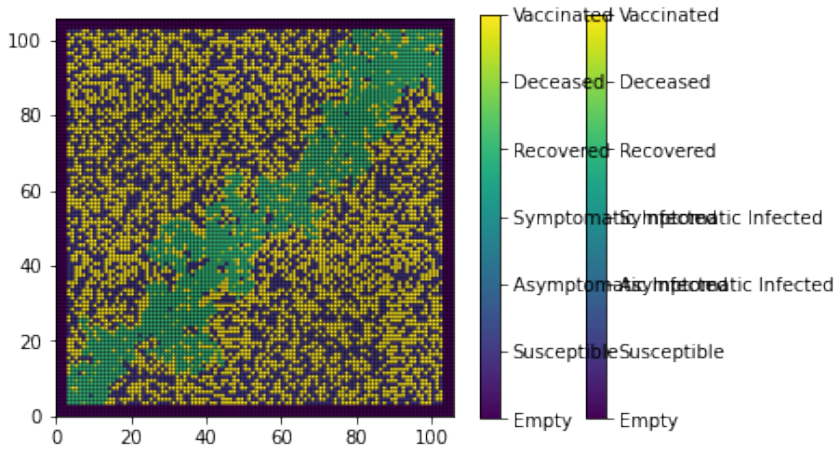
[ ]:
```
susceptibleList3 = []
asympInfectedList3 = []
sympInfectedList3 = []
recoveredList3 = []
deceasedList3 = []
vaccinatedList3 = []


def main(initialPercentage, timeSteps):
  world, asympCells, sympCells = createWorld(n, int(initialPercentage))
```

```
  createGrid(world)

  for i in range(timeSteps):
    suscep = susceptible(world)[1]
    susceptibleList3.append(suscep/n)

    asympI = asympInfected(world)[1]
    asympInfectedList3.append(asympI/n)

    sympI = sympInfected(world)[1]
    sympInfectedList3.append(sympI/n)

    recover = recovered(world)[1]
    recoveredList3.append(recover/n)

    dead = deceased(world)[1]
    deceasedList3.append(dead/n)

    vacc = vaccinated(world)[1]
    vaccinatedList3.append(vacc/n)

    stepEvolution(world, populationAgeGroups[2], wantVaccineAgeGroups[2],␣
 ↪vaccineRolloutRate, asympCells, sympCells)

  createGrid(world)
main(initialPercentage, timeSteps)

tot_inf3 = [sum(x) for x in zip(asympInfectedList3, sympInfectedList3)]
```



Simulation run for *ageGroup* = **60-79 years**. This prioritizes the *60-79 years* age group for the vaccine distribution.

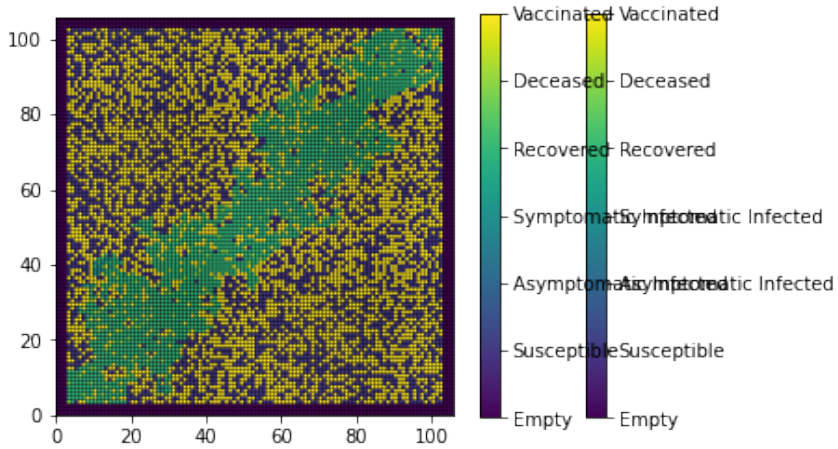
[ ]:
```
susceptibleList4 = []
asympInfectedList4 = []
sympInfectedList4 = []
recoveredList4 = []
deceasedList4 = []
vaccinatedList4 = []


def main(initialPercentage, timeSteps):
  world, asympCells, sympCells = createWorld(n, int(initialPercentage))

  createGrid(world)

  for i in range(timeSteps):
    suscep = susceptible(world)[1]
    susceptibleList4.append(suscep/n)

    asympI = asympInfected(world)[1]
    asympInfectedList4.append(asympI/n)

    sympI = sympInfected(world)[1]
    sympInfectedList4.append(sympI/n)

    recover = recovered(world)[1]
    recoveredList4.append(recover/n)

    dead = deceased(world)[1]
    deceasedList4.append(dead/n)

    vacc = vaccinated(world)[1]
    vaccinatedList4.append(vacc/n)

    stepEvolution(world, populationAgeGroups[3], wantVaccineAgeGroups[3],␣
 ↪vaccineRolloutRate, asympCells, sympCells)

  createGrid(world)
main(initialPercentage, timeSteps)

tot_inf4 = [sum(x) for x in zip(asympInfectedList4, sympInfectedList4)]
```



Simulation run for *ageGroup* = **80+ years**. This prioritizes the *80+ years* age group for the vaccine distribution.

[ ]:
```
susceptibleList5 = []
asympInfectedList5 = []
sympInfectedList5 = []
recoveredList5 = []
deceasedList5 = []
vaccinatedList5 = []


def main(initialPercentage, timeSteps):
  world, asympCells, sympCells = createWorld(n, int(initialPercentage))

  createGrid(world)

  for i in range(timeSteps):
    suscep = susceptible(world)[1]
    susceptibleList5.append(suscep/n)

    asympI = asympInfected(world)[1]
    asympInfectedList5.append(asympI/n)

    sympI = sympInfected(world)[1]
    sympInfectedList5.append(sympI/n)

    recover = recovered(world)[1]
    recoveredList5.append(recover/n)

    dead = deceased(world)[1]
    deceasedList5.append(dead/n)
```

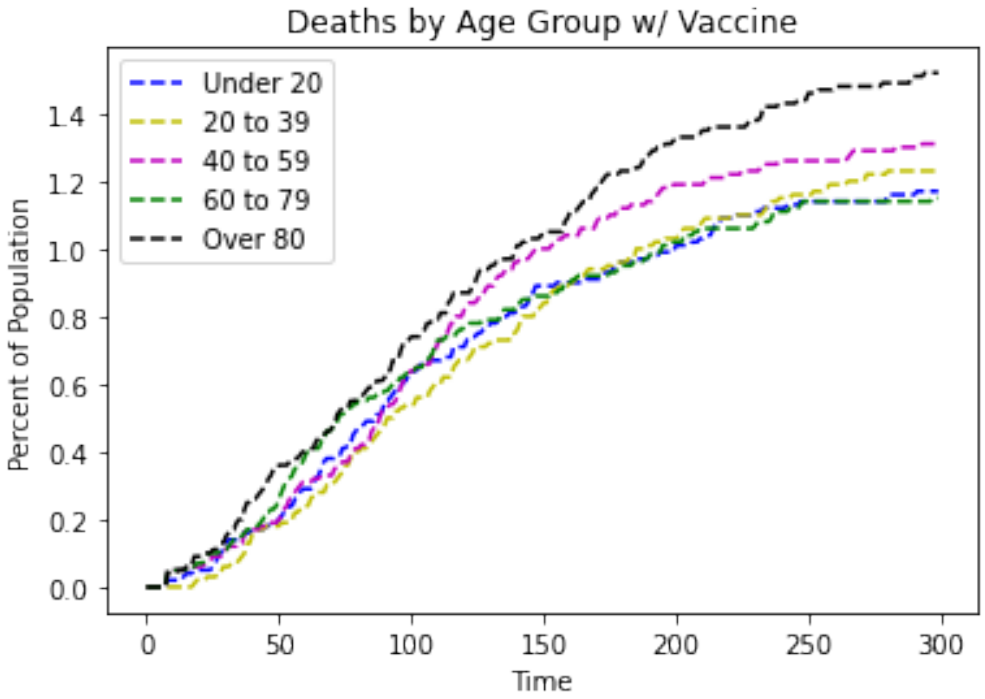
```
    vacc = vaccinated(world)[1]
    vaccinatedList5.append(vacc/n)

    stepEvolution(world, populationAgeGroups[4], wantVaccineAgeGroups[4],␣
 ↪vaccineRolloutRate, asympCells, sympCells)

  createGrid(world)
main(initialPercentage, timeSteps)

tot_inf5 = [sum(x) for x in zip(asympInfectedList5, sympInfectedList5)]
```

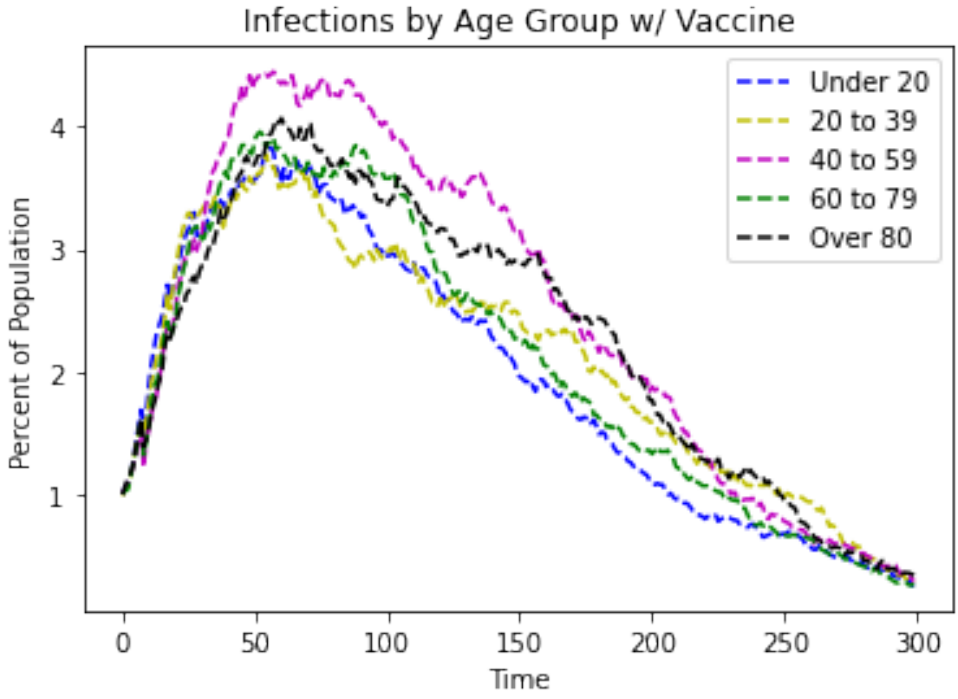


Plot of the **Deaths by Age Group** (cumulative deaths) and **Infections by Age Group** (daily infections) by prioritizing a single age group for the vaccine in each scenario:

[ ]:
```
plt.plot(deceasedList1,'b--',label='Under 20')
plt.plot(deceasedList2,'y--',label='20 to 39')
plt.plot(deceasedList3,'m--',label='40 to 59')
plt.plot(deceasedList4,'g--',label='60 to 79')
plt.plot(deceasedList5,'k--',label='Over 80')
plt.xlabel('Time')
plt.ylabel('Percent of Population')
plt.legend()
plt.title("Deaths by Age Group w/ Vaccine")
plt.show()

plt.plot(tot_inf1,'b--',label='Under 20')
plt.plot(tot_inf2,'y--',label='20 to 39')
plt.plot(tot_inf3,'m--',label='40 to 59')
plt.plot(tot_inf4,'g--',label='60 to 79')
plt.plot(tot_inf5,'k--',label='Over 80')
plt.xlabel('Time')
plt.ylabel('Percent of Population')
plt.legend()
plt.title("Infections by Age Group w/ Vaccine")
```

```
plt.show()
```


Deaths by Age Group w/ Vaccine
Under 20
20 to 39
40 to 59
60 to 79
Over 80
Percent of Population
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
50
100
150
200
250
300
Time


Infections by Age Group w/ Vaccine
Under 20
20 to 39
40 to 59
60 to 79
Over 80
Percent of Population
1
2
3
4
0
50
100
150
200
250
300
Time

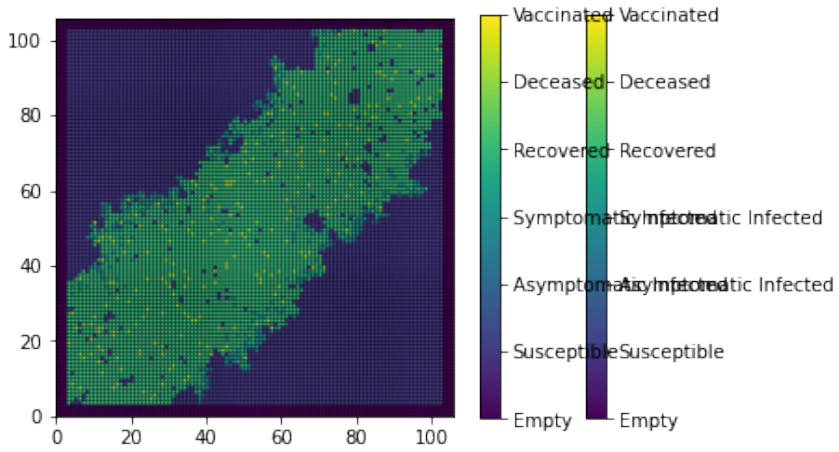
5 Cellular Automata simulation of COVID-19 spread without vaccine distribution

This simulation is used as a baseline for comparing results of the effectiveness of prioritizing each of the age groups.

The *stepEvolution* function is simply edited to remove the *vaccinate cells* step, and instead only account for the asymptomatic and symptomatic spread across cells using the same model parameters as in previous simulations.

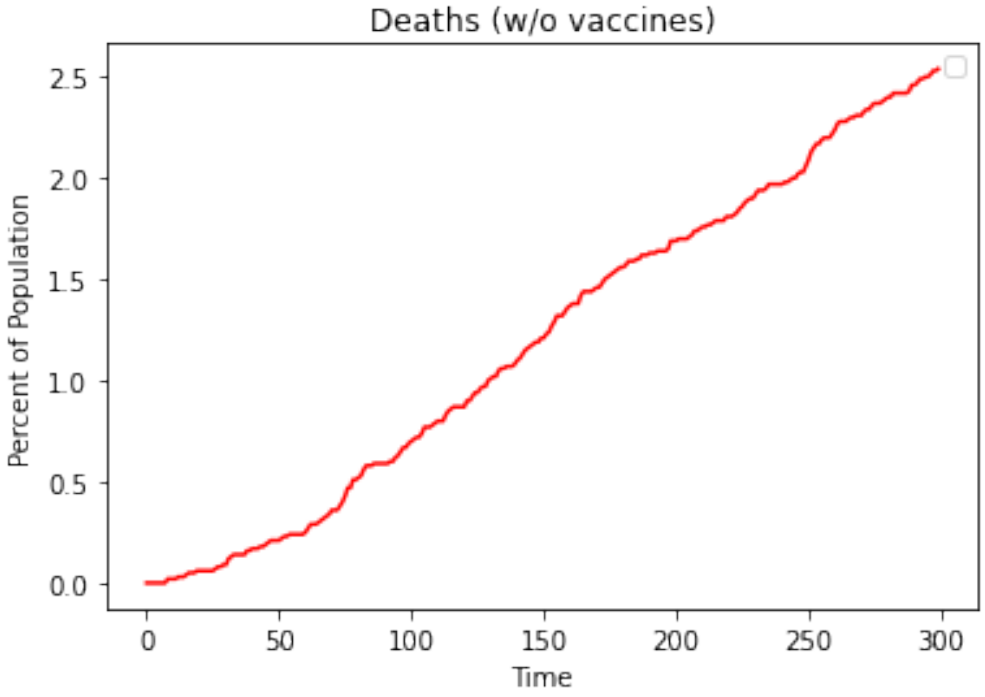
[ ]:
```
def stepEvolution(grid, ageGroupPopulation, ageGroupwantVaccine,␣
↪vaccineRolloutRate, asympCells, sympCells):
  updateAsympInfected(grid, asympCells)
  updateSympInfected(grid, sympCells, recoveryRate, deathRate)

  infectionProbability = infectionSpreadProbability(grid, asympRate, sympRate)

  for (x,y) in zip(*np.where(infectionProbability > 0)):
    number = random.random()

    if (number < infectionProbability[x,y]):
      grid[x,y] = ASYMP_INFECTED


susceptibleList11 = []
asympInfectedList11 = []
sympInfectedList11 = []
recoveredList11 = []
deceasedList11 = []
vaccinatedList11 = []


def main(initialPercentage, timeSteps):
  world, asympCells, sympCells = createWorld(n, int(initialPercentage))

  createGrid(world)

  for i in range(timeSteps):
    suscep = susceptible(world)[1]
    susceptibleList11.append(suscep/n)

    asympI = asympInfected(world)[1]
    asympInfectedList11.append(asympI/n)

    sympI = sympInfected(world)[1]
    sympInfectedList11.append(sympI/n)

    recover = recovered(world)[1]
```

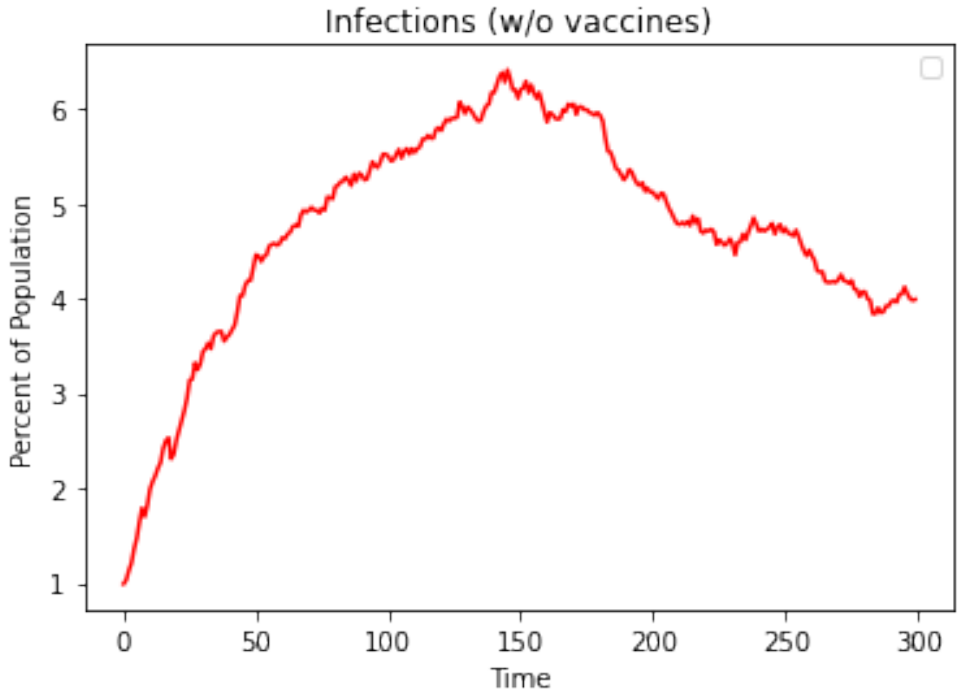
```
    recoveredList11.append(recover/n)

    dead = deceased(world)[1]
    deceasedList11.append(dead/n)

    vacc = vaccinated(world)[1]
    vaccinatedList11.append(vacc/n)

    stepEvolution(world, populationAgeGroups[0], wantVaccineAgeGroups[0],␣
 ↪vaccineRolloutRate, asympCells, sympCells)

  createGrid(world)
main(initialPercentage, timeSteps)

tot_inf11 = [sum(x) for x in zip(asympInfectedList11, sympInfectedList11)]
```



Plot showing the results of COVID-19 spread without any vaccination distributions:

[ ]:
```
plt.plot(deceasedList11,'r')
plt.xlabel('Time')
plt.ylabel('Percent of Population')
plt.legend()
plt.title("Deaths (w/o vaccines)")
plt.show()

plt.plot(tot_inf11,'r')
plt.xlabel('Time')
plt.ylabel('Percent of Population')
plt.legend()
plt.title("Infections (w/o vaccines)")
plt.show()
```

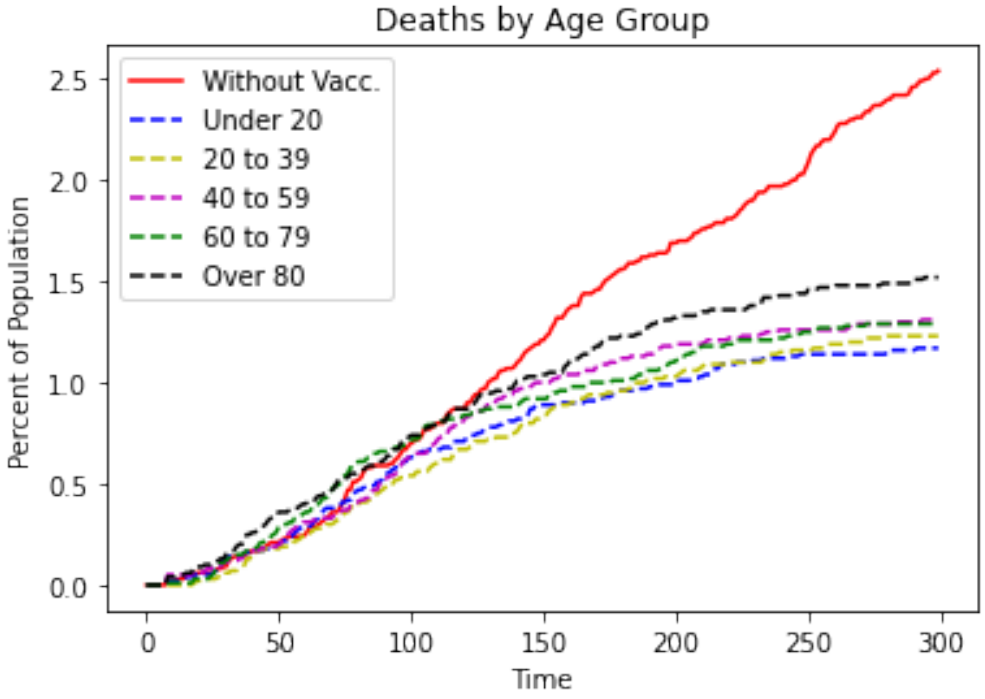

Deaths (w/o vaccines)
Percent of Population
Time
0.0
0.5
1.0
1.5
2.0
2.5
0
50
100
150
200
250
300

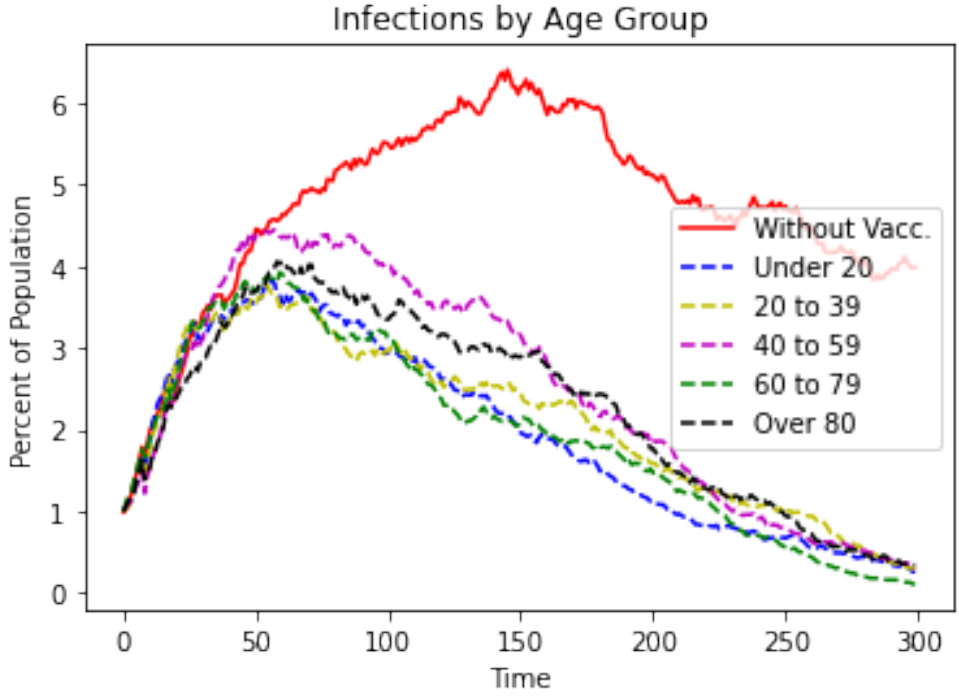

Infections (w/o vaccines)
Percent of Population
Time
1
2
3
4
5
6
0
50
100
150
200
250
300

6 Evaluation of Results

Plot of the **Deaths by Age Group** (cumulative deaths) and **Infections by Age Group** (daily infections) by prioritizing a single age group for the vaccine in each scenario, side-by-side with the data of simulating just the COVID-19 spread with no vaccination distribution:

[ ]:
```
plt.plot(deceasedList11,'r',label='Without Vacc.')
plt.plot(deceasedList1,'b--',label='Under 20')
plt.plot(deceasedList2,'y--',label='20 to 39')
plt.plot(deceasedList3,'m--',label='40 to 59')
plt.plot(deceasedList4,'g--',label='60 to 79')
plt.plot(deceasedList5,'k--',label='Over 80')
plt.xlabel('Time')
plt.ylabel('Percent of Population')
plt.legend()
plt.title("Deaths by Age Group")
plt.show()

plt.plot(tot_inf11,'r',label='Without Vacc.')
plt.plot(tot_inf1,'b--',label='Under 20')
plt.plot(tot_inf2,'y--',label='20 to 39')
plt.plot(tot_inf3,'m--',label='40 to 59')
plt.plot(tot_inf4,'g--',label='60 to 79')
plt.plot(tot_inf5,'k--',label='Over 80')
plt.xlabel('Time')
plt.ylabel('Percent of Population')
plt.legend()
plt.title("Infections by Age Group")
plt.show()
```


Deaths by Age Group
Without Vacc.
Under 20
20 to 39
40 to 59
60 to 79
Over 80
Percent of Population
Time
0.0
0.5
1.0
1.5
2.0
2.5
0
50
100
150
200
250
300


Infections by Age Group
Without Vacc.
Under 20
20 to 39
40 to 59
60 to 79
Over 80
Percent of Population
Time
0
1
2
3
4
5
6
0
50
100
150
200
250
300

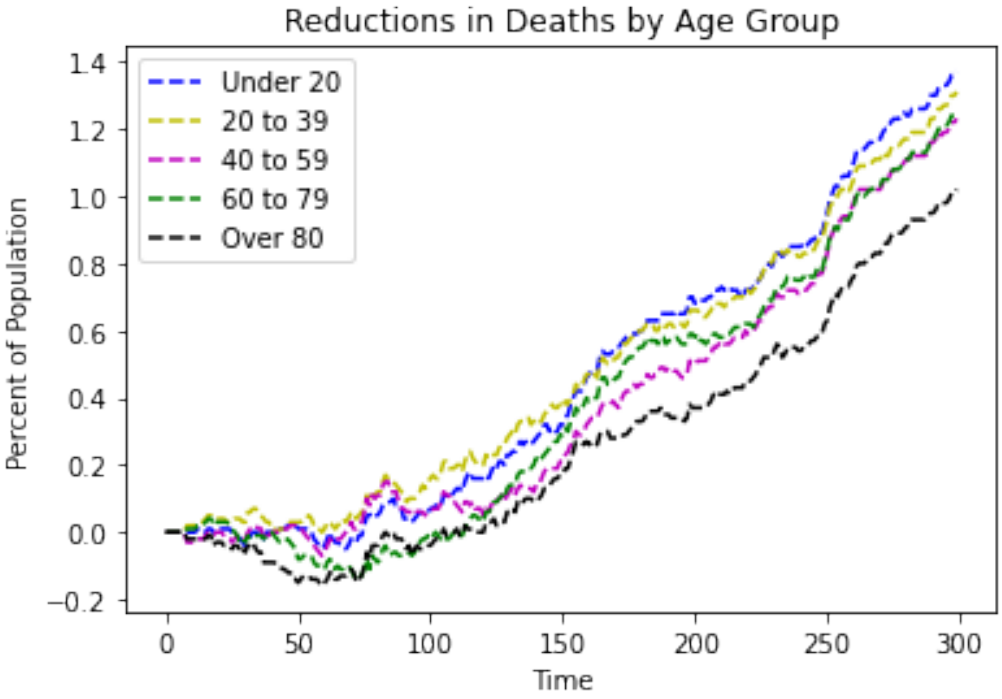
The following are observations from the above graphs:

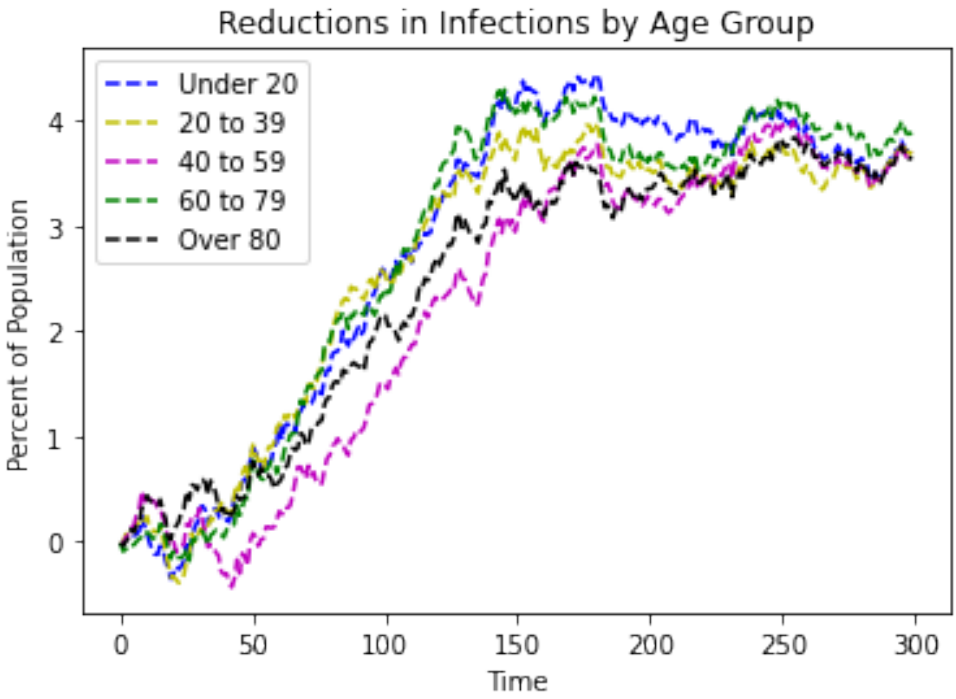
1. The simulation with no vaccination produces much higher deaths and infections in the population. This is accurate because the introduction of vaccines lowers the spread and infection rate of the virus, hence lowering the percentages of deaths and infections in the population.
2. The distribution of vaccines "flattens the curve". In the deaths graph, the deaths as a percentage of the population start to flatten out at approx. day 200, where after it is only slowly increasing. In comparison, the simulation with no vaccine distribution is increasing at a high continuous rate throughout the simulation.
3. The daily infections decrease at a faster rate with the distribution of vaccines. In the infections graph, the infections as a percentage of the population is greatly decreased after a quantifiable amount of people receive the vaccine. It can be seen that by day 300, the infection rate is far below 1% per day. In comparison, the simulation with no vaccine keeps a steady 4-5% daily infections, with no huge decrease in cases.

Plot of the **Reductions in Deaths by Age Group** (cumulative deaths) and **Reductions in Infections by Age Group** (daily infections) by prioritizing a single age group for the vaccine in each scenario. To calculate the reductions in both deaths and infections, we used the results from the simulation of the COVID-19 spread without vaccine distribution, and took the difference between the baseline and the results gathered from prioritizing each of the age groups:

[ ]:
```
plt.plot([a_i - b_i for a_i, b_i in zip(deceasedList11,␣
 ↪deceasedList1)],'b--',label='Under 20')
plt.plot([a_i - b_i for a_i, b_i in zip(deceasedList11,␣
 ↪deceasedList2)],'y--',label='20 to 39')
plt.plot([a_i - b_i for a_i, b_i in zip(deceasedList11,␣
 ↪deceasedList3)],'m--',label='40 to 59')
plt.plot([a_i - b_i for a_i, b_i in zip(deceasedList11,␣
 ↪deceasedList4)],'g--',label='60 to 79')
plt.plot([a_i - b_i for a_i, b_i in zip(deceasedList11,␣
 ↪deceasedList5)],'k--',label='Over 80')
plt.xlabel('Time')
plt.ylabel('Percent of Population')
plt.legend()
plt.title("Reductions in Deaths by Age Group")
plt.show()

plt.plot([a_i - b_i for a_i, b_i in zip(tot_inf11, tot_inf1)],'b--',label='Under␣
 ↪20')
plt.plot([a_i - b_i for a_i, b_i in zip(tot_inf11, tot_inf2)],'y--',label='20 to␣
 ↪39')
plt.plot([a_i - b_i for a_i, b_i in zip(tot_inf11, tot_inf3)],'m--',label='40 to␣
 ↪59')
plt.plot([a_i - b_i for a_i, b_i in zip(tot_inf11, tot_inf4)],'g--',label='60 to␣
 ↪79')
plt.plot([a_i - b_i for a_i, b_i in zip(tot_inf11, tot_inf5)],'k--',label='Over␣
 ↪80')
plt.xlabel('Time')
```

```
plt.ylabel('Percent of Population')
plt.legend()
plt.title("Reductions in Infections by Age Group")
plt.show()
```





This is the result of our comprehensive simulation using the Cellular Automata model. The above graphs show the reductions in deaths and infections by prioritizing each age group for the vaccinations, where the higher the line in the graph, the larger the benefit from prioritizing that age group.

Reductions in Deaths

Highest Reduction: Prioritizing the *Under 20* age group
Lowest Reduction: Prioritizing the *Over 80* age group

Reductions in Infections

Highest Reduction: Prioritizing the *Under 20* age group
Lowest Reduction: Prioritizing the *40-59* age group

The distribution of vaccines produced very good reductions in deaths and infections, with the prioritization of each age group producing a slightly better or worse reduction. The reduction results for each of the age groups were very close.

To **minimize the *deaths*** from the COVID-19 virus, it should be preferred to prioritize the ***Under 20* age group**. Conversly, if the *Over 80* age group was prioritized, the reductions in deaths would drop to only approx. 1% as compared to a greater approx. 1.4%.

To **minimize the *infections*** from the COVID-19 virus, it should be preferred to prioritize either the ***Under 20* or *60-79 years* age groups**. This is because prioritizing each of those age groups produced comparable results of around 4-5% reductions with very little variations between both. Conversly, the *40-59 years* age group should not be prioritized because it reduces the reductions to only around 3.5% at its peak.

These age group prioritizations are intuitive because the Under 20 age group has a large population density, therefore prioritizing them for the vaccine would inevitably reduce the amount of infections and deaths in the population. Moreover, the 40-59 years age group is the least accepting of the vaccine, hence it can be inferred that if they are prioritized then the number of infections will still be high.

Vaccine Prioritizations using SEIR model

1 Introduction

The SEIR model is a variation of the basic SIR model that incorporates an extra "exposed" state for individuals. The SEIR model is used for diseases that have an incubation period which the exposed state represents, which is valid for COVID-19. We'll be using the extended SEIR model, which extends the classic model to represent asymptomatic and symptomatic states, which have particular relevance to the COVID-19 pandemic.

The goal of our study is to implement the susceptible-exposed-infectious-recovered (SEIR) dynamics on a stochastic dynamical network that is age-stratified, and provide users with data-driven results for vaccine prioritizations within our population. The SEIR model is a dynamical system that is represented by a series of differential equations, describing the evolution of the system in terms of how many people are at each state.

After we set up our models and understand the inputs, we'll first show a simple deterministic variation of the SEIR model, so we can see how the model behaves with a given set of fixed arguments. In addition, we'll add external factors such as social distancing at fixed timestamps to see how we can utilize the different extensions that the SEIRS+ Model Framework offers for representing the real-world. Finally, we'll create our stochastic dynamical network where each node is age-stratified and represents an individual. We can then apply the SEIR model on this network along with varying vaccine distributions to simulate how it may impact our population of interest.

2 Model Setup / Definitions

We'll import the SEIRS+ Model Framework which implements models of generalized SEIRS infectious disease dynamics, with extensions that will allow us to study the effect of vaccination prioritizations on different age groups.

[ ]:
```
!pip install seirsplus
from seirsplus.models import *
```

```
|████████████████|
|████████████████|
|████████████████|
|████████████████████|
```

```
Successfully built seirsplus
```

```
Installing collected packages: seirsplus
Successfully installed seirsplus-1.0.9
```

All model parameter values are set in the call to the SEIRSModel constructor. The following parameters are defined with their data type and default value:

- **beta:** rate of transmission, *float*, REQUIRED
- **sigma:** rate of progression, *float*, REQUIRED
- **gamma:** rate of recovery, *float*, REQUIRED
- **xi:** rate of re-susceptibility, *float*, 0
- **mu_I:** rate of infection-related mortality, *float*, 0
- **mu_0:** rate of baseline mortality, *float*, 0
- **nu:** rate of baseline birth, *float*, 0
- **beta_Q:** rate of transmission for detected cases, *float*, None (set equal to beta)
- **sigma_Q:** rate of progression for detected cases, *float*, None (set equal to sigma)
- **gamma_Q:** rate of recovery for detected cases, *float*, None (set equal to gamma)
- **mu_Q:** rate of infection-related mortality for detected cases, *float*, None (set equal to mu_I)
- **theta_E:** rate of testing for exposed individuals, *float*, 0
- **theta_I:** rate of testing for infectious individuals, *float*, 0
- **psi_E:** probability of positive tests for exposed individuals, *float*, 0
- **psi_I:** probability of positive tests for infectious individuals, *float*, 0
- **initN:** initial total number of individuals, *int*, 10
- **initI:** initial number of infectious individuals, *int*, 10
- **initE:** initial number of exposed individuals, *int*, 0
- **initQ_E:** initial number of detected exposed individuals, *int*, 0
- **initQ_I:** initial number of detected infectious individuals, *int*, 0
- **initR:** initial number of recovered individuals, *int*, 0
- **initF:** initial number of deceased individuals, *int*, 0

The basic SEIR parameters are initN, beta, sigma, and gamma. An example state of the system where we have a Disease-Free-Equilibrium is (N,0,0,0), where N is our population. For our simulations, we will be using statistics from the United States.

3 Creating a basic deterministic SEIR model

Here, we will be building a simple deterministic model which demonstrates how we can manipulate simulation parameters to model potential scenarios. The system of differential equations that the model is built upon is simply solved numerically, excluding any external chance events of the system. This simple model will show the spread of COVID-19 before any vaccinations, and will be the foundation of our future simulations. In this example, we'll also see how we can introduce factors such as social distancing and how they impact our system.

3.1 Initializing model parameters

Using the parameters defined in the previous section, we can create a SEIRSModel with our desired values. The only required parameters are initN, beta, sigma, and gamma. For this example, all parameters are listed, so feel free to manipulate the variables and see how they change the output!

Using our pre-determined values for the United States, we get 0.147, 0.192, 0.077 for beta, sigma, and gamma, respectively. We also have 0.0015 for mu_I, the infection-related mortality. This model is completely dependent upon these values, and no other factors are included. To better see our results, we use a population initN of 1,000,000 which provides us with more granularity, but still represents our target population.

[ ]:
```
model = SEIRSModel(initN   =1000000,
                   beta    =0.147,
                   sigma   =0.192,
                   gamma   =.077,
                   mu_I    =0.0015,
                   mu_0    =0,
                   nu      =0,
                   xi      =0,
                   beta_Q  =0.147,
                   sigma_Q =.192,
                   gamma_Q =.077,
                   mu_Q    =0.0015,
                   theta_E =0,
                   theta_I =0,
                   psi_E   =1.0,
                   psi_I   =1.0,
                   initI   =10000,
                   initE   =0,
                   initQ_E =0,
                   initQ_I =0,
                   initR   =0,
                   initF   =0)
```

3.2 Running our simulation

Once we've initialized our SEIR model, we can run a simulation with a call to run(), which has the following arguments:

- **T:** runtime of simulation, *numeric*, REQUIRED
- **checkpoints:** dictionary of checkpoint lists (see section below), *dictionary*, None
- **print_interval:** (network model only) time interval to print sim status to console, *numeric*, 10
- **verbose:** if True, print count in each state at print intervals, else just the time, *bool*, False

[ ]:
```
model.run(T=300)
```

```
t = 299.90
True
```

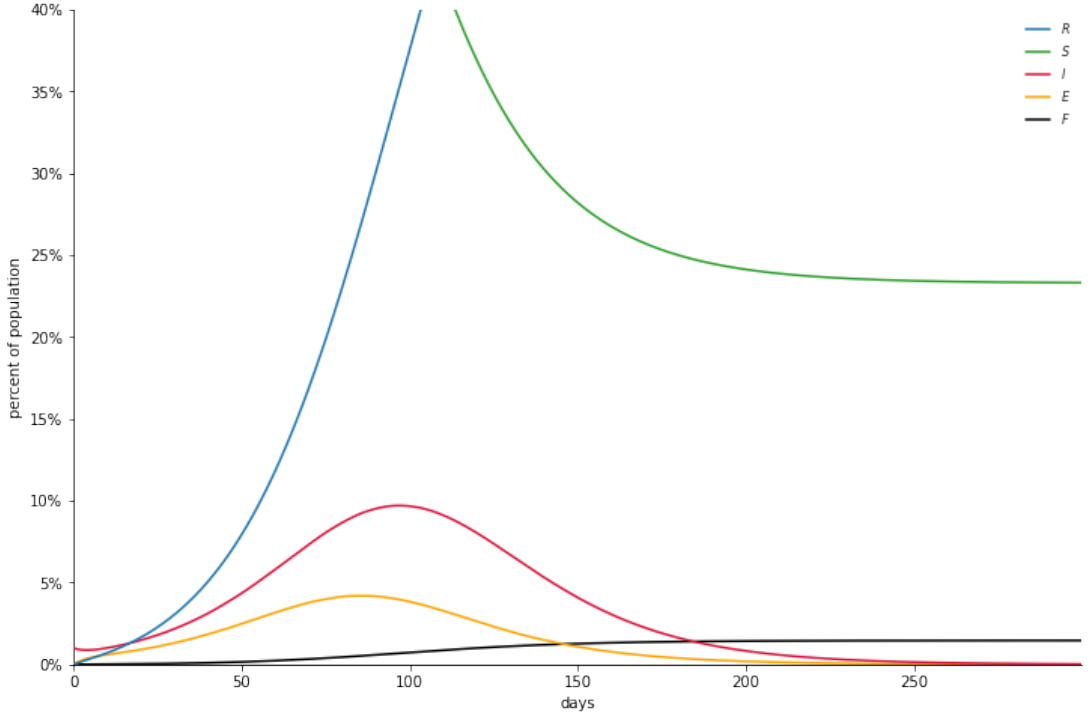
3.3 Visualizing the results

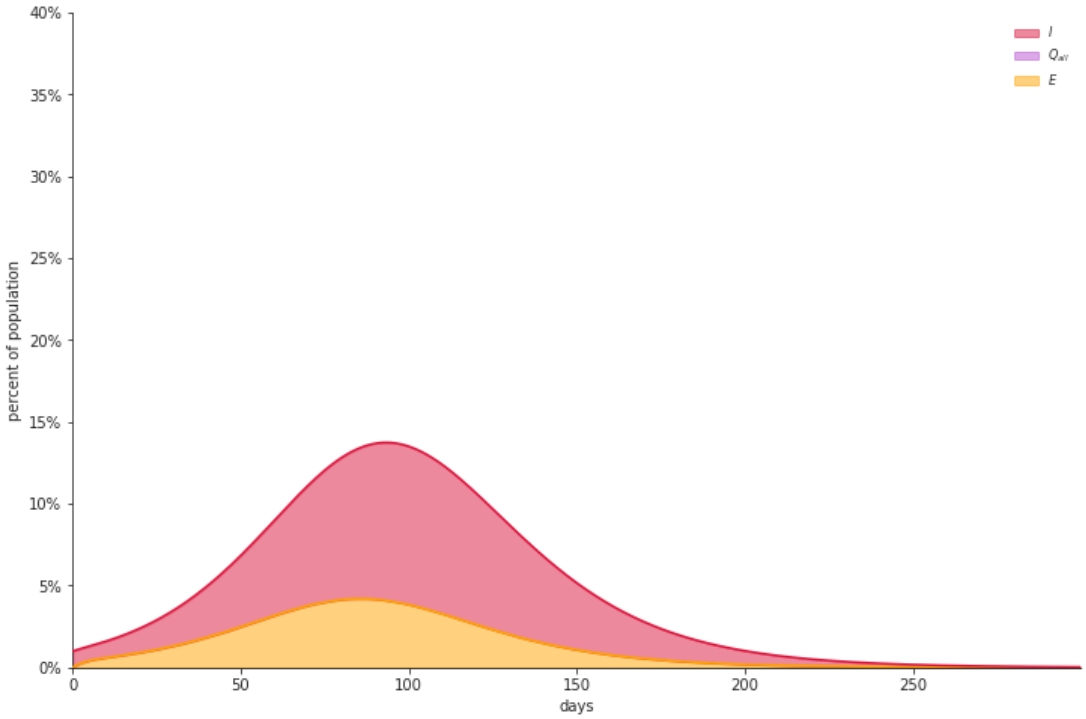
Now that we have everything setup, all we need to do is create a plot for our results. The SEIRSNetworkModel class provides functions that generate models of our simulation. Here, we

demonstrate the two functions:

- **figure_basic()** generates a line plot of the frequency of each state in our population
- **figure_infections()** generates a stacked area plot of the frequency of only the infection states (E, I, Q) in our population

[ ]:
```
model.figure_basic(ylim=0.4)
model.figure_infections(ylim=0.4)
```





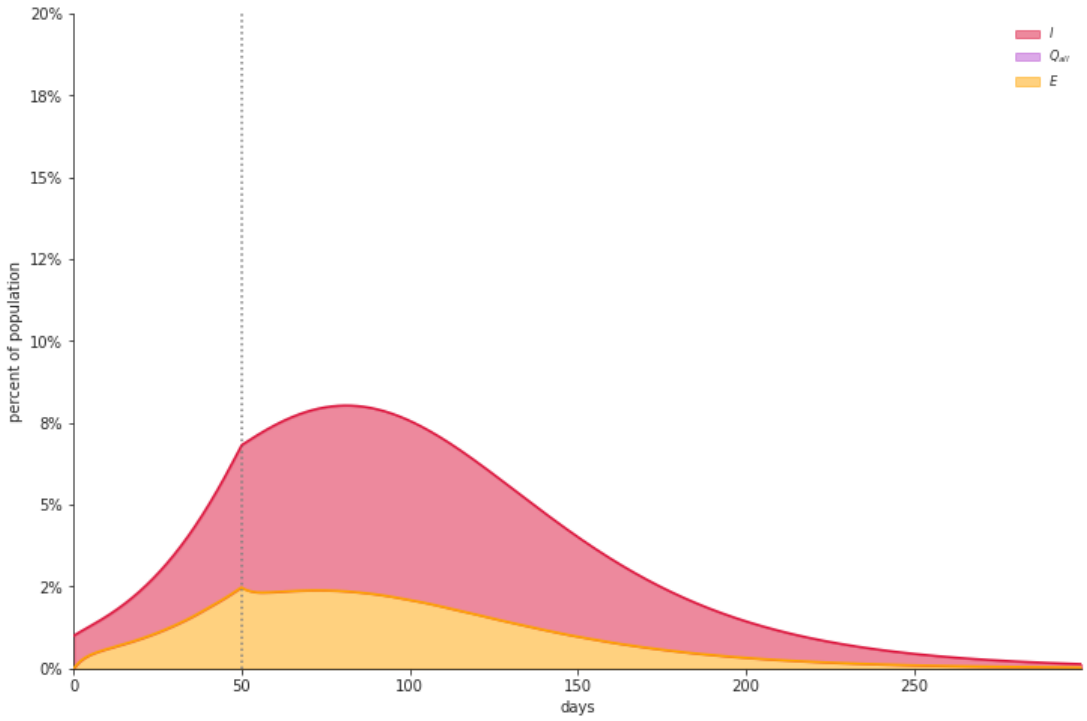
3.4 Implementing additional factors

Now that we have a basic understanding of how the SEIR model is used, we can add additional factors such as social distancing and view its effects. To see how additional factors can impact our model, we reduce the rate of transmission at t = 50, which can correspond to factors like social distancing (or vaccines) which reduce the spread of the virus.

[ ]:
```
model = SEIRSModel(initN   =1000000,
                   beta    =0.147,
                   sigma   =0.192,
                   gamma   =.077,
                   mu_I    =0.0015,
                   mu_0    =0,
                   nu      =0,
                   xi      =0,
                   beta_Q  =0.147,
                   sigma_Q =.192,
                   gamma_Q =.077,
                   mu_Q    =0.0015,
                   theta_E =0,
                   theta_I =0,
                   psi_E   =1.0,
                   psi_I   =1.0,
                   initI   =10000,
```

```
                   initE   =0,
                   initQ_E =0,
                   initQ_I =0,
                   initR   =0,
                   initF   =0)

checkpoints = {'t':       [50],
               'beta':    [0.11]
              }

model.run(T=300, checkpoints=checkpoints)

model.figure_infections(vlines=checkpoints['t'], ylim=0.2)
```

```
[Checkpoint: Updating parameters]
t = 49.90
```



From our resulting model, we can see that at t = 50, the increase of the number of infected dramatically stops increasing, and quickly reaches its peak. Compared to our first model, we can see that the maximum number of infected individuals in our population is much lower.

4 The Stochastical Model

We will now be creating the stochastical models for our study. In contrast to the deterministic model, our stochastical models will possess some inherent randomness.

When studying disease transmission, it's important to consider patterns of interactions which can be seen in real-world contact networks (e.g., heterogeneity, transitivity, network structure). The creation of a network model will allow us to analyze transmission patterns and network-based interventions with respect to properties of a realistic contact network. Strategies for mitigating spread can be thought of as perturbing the contact network (e.g., vaccinations, social distancing, testing).

In this section, we'll first be creating the network for our population where each node will be age-stratified and will represent an individual. Then, for each age group, we'll apply vaccinations and run our simulation. As a result, we'll see how varying prioritizations of age groups may impact the overall population.

4.1 Creating our network

There are multiple networks that we can use to represent our population, depending on what we want to show. For our study, we'll be using FARZ networks, which has built-in community structure which will allow us to represent the various age groups as communities within our population. For these communities, the FARZ algoritm offers parameters which will allow for randomness and transitivity among nodes, which will represent the virus and its spread among the different communities.

The following imports will give us the tools needed to create our network and run our simulation:

[ ]:
```
from seirsplus.models import *
from seirsplus.networks import *
from seirsplus.sim_loops import *
from seirsplus.utilities import *
```

In our network, our nodes (which represent people) are divided up into households, where the distribution of household size and age demographics are provided for the generation of our graph. The resulting values are randomized and change each time it's run to provide stochasticity for each simulation, but still remain close to our given distributions. The nodes in each household will be strongly connected.

To represent out-of-household contact between nodes, we create layers in our FARZ network. Each layer is age-stratisfied, and is used to represent contact among individuals of the same age group. We'll be using our pre-determined age groups of under 20 (communities can be thought of as school/college), 20 - 39 (communities can be thought of as workplaces), 40 - 59, 60-79, and 80+. Each of the nodes in our households will belong to a layer.

In addition to our network layer interactions, there will also be a probability of well-mixed global interactions, where random nodes interact with another node anywhere in the network (i.e. people that interact with others outside their age group). This is a default feature in the SEIRS+ network models.

4.2 Setting basic parameters

Here we set the population size and the initial population that is exposed. We use a graph size of 1000 to reduce simulation time, but networks with ~10,000 nodes better represent per-capita population dynamics. From our data, we set the intial prevalance to be 6% of the population.

[ ]:
```
N = 1000
INIT_EXPOSED = int(N*0.06)
```

4.3 Generating the network

Now we create the household and age-stratified layers of our network, and pass it into the network generation function to create our network.

Remember - this model is *stochastic*, so the number of nodes in each age group should slightly change from one execution to the next. The number of nodes in each age group is printed.

[ ]:
```
# we create the data for our households, using our initial US data for age␣
 ↪distribution,
# and data from the US census for household size
household_data = {
                  'age_distn':{'0-9': 0.12435,
                               '10-19': 0.12435,
                               '20-29': 0.136,
                               '30-39': 0.136,
                               '40-49': 0.12595,
                               '50-59': 0.12595,
                               '60-69': 0.094,
                               '70-79': 0.094,
                               '80+'  : 0.0394
                              },
                  'household_size_distn':{ 1: 0.284,
                                           2: 0.345,
                                           3: 0.151,
                                           4: 0.128,
                                           5: 0.058,
                                           6: 0.023,
                                           7: 0.011
                                           },
                  'household_stats':{ 'pct_with_under20': 0.337,                 ␣
↪      # percent of households with at least one member under 60
                                      'pct_with_over60': 0.380,                  ␣
↪      # percent of households with at least one member over 60
                                      'pct_with_under20_over60':  0.034,         ␣
↪      # percent of households with at least one member under 20 and at least␣
↪one member over 60
                                      'pct_with_over60_givenSingleOccupant': 0.
↪110,   # percent of households with a single-occupant that is over 60
```

```
                                        'mean_num_under20_givenAtLeastOneUnder20':
 ↪ 1.91 # number of people under 20 in households with at least one member under␣
 ↪20
                                    }
                }

# we create our layers, which are age-stratified and used to model
# out-of-household interactions between communities
layer_info  = {
                'Under 20': {'ageBrackets': ['0-9', '10-19'], 'meanDegree': 13.
 ↪2, 'meanDegree_CI': (0, 19.8) },
                '20-39': {'ageBrackets': ['20-29', '30-39'], 'meanDegree': 16.2,␣
 ↪'meanDegree_CI': (12.5, 19.8) },
                '40-59': {'ageBrackets': ['40-49', '50-59'], 'meanDegree': 14.2,␣
 ↪'meanDegree_CI': (12.5, 19.8) },
                '60+': {'ageBrackets': ['60-69', '70-79', '80+'], 'meanDegree':␣
 ↪13.9, 'meanDegree_CI': (7.3, 20.5) }
              }

# finally we create our graph with our pre-defined dicts; household_data
# and layer_info
demographic_graph, ageGroups, households =␣
 ↪generate_demographic_contact_network(N, household_data, 'FARZ', layer_info,␣
 ↪distancing_scales=[0.7], isolation_groups=[])

G_baseline = demographic_graph['baseline']
G_quarantine = demographic_graph['distancingScale0.7']
households_indices = [household['indices'] for household in households]

# print out the number of nodes in each age group to better see our distribution
nodesU20 = []
nodesU39O20 = []
nodesU59O40 = []
nodesU79O60 = []
nodesO80 = []
nodeNum = 0
for ageGroup in ageGroups:
  if ageGroup == '0-9' or ageGroup == '10-19':
    nodesU20.append(nodeNum)
  elif ageGroup == '20-29' or ageGroup == '30-39':
    nodesU39O20.append(nodeNum)
  elif ageGroup == '40-49' or ageGroup == '50-59':
    nodesU59O40.append(nodeNum)
  elif ageGroup == '60-69' or ageGroup == '70-79':
    nodesU79O60.append(nodeNum)
  elif ageGroup == '80+':
```

```
    nodes080.append(nodeNum)
  nodeNum += 1

print("\nNum nodes Under 20:", len(nodesU20))
print("Num nodes 20-39:", len(nodesU3920))
print("Num nodes 40-59:", len(nodesU5940))
print("Num nodes 60-79:", len(nodesU7960))
print("Num nodes 80+:", len(nodes080), "\n")
```

```
Generated overall age distribution:
0-9: 0.1250     (0.0006 from target)
10-19: 0.1300   (0.0057 from target)
20-29: 0.1190   (-0.0170 from target)
30-39: 0.1470   (0.0110 from target)
40-49: 0.1250   (-0.0010 from target)
50-59: 0.1200   (-0.0060 from target)
60-69: 0.0850   (-0.0090 from target)
70-79: 0.1030   (0.0090 from target)
80+: 0.0460     (0.0066 from target)

Generated household size distribution:
1: 0.2680       (-0.0160 from target)
2: 0.3226       (-0.0224 from target)
3: 0.1737       (0.0227 from target)
4: 0.1538       (0.0258 from target)
5: 0.0645       (0.0065 from target)
6: 0.0124       (-0.0106 from target)
7: 0.0050       (-0.0060 from target)
Num households: 403
mean household size: 2.444

Generating graph for Under 20...
Generating graph for 20-39...
Generating graph for 40-59...
Generating graph for 60+...

Num nodes Under 20: 255
Num nodes 20-39: 266
Num nodes 40-59: 245
Num nodes 60-79: 188
Num nodes 80+: 46
```

4.4 Setting model parameters

Parameter values for our model are assigned to members of the population on an individual basis. Each node will have a differing rate of transmission, progression, recovery, etc. We create a distribution of values for each of our parameters that will have a mean of our initial data.

[ ]:
```
# network_info(G_baseline, "Baseline", plot=True)
latentPeriod_mean, latentPeriod_coeffvar = 3.0, 0.6
SIGMA   = 1 / gamma_dist(latentPeriod_mean, latentPeriod_coeffvar, N)

presymptomaticPeriod_mean, presymptomaticPeriod_coeffvar = 2.2, 0.5
LAMDA   = 1 / gamma_dist(presymptomaticPeriod_mean,␣
 ↪presymptomaticPeriod_coeffvar, N)

symptomaticPeriod_mean, symptomaticPeriod_coeffvar = 3.0, 0.4
GAMMA   = 1 / gamma_dist(symptomaticPeriod_mean, symptomaticPeriod_coeffvar, N)

infectiousPeriod = 1/LAMDA + 1/GAMMA

onsetToHospitalizationPeriod_mean, onsetToHospitalizationPeriod_coeffvar = 11.0,␣
 ↪0.45
ETA     = 1 / gamma_dist(onsetToHospitalizationPeriod_mean,␣
 ↪onsetToHospitalizationPeriod_coeffvar, N)

hospitalizationToDischargePeriod_mean, hospitalizationToDischargePeriod_coeffvar␣
 ↪= 11.0, 0.45
GAMMA_H = 1 / gamma_dist(hospitalizationToDischargePeriod_mean,␣
 ↪hospitalizationToDischargePeriod_coeffvar, N)

hospitalizationToDeathPeriod_mean, hospitalizationToDeathPeriod_coeffvar = 7.0,␣
 ↪0.45
MU_H    = 1 / gamma_dist(hospitalizationToDeathPeriod_mean,␣
 ↪hospitalizationToDeathPeriod_coeffvar, N)

PCT_ASYMPTOMATIC = 0.25
PCT_ASYMPTOMATIC = [0.8 if age in ['0-9', '10-19'] else PCT_ASYMPTOMATIC for age␣
 ↪in ageGroups]

R0_mean     = 2.5
R0_coeffvar = 0.2
R0 = gamma_dist(R0_mean, R0_coeffvar, N)
BETA = 1/infectiousPeriod * R0
BETA_Q = BETA * (0.3/R0_mean)
BETA_PAIRWISE_MODE  = 'infected'
DELTA_PAIRWISE_MODE = 'mean'
ALPHA = [0.5 if age in ['0-9', '10-19'] else 1.0 for age in ageGroups]
P_GLOBALINTXN = 0.2
Q_GLOBALINTXN = 0.05
```

4.5 Administering Vaccines

While the SEIRS+ Model framework doesn't provide a specific option for vaccinations, we're able to emulate the effect through the manipulation of relevant variables. When an individual in the

population is vaccinated, they are still able to interact with others and spread the virus.

In the following simulation runs, we'll be manipulating the hospilization and resulting fatality rates for each age group to achieve this characteristic. By changing these variables for each of our age groups, any node in our network is still able to interact with other nodes and transmit/contract the virus, except they are highly unlikely to be hospilized and die as a result. When an age group in our network is vaccinated, we'll be setting the hospitilization and resulting fatality rate to 0%.

For hospilization and resulting fatality rates of our population, we'll be using data from the following: https://covid.cdc.gov/covid-data-tracker/#demographicsovertime

In addition, remember that this model is *stochastic*, meaning that you should expect the resulting values to change after each run. While you may run into outliers, running the simulation multiple times will allow you to see the typical outputs. Further, setting the N value to 10,000 will result in more stable, accurate, and representative results. To quickly run our simulations, we use an N value of 1000, but feel free to change it to whatever you like (keep in mind using a very large N value will result in very long simulation times).

Each of our simulations are independent of each other, and are split up for better viewing. In our study, we're focused more on the fatality rates so that's what we'll be plotting, but the infection rates are printed as well. To further simplify the code, factors such as testing and social distancing are not applied, so you may observe a high percentage of infection rates. While this may not depict what is happening in the US population, the trend that the model represents is still accurate.

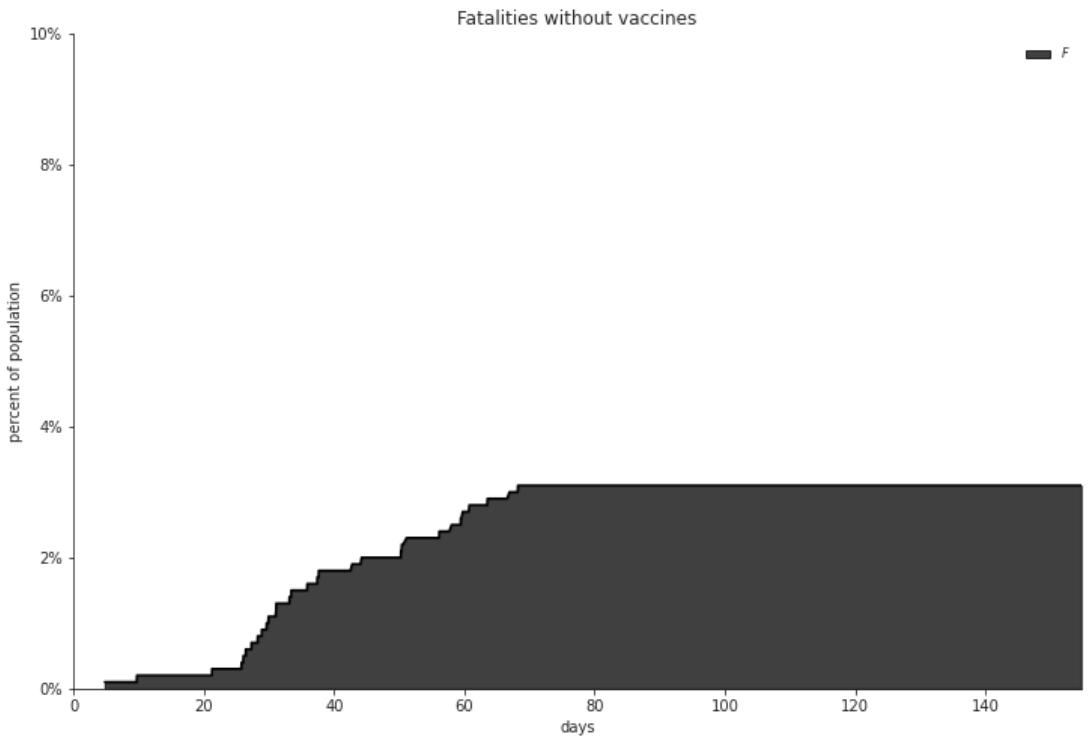
4.5.1 Simulation 0: Baseline - No Vaccinations

We run our simulation with no changes to the hospilization and fatality rates from our source data to get our baseline graph for comparison. We print the percent fatalities of our population and its graph.

[ ]:
```
ageGroup_pctHospitalized = {
                            '0-9':      0.0000,
                            '10-19':    0.0004,
                            '20-29':    0.0104,
                            '30-39':    0.0343,
                            '40-49':    0.0425,
                            '50-59':    0.0816,
                            '60-69':    0.218,
                            '70-79':    0.466,
                            '80+':      0.684
                            }
PCT_HOSPITALIZED = [ageGroup_pctHospitalized[ageGroup] for ageGroup in ageGroups]

ageGroup_hospitalFatalityRate = {
                                 '0-9':     0.0000,
                                 '10-19':   0.3627,
                                 '20-29':   0.0577,
```

```
                                    '30-39':   0.3426,
                                    '40-49':   0.3694,
                                    '50-59':   0.3532,
                                    '60-69':   0.3381,
                                    '70-79':   0.5187,
                                    '80+':     0.7283
                                    }
PCT_FATALITY = [ageGroup_hospitalFatalityRate[ageGroup] for ageGroup in␣
 ↪ageGroups]

BETA = .3
model = ExtSEIRSNetworkModel(G=G_baseline, p=P_GLOBALINTXN,
                             beta=BETA, sigma=SIGMA, lamda=LAMDA, gamma=GAMMA,
                             gamma_asym=GAMMA, eta=ETA, gamma_H=GAMMA_H,␣
 ↪mu_H=MU_H,
                             a=PCT_ASYMPTOMATIC, h=PCT_HOSPITALIZED,␣
 ↪f=PCT_FATALITY,
                             alpha=ALPHA,␣
 ↪beta_pairwise_mode=BETA_PAIRWISE_MODE, delta_pairwise_mode=DELTA_PAIRWISE_MODE,
                             G_Q=G_quarantine, q=0, beta_Q=BETA_Q,␣
 ↪isolation_time=14,
                             initE=INIT_EXPOSED, mu_0 = 5.20)

model.run(300)
results_summary(model)
fig, ax = model.figure_basic(plot_F='stacked', plot_I_asym=False,␣
 ↪plot_I_pre=False,
                            plot_R=False, plot_H=False, plot_E=False,
                            plot_I_sym=False, plot_S=False, ylim=.1,
                            title="Fatalities without vaccines")
noVaccModel = model
```

```
total percent infected: 54.60%
total percent fatality: 3.10%
peak  pct hospitalized: 1.70%
```



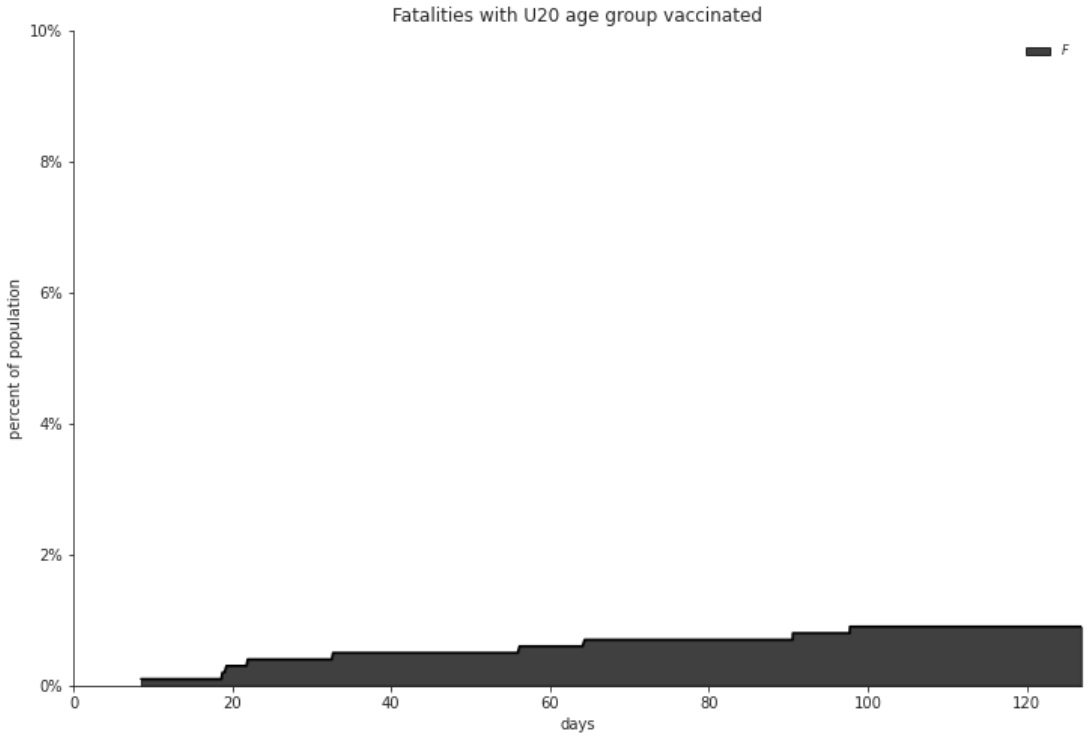
4.5.2 Simulation 1: Prioritizing Age Group Under 20

We run the simulation with the hospitilization and resulting fatality rates for the under 20 age group significantly reduced. Nodes in this age group will now be considered vaccinated for this model.

[ ]:
```
ageGroup_pctHospitalized = {
                            '0-9':      0.0000,
                            '10-19':    0.0000,
                            '20-29':    0.0104,
                            '30-39':    0.0343,
                            '40-49':    0.1425,
                            '50-59':    0.1816,
                            '60-69':    0.218,
                            '70-79':    0.366,
                            '80+':      0.400
                            }
PCT_HOSPITALIZED = [ageGroup_pctHospitalized[ageGroup] for ageGroup in ageGroups]

ageGroup_hospitalFatalityRate = {
                                 '0-9':     0.0000,
                                 '10-19':   0.0000,
                                 '20-29':   0.0577,
```

```
                                    '30-39':   0.3426,
                                    '40-49':   0.3694,
                                    '50-59':   0.3532,
                                    '60-69':   0.3381,
                                    '70-79':   0.5187,
                                    '80+':     0.7283
                                    }
PCT_FATALITY = [ageGroup_hospitalFatalityRate[ageGroup] for ageGroup in␣
 ↪ageGroups]

BETA = .2
model1 = ExtSEIRSNetworkModel(G=G_baseline, p=P_GLOBALINTXN,
                              beta=BETA, sigma=SIGMA, lamda=LAMDA, gamma=GAMMA,
                              gamma_asym=GAMMA, eta=ETA, gamma_H=GAMMA_H,␣
 ↪mu_H=MU_H,
                              a=PCT_ASYMPTOMATIC, h=PCT_HOSPITALIZED,␣
 ↪f=PCT_FATALITY,
                              alpha=ALPHA,␣
 ↪beta_pairwise_mode=BETA_PAIRWISE_MODE, delta_pairwise_mode=DELTA_PAIRWISE_MODE,
                              G_Q=G_quarantine, q=0, beta_Q=BETA_Q,␣
 ↪isolation_time=14,
                              initE=INIT_EXPOSED)

# for node in nodesU20:
#   model1.X[node] = 7
#   model1.set_isolation(model1.X[node], True)

model1.run(300)
results_summary(model1)
fig, ax = model1.figure_basic(plot_F='stacked', plot_I_asym=False,␣
 ↪plot_I_pre=False,
                              plot_R=False, plot_H=False, plot_E=False,
                              plot_I_sym=False, plot_S=False, ylim=.1,
                              title="Fatalities with U20 age group vaccinated")
under20Model = model1
```

```
total percent infected: 24.40%
total percent fatality: 0.90%
peak  pct hospitalized: 0.60%
```



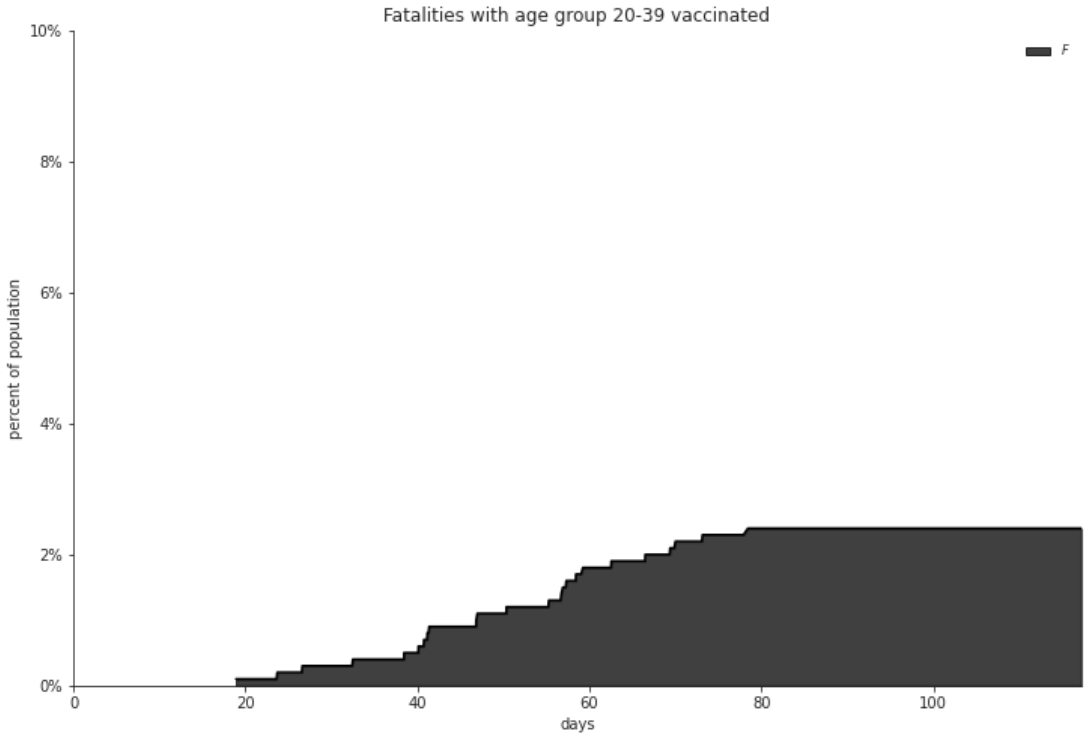
4.5.3 Simulation 2: Prioritizing Age Group 20-39

We run the simulation with the hospitilization and resulting fatality rates for the 20-39 age group significantly reduced. Nodes in this age group will now be considered vaccinated for this model.

[ ]:
```
ageGroup_pctHospitalized = {
                            '0-9':      0.0000,
                            '10-19':    0.0004,
                            '20-29':    0.0000,
                            '30-39':    0.0000,
                            '40-49':    0.0425,
                            '50-59':    0.0816,
                            '60-69':    0.218,
                            '70-79':    0.366,
                            '80+':      0.484
                            }
PCT_HOSPITALIZED = [ageGroup_pctHospitalized[ageGroup] for ageGroup in ageGroups]

ageGroup_hospitalFatalityRate = {
                                 '0-9':     0.0000,
                                 '10-19':   0.3627,
                                 '20-29':   0.0000,
                                 '30-39':   0.0000,
```

```
                                    '40-49':   0.3694,
                                    '50-59':   0.3532,
                                    '60-69':   0.3381,
                                    '70-79':   0.5187,
                                    '80+':     0.7283
                                    }
PCT_FATALITY = [ageGroup_hospitalFatalityRate[ageGroup] for ageGroup in␣
 ↪ageGroups]

BETA = .2
model2 = ExtSEIRSNetworkModel(G=G_baseline, p=P_GLOBALINTXN,
                              beta=BETA, sigma=SIGMA, lamda=LAMDA, gamma=GAMMA,
                              gamma_asym=GAMMA, eta=ETA, gamma_H=GAMMA_H,␣
 ↪mu_H=MU_H,
                              a=PCT_ASYMPTOMATIC, h=PCT_HOSPITALIZED,␣
 ↪f=PCT_FATALITY,
                              alpha=ALPHA,␣
 ↪beta_pairwise_mode=BETA_PAIRWISE_MODE, delta_pairwise_mode=DELTA_PAIRWISE_MODE,
                              G_Q=G_quarantine, q=0, beta_Q=BETA_Q,␣
 ↪isolation_time=14,
                              initE=INIT_EXPOSED)

model2.run(300)
results_summary(model2)
fig, ax = model2.figure_basic(plot_F='stacked', plot_I_asym=False,␣
 ↪plot_I_pre=False,
                             plot_R=False, plot_H=False, plot_E=False,
                             plot_I_sym=False, plot_S=False, ylim=.1,
                             title="Fatalities with age group 20-39 vaccinated")
to39Model = model2
```

```
total percent infected: 29.50%
total percent fatality: 2.40%
peak  pct hospitalized: 0.70%
```



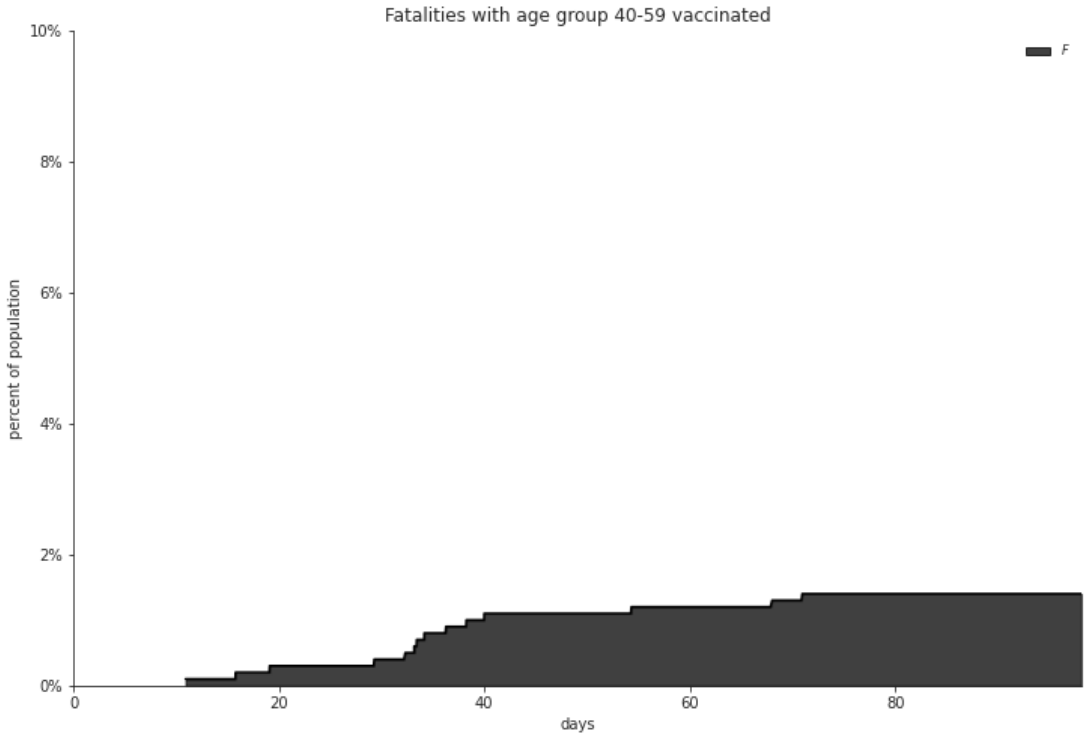
4.5.4 Simulation 3: Prioritizing Age Group 40-59

We run the simulation with the hospitilization and resulting fatality rates for the 40-59 age group significantly reduced. Nodes in this age group will now be considered vaccinated for this model.

[ ]:
```
ageGroup_pctHospitalized = {
                            '0-9':      0.0000,
                            '10-19':    0.0004,
                            '20-29':    0.0104,
                            '30-39':    0.0343,
                            '40-49':    0.0000,
                            '50-59':    0.0000,
                            '60-69':    0.318,
                            '70-79':    0.366,
                            '80+':      0.400
                            }
PCT_HOSPITALIZED = [ageGroup_pctHospitalized[ageGroup] for ageGroup in ageGroups]

ageGroup_hospitalFatalityRate = {
                                 '0-9':     0.0000,
                                 '10-19':   0.3627,
                                 '20-29':   0.0577,
                                 '30-39':   0.3426,
```

```
                                    '40-49':   0.0000,
                                    '50-59':   0.0000,
                                    '60-69':   0.3381,
                                    '70-79':   0.5187,
                                    '80+':     0.7283
                                    }
PCT_FATALITY = [ageGroup_hospitalFatalityRate[ageGroup] for ageGroup in␣
 ↪ageGroups]

BETA = .2
model3 = ExtSEIRSNetworkModel(G=G_baseline, p=P_GLOBALINTXN,
                              beta=BETA, sigma=SIGMA, lamda=LAMDA, gamma=GAMMA,
                              gamma_asym=GAMMA, eta=ETA, gamma_H=GAMMA_H,␣
 ↪mu_H=MU_H,
                              a=PCT_ASYMPTOMATIC, h=PCT_HOSPITALIZED,␣
 ↪f=PCT_FATALITY,
                              alpha=ALPHA,␣
 ↪beta_pairwise_mode=BETA_PAIRWISE_MODE, delta_pairwise_mode=DELTA_PAIRWISE_MODE,
                              G_Q=G_quarantine, q=0, beta_Q=BETA_Q,␣
 ↪isolation_time=14,
                              initE=INIT_EXPOSED)

model3.run(300)
results_summary(model3)
fig, ax = model3.figure_basic(plot_F='stacked', plot_I_asym=False,␣
 ↪plot_I_pre=False,
                             plot_R=False, plot_H=False, plot_E=False,
                             plot_I_sym=False, plot_S=False, ylim=.1,
                             title="Fatalities with age group 40-59 vaccinated")
to59Model = model3
```

```
total percent infected: 30.70%
total percent fatality: 1.40%
peak  pct hospitalized: 1.20%
```



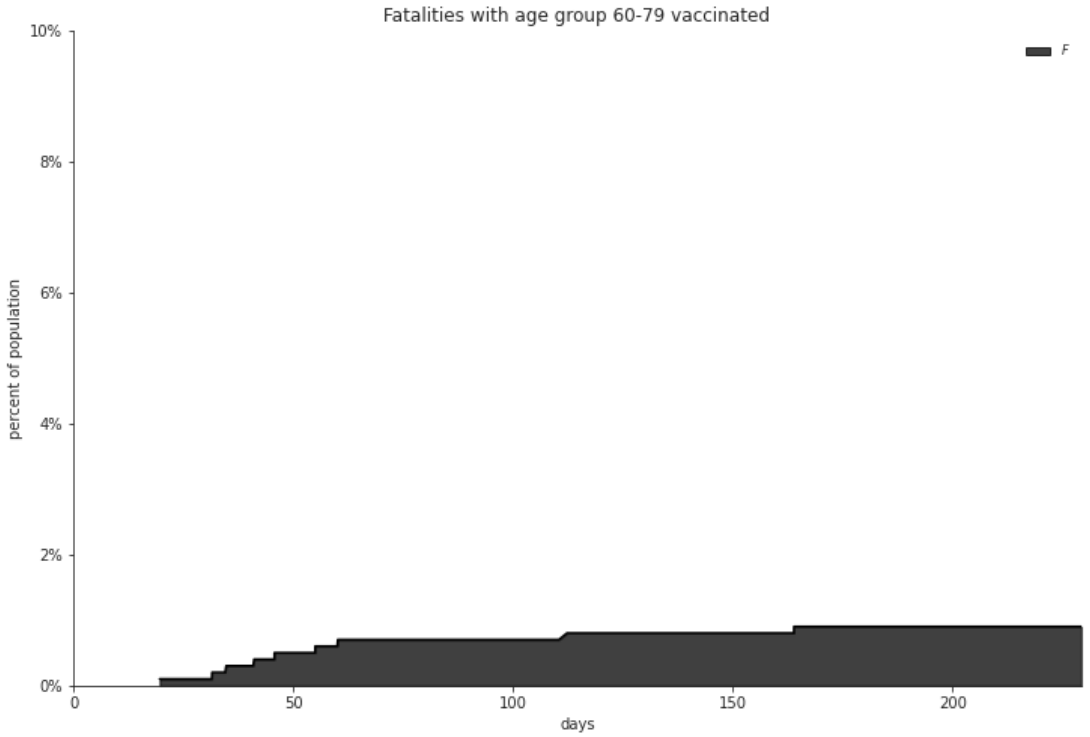
4.5.5 Simulation 4: Prioritizing Age Group 60-79

We run the simulation with the hospitilization and resulting fatality rates for the 60-79 age group significantly reduced. Nodes in this age group will now be considered vaccinated for this model.

[ ]:
```
ageGroup_pctHospitalized = {
                            '0-9':      0.0004,
                            '10-19':    0.0030,
                            '20-29':    0.0104,
                            '30-39':    0.1343,
                            '40-49':    0.1425,
                            '50-59':    0.1816,
                            '60-69':    0.0000,
                            '70-79':    0.0000,
                            '80+':      0.384
                            }
PCT_HOSPITALIZED = [ageGroup_pctHospitalized[ageGroup] for ageGroup in ageGroups]

ageGroup_hospitalFatalityRate = {'0-9':     0.0004,
                                 '10-19':   0.3627,
                                 '20-29':   0.3577,
                                 '30-39':   0.3426,
                                 '40-49':   0.3694,
```

```
                                    '50-59':   0.3532,
                                    '60-69':   0.0000,
                                    '70-79':   0.0000,
                                    '80+':     0.7283
                                    }
PCT_FATALITY = [ageGroup_hospitalFatalityRate[ageGroup] for ageGroup in␣
 ↪ageGroups]

BETA = .2
model4 = ExtSEIRSNetworkModel(G=G_baseline, p=P_GLOBALINTXN,
                              beta=BETA, sigma=SIGMA, lamda=LAMDA, gamma=GAMMA,
                              gamma_asym=GAMMA, eta=ETA, gamma_H=GAMMA_H,␣
 ↪mu_H=MU_H,
                              a=PCT_ASYMPTOMATIC, h=PCT_HOSPITALIZED,␣
 ↪f=PCT_FATALITY,
                              alpha=ALPHA,␣
 ↪beta_pairwise_mode=BETA_PAIRWISE_MODE, delta_pairwise_mode=DELTA_PAIRWISE_MODE,
                              G_Q=G_quarantine, q=0, beta_Q=BETA_Q,␣
 ↪isolation_time=14,
                              initE=INIT_EXPOSED)

model4.run(300)
results_summary(model4)
fig, ax = model4.figure_basic(plot_F='stacked', plot_I_asym=False,␣
 ↪plot_I_pre=False,
                            plot_R=False, plot_H=False, plot_E=False,
                            plot_I_sym=False, plot_S=False, ylim=.1,
                            title="Fatalities with age group 60-79 vaccinated")

to79Model = model4
```

```
total percent infected: 30.90%
total percent fatality: 0.90%
peak  pct hospitalized: 0.50%
```



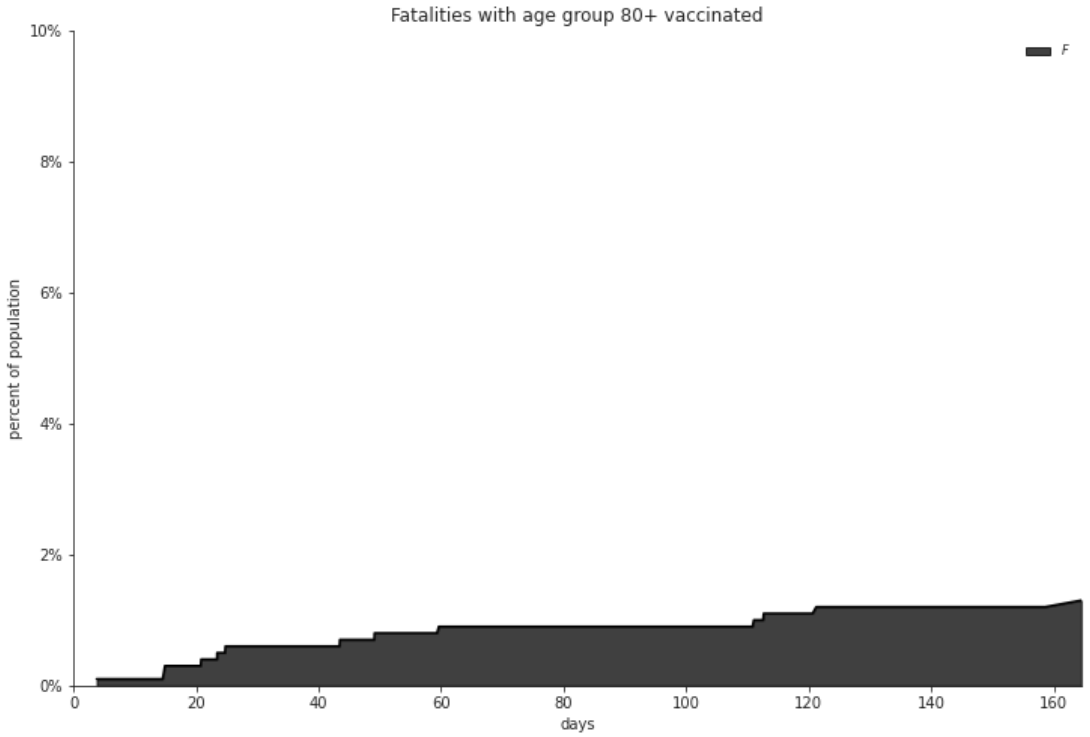
4.5.6 Simulation 5: Prioritizing Age Group 80+

We run the simulation with the hospitilization and resulting fatality rates for the 80+ age group significantly reduced. Nodes in this age group will now be considered vaccinated for this model.

[ ]:
```
ageGroup_pctHospitalized = {
                            '0-9':      0.0004,
                            '10-19':    0.0030,
                            '20-29':    0.0104,
                            '30-39':    0.0343,
                            '40-49':    0.1425,
                            '50-59':    0.1816,
                            '60-69':    0.318,
                            '70-79':    0.366,
                            '80+':      0.0000
                            }
PCT_HOSPITALIZED = [ageGroup_pctHospitalized[ageGroup] for ageGroup in ageGroups]

ageGroup_hospitalFatalityRate = {'0-9':     0.0004,
                                 '10-19':   0.3627,
                                 '20-29':   0.3577,
                                 '30-39':   0.3426,
                                 '40-49':   0.3694,
```

```
                                    '50-59':   0.3532,
                                    '60-69':   0.3381,
                                    '70-79':   0.5187,
                                    '80+':     0.0000
                                    }
PCT_FATALITY = [ageGroup_hospitalFatalityRate[ageGroup] for ageGroup in␣
 ↪ageGroups]

BETA = .2
model5 = ExtSEIRSNetworkModel(G=G_baseline, p=P_GLOBALINTXN,
                              beta=BETA, sigma=SIGMA, lamda=LAMDA, gamma=GAMMA,
                              gamma_asym=GAMMA, eta=ETA, gamma_H=GAMMA_H,␣
 ↪mu_H=MU_H,
                              a=PCT_ASYMPTOMATIC, h=PCT_HOSPITALIZED,␣
 ↪f=PCT_FATALITY,
                              alpha=ALPHA,␣
 ↪beta_pairwise_mode=BETA_PAIRWISE_MODE, delta_pairwise_mode=DELTA_PAIRWISE_MODE,
                              G_Q=G_quarantine, q=0, beta_Q=BETA_Q,␣
 ↪isolation_time=14,
                              initE=INIT_EXPOSED)

model5.run(300)
results_summary(model5)
fig, ax = model5.figure_basic(plot_F='stacked', plot_I_asym=False,␣
 ↪plot_I_pre=False,
                             plot_R=False, plot_H=False, plot_E=False,
                             plot_I_sym=False, plot_S=False, ylim=.1,
                             title="Fatalities with age group 80+ vaccinated")
over80Model = model5
```

```
total percent infected: 32.60%
total percent fatality: 1.30%
peak  pct hospitalized: 0.60%
```

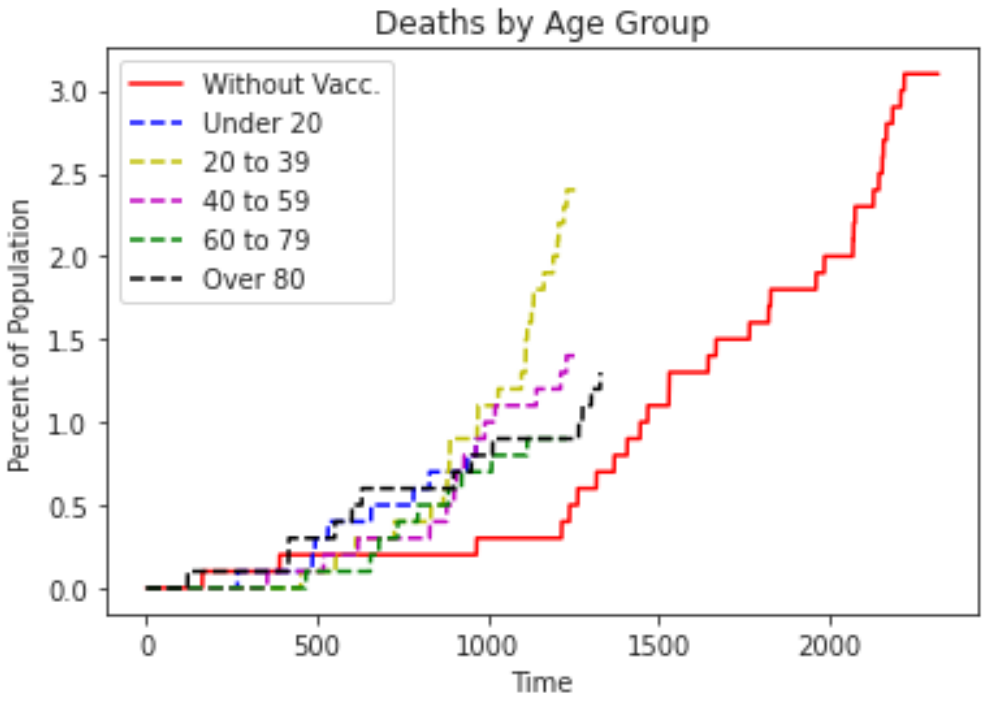


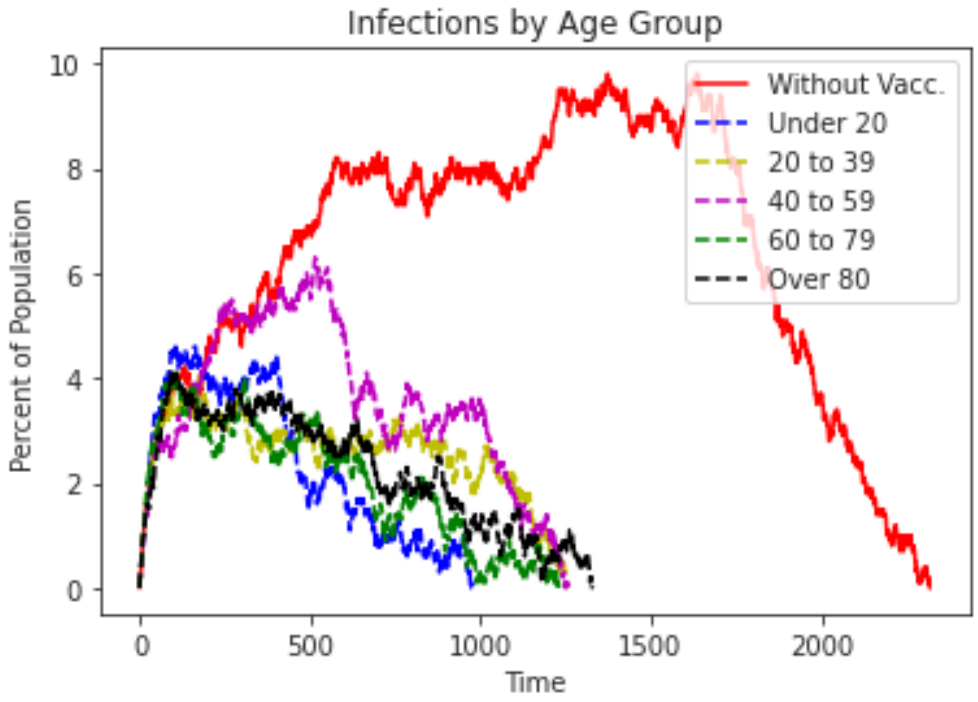
5 Evaluation of Results

Plot that combines the results of prioritizing a single age group in each scenario, side-by-side with the baseline simulating just the COVID-19 spread with no vaccination distribution:

[ ]:
```
import matplotlib.pyplot as plt

totalinfNO = noVaccModel.numI_asym + noVaccModel.numI_pre + noVaccModel.numI_sym
totalinf20 = under20Model.numI_asym + under20Model.numI_pre + under20Model.
 ↪numI_sym
totalinf39 = to39Model.numI_asym + to39Model.numI_pre + to39Model.numI_sym
totalinf59 = to59Model.numI_asym + to59Model.numI_pre + to59Model.numI_sym
totalinf79 = to79Model.numI_asym + to79Model.numI_pre + to79Model.numI_sym
totalinf80 = over80Model.numI_asym + over80Model.numI_pre + over80Model.numI_sym

plt.plot(noVaccModel.numF/10,'r',label='Without Vacc.')
plt.plot(under20Model.numF/10,'b--',label='Under 20')
plt.plot(to39Model.numF/10,'y--',label='20 to 39')
plt.plot(to59Model.numF/10,'m--',label='40 to 59')
plt.plot(to79Model.numF/10,'g--',label='60 to 79')
plt.plot(over80Model.numF/10,'k--',label='Over 80')
plt.xlabel('Time')
```

```
plt.ylabel('Percent of Population')
plt.legend()
plt.title("Deaths by Age Group")
plt.show()

plt.plot(totalinfNO/10,'r',label='Without Vacc.')
plt.plot(totalinf20/10,'b--',label='Under 20')
plt.plot(totalinf39/10,'y--',label='20 to 39')
plt.plot(totalinf59/10,'m--',label='40 to 59')
plt.plot(totalinf79/10,'g--',label='60 to 79')
plt.plot(totalinf80/10,'k--',label='Over 80')
plt.xlabel('Time')
plt.ylabel('Percent of Population')
plt.legend()
plt.title("Infections by Age Group")
plt.show()
```





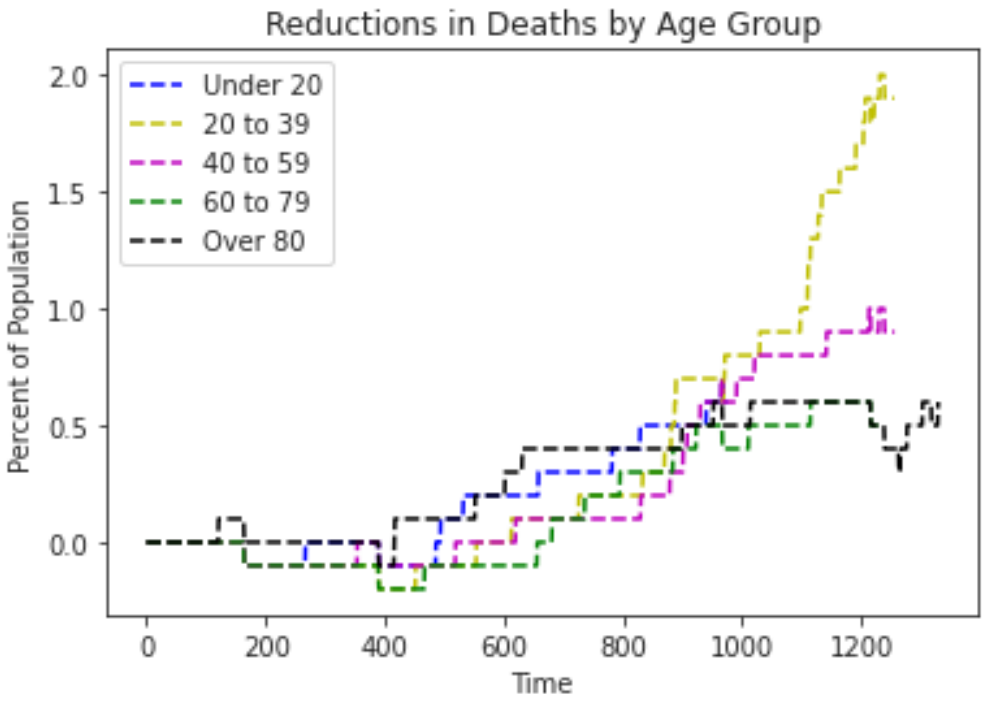
The above graphs show the following:

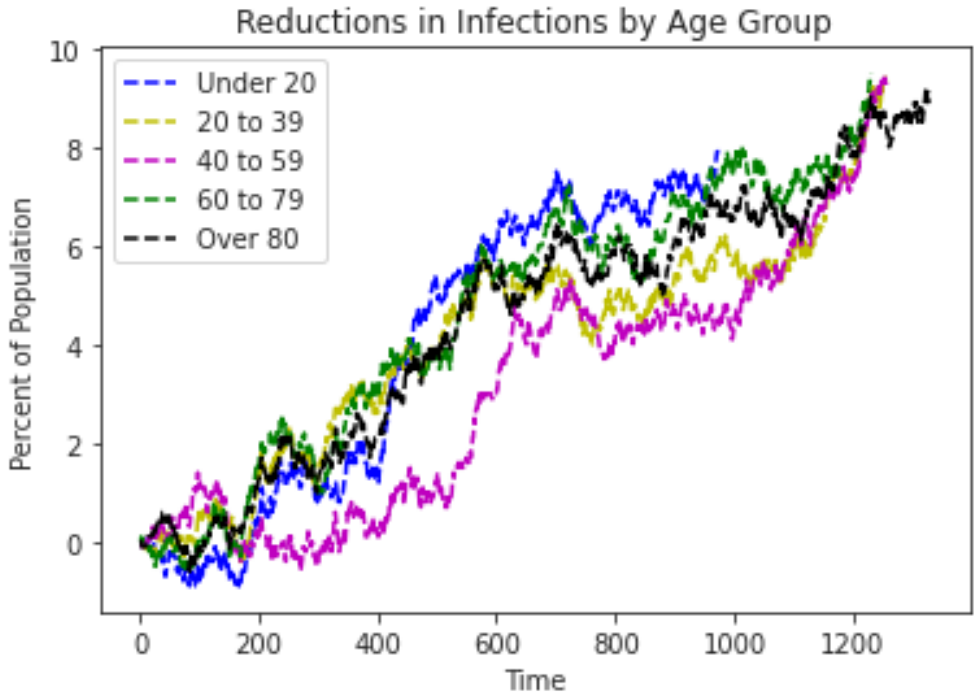
1. The maximum amount of deaths and infections achieved throughout the simulations was by the no-vaccination COVID-19 spread model. This is accurate because the introduction of vaccine distribution decreases the infection and spread of the virus, in turn decreasing the total deaths.
2. Prioritizing each of the age groups in the population for the vaccine consistently produced lower cumulative deaths and daily infections, which indicates that the vaccine is working.
3. The simulations with vaccine distributions produced much faster and better results in decreasing the deaths and infections. This can be seen by the vaccine distribution models completing at earlier time stamps than the no-vaccination COVID-19 spread model.

Plot of the **Reductions in Deaths by Age Group** (cumulative deaths) and **Reductions in Infections by Age Group** (daily infections) by prioritizing a single age group for the vaccine in each scenario. To calculate the reductions in both deaths and infections, we used the results from the simulation of the COVID-19 spread without vaccine distribution, and took the difference between the baseline and the results gathered from prioritizing each of the age groups:

[ ]:
```
plt.plot([(b_i - a_i)/10 for a_i, b_i in zip(noVaccModel.numF, under20Model.
 ↪numF)],'b--',label='Under 20')
plt.plot([(b_i - a_i)/10 for a_i, b_i in zip(noVaccModel.numF, to39Model.
 ↪numF)],'y--',label='20 to 39')
plt.plot([(b_i - a_i)/10 for a_i, b_i in zip(noVaccModel.numF, to59Model.
 ↪numF)],'m--',label='40 to 59')
```

```
plt.plot([(b_i - a_i)/10 for a_i, b_i in zip(noVaccModel.numF, to79Model.
 ↪numF)],'g--',label='60 to 79')
plt.plot([(b_i - a_i)/10 for a_i, b_i in zip(noVaccModel.numF, over80Model.
 ↪numF)],'k--',label='Over 80')
plt.xlabel('Time')
plt.ylabel('Percent of Population')
plt.legend()
plt.title("Reductions in Deaths by Age Group")
plt.show()

plt.plot([(a_i - b_i)/10 for a_i, b_i in zip(totalinfNO,␣
 ↪totalinf20)],'b--',label='Under 20')
plt.plot([(a_i - b_i)/10 for a_i, b_i in zip(totalinfNO,␣
 ↪totalinf39)],'y--',label='20 to 39')
plt.plot([(a_i - b_i)/10 for a_i, b_i in zip(totalinfNO,␣
 ↪totalinf59)],'m--',label='40 to 59')
plt.plot([(a_i - b_i)/10 for a_i, b_i in zip(totalinfNO,␣
 ↪totalinf79)],'g--',label='60 to 79')
plt.plot([(a_i - b_i)/10 for a_i, b_i in zip(totalinfNO,␣
 ↪totalinf80)],'k--',label='Over 80')
plt.xlabel('Time')
plt.ylabel('Percent of Population')
plt.legend()
plt.title("Reductions in Infections by Age Group")
plt.show()
```





This is the artifact of our vaccine prioritization simulation using the SEIR model. The graphs show reductions in deaths and infections by prioritized age group. The higher the graph line, the higher the reduction, the better the effect.

Reductions in Deaths:

Highest Reduction: Prioritizing the *20-39* age group
Lowest Reduction: Prioritizing the *Over 80* age group

Reductions in Infections:

Highest Reduction: Prioritizing the *Under 20* age group
Lowest Reduction: Prioritizing the *Over 80* age group

Using the SEIR model was powerful in providing the tools to be able to simulate the effect of vaccinations against the baseline. Although prioritizing each age group produces far better results than the baseline, some age groups perform better than others. From the results, we can see that prioritizing the *Over 80* age group is the least advantageous since it provides the lowest reduction and deaths over the simulation period. Our goal is to make a decision on which age group to prioritize in order to reduce deaths and infections, and the simulations using the SEIR model show that prioritizing the *20-39* age group produces the largest reduction in deaths, while prioritizing the *Under 20* age group produces the largest reduction in infections. To maximize the benefits, the vaccine prioritization should be towards one of those two age groups.

2 Model Verification and Validation

Verifying and validating our system is as important as the simulation itself. The goal is to improve the credibility and confidence in our model, to ensure that the results we achieve can assist in reaching a well-motivated, and correct, conclusion. The following sections focus on the *model verification* and *model validation* steps taken to ensure that we built the models correctly as well as built the correct models.

2.1 Verification

Model verification is the process where we ask the question: *Did we build the model correctly?* Both the Cellular Automata and SEIR models were verified during and after their programming, which involved intensive debugging. Our aim in this activity is to make sure that both models follow their stated conceptual models. The following are the verification steps taken for both models:

1. Cellular Automata model:
    - The state transitions, as shown in the conceptual model, were tracked to make sure that individuals on the grid are transitioning to their correct state. This was done by reducing the number of individuals in the model to 100 and using print statements to show the state of each individual at each time step.
    - There was a global and local counter for the vaccination distribution, where the global counter kept track of the total vaccines distributed over all time steps and the local counter kept track of the total vaccines distributed over the current time step. This was used to make sure that the vaccine rollout percentage was accurately followed.
    - During simulation runs, there were print statements that displayed the total number of individuals in each state, along with the running statistics of the simulation. This was cross checked with hand calculations.
2. SEIR model:
    - The age-stratified layers that were passed into the network generation function was the most crucial initial step to verify. The distribution and demographics that were used to create the layers and the households were verified by executing the cell block for the creation process several times, where we noted the age group distribution at each execution. At each execution, the number of nodes per age group were printed out and were verified to correspond closely to the demographic distribution in the US.
    - The distribution of values for each of the parameters to the model were verified by running our input values through an online distribution calculator and making sure that our final parameters have the desired value as compared to the outside tool.
    - Similar to the CA model, there were print statements displaying the total number of individuals in each state at each time step in the simulation. This was helpful to make sure that the values are correct at a face examination.

2.2 Validation

Model validation is the process where we ask the question: *Did we build the correct model?* This validation deals with checking if the simulation model matches the envisioned system and problem description. We chose to evaluate and simulate our original problem description using two validation techniques:

1. Behavior validation: This involves validating using a collection of anticipated known behaviors. The two behaviors we focused on to validate the models were the following:

- Increasing the vaccine rollout percentage rapidly increases the number of individuals in the "vaccinated" state and decreases infections. This behavior was validated using the interactive method of adjusting the vaccine rollout percentages before the start of a simulation run. It was found that as we increased the vaccine rollout percentage, the individuals in the "vaccinated" state did in fact increase and the infections decreased as a result.
- Increasing the percentage of initial infections at the beginning of the model leads to a faster peak in infections. This behavior was validated by as well adjusting the initial infections percentage and evaluating the simulation run graphs that were plotted at the completion of a run. The behavior was validated in both models and they behaved as they intuitively should.

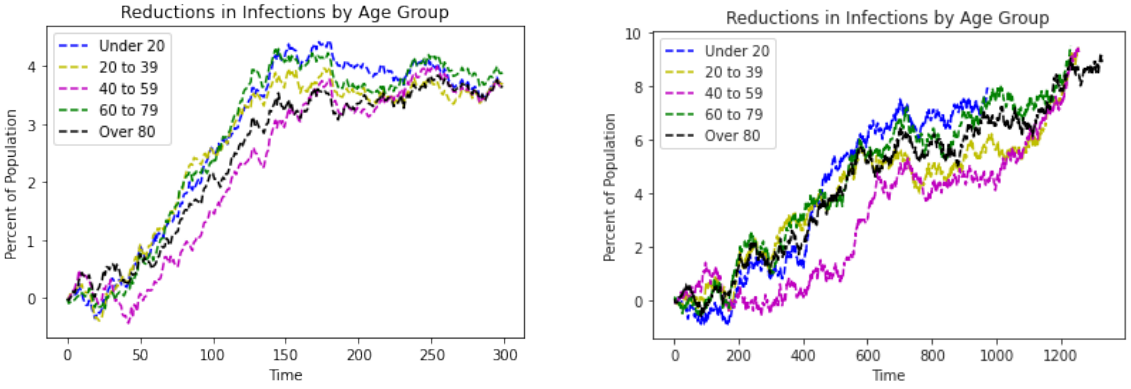
2. Comparing against another simulation model: This was done by comparing the behavior and results of the Cellular Automata model to that of the SEIR model, and vice versa. In order for the models to be validated, they need to produce approximately the same results, given the same input variables and values. The models were fully validated because they both produced almost identical results, with the variations in behavior being quantitatively small. The results of both models will be discussed in Section 3.

3 Conclusion: The Prioritized Age Group

This section focuses on comparing the results of both the Cellular Automata model and the SEIR model to answer the important question: *Which age group should be prioritized in the vaccine distribution?*

Given that there is only a limited quantity of the COVID-19 vaccines available, the goal of the project was to create a simulation in order to arrive at a well-motivated and data-driven decision on which age group to prioritize in the vaccine distribution plan with the goal of *slowing down infections* and *decreasing death rates*. Both models provided very rich and data-driven results that simulate the different scenarios of prioritizing one of the five age groups. In order to arrive at an answer to our question, we need to evaluate each of the two goals we had.

Slowing down infections As compared to the baseline of no vaccinations, both models came to the conclusion that the introduction and administration of vaccines decreased the daily infections in the system. Figure 1 shows the reduction in infections by prioritizing each of the age groups in both the CA and SEIR models.



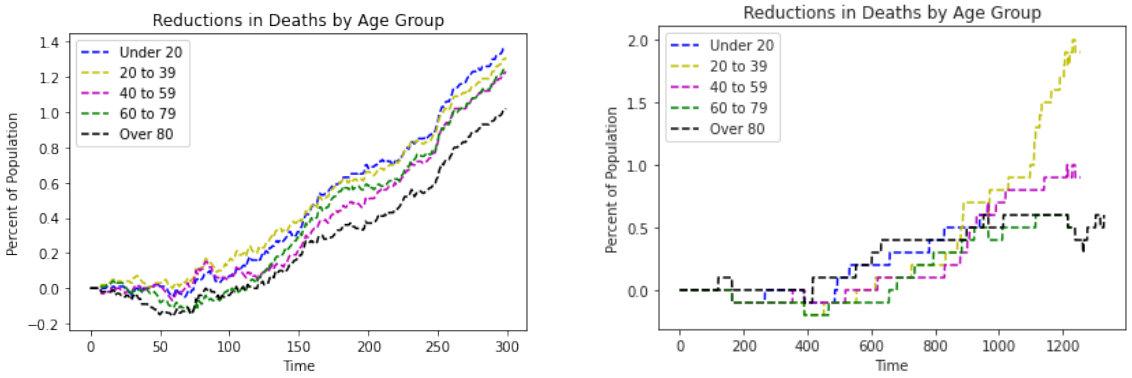
(a) Cellular Automata Model

(b) SEIR Model

Figure 1: Reductions in infections by age group for both CA and SEIR models.

The results produced by both models are very close. The first striking observation to notice is that both models conclude that prioritizing the 40-59 years age group consistently produces the lowest reductions in infections. This can be seen by the data line of the age group being the lowest in both models. In both models, the under 20 and 60-79 years age group data lines are the highest and produce the largest reductions in infections consistently. Using this observation, it suffices to say that the *under 20* and *60-79 years* age groups should be prioritized given that our goal is to slow down infections.

Decreasing death rates Consistent with the previous observations, it can as well be seen that both models conclude that the vaccinated scenarios produce a higher reduction in deaths than that of a non-vaccinated scenario. Figure 2 shows the reduction in deaths by prioritizing each of the age groups in both the CA and SEIR models.



(a) Cellular Automata Model

(b) SEIR Model

Figure 2: Reductions in deaths by age group for both CA and SEIR models.

Once again, the results produced by both models are very close. The first striking observation to notice is that both models agree that prioritizing the over 80 age group consistently produces the lowest reductions in deaths. This can be seen in Figure 2 through the data line of the age group being the lowest in both models. In the CA model, the under 20 age group produces the largest reduction in deaths, closely followed by the 20-39 age group. Similarly, in the SEIR model, the 20-39 age group produces the largest reduction in deaths. Due to the variation in results from both models, our finalized result should as well include a bias from the age group chosen to prioritize in the first goal.

Overall, to slow down infections it is important to prioritize the under 20 or 60-79 years age groups, while to decrease death rates it is important to prioritize the under 20 or 20-39 age groups. Bringing both results together, we are able to make a robust decision. Considering that prioritizing the under 20 age group in the vaccine distribution consistently produced far better results as compared to the other age groups, it is apparent that they are the age group to be chosen in order to meet both goals of the project. **Therefore, using the results from both the CA and SEIR models, our well-motivated and data-driven decision is to prioritize the *Under 20 age group* in the vaccine distribution with the goal of slowing down infections and decreasing death rates.**

4 Literature Review

Cellular Automata models as well as SEIR models are very widely used for modeling certain aspects of infectious disease spread; in fact these models have already been used a lot to model the spread of COVID-19 since the beginning of the pandemic in early 2020.

Many papers tend to model various scenarios of an infectious disease spread. For example, a paper from 2008 [5] models the outcomes of implementing quarantine, widespread use of different medications, and changes in social behavior. This paper focuses on an avian flu like virus in the Austrian state of Tyrol, but the model allows for the user to simulate different types of infectious diseases. This study investigates optimization of medical treatment and vaccination regimens. In contrast to our own simulations, this paper mixed the Cellular Automata model with a SIR (susceptible, infectious, recovered) model.

Another paper published in 2006 [7] also simulates the spread of epidemics using Cellular Automata and a SIR model. In this study, the state of each cell represents the portion of the SIR classes of individuals at every timestep. A paper from 2015 [1] studies the dynamics of the Ebola virus, simulating the timeline of the infection and finding fatality rates. This model uses stochastic Cellular Automata to model the spread of Ebola. Another Cellular Automata model [6] that studies the use of drug therapy for HIV also uses a different variation of CA, specifically non-uniform Cellular Automata. The paper describes this version of CA as being different than normal CA models because: (1) each computational domain may contain different CA rules; (2) there are two extreme time scales for HIV infection (days and decades); (3) there are four phases of drug treatment; and (4) there is a 3-phase pattern of HIV infection.

There are also many papers published that use the SEIR model or variations of it, and a lot of the new ones focus on COVID-19. A paper from May 2020 [2] implements SEIR to compute the infected population and the number of casualties of the epidemic at that time, focusing on Italy for a portion of the model. An aspect of SEIR model studies that make one different from another is that there are different values for things like incubation period and infectious period, as well as values like infection fatality rate. For instance, this paper uses an incubation period of 4.25 days and an infectious period of 4 days. It is interesting to see how these numbers change with different studies over the course of the pandemic.

Another difference between various SEIR models is in how the differential equations are solved. For example, the May 2020 paper solved the equations with a forward Euler scheme, whereas a paper from July 2020 [4] solves the equations numerically using 4th and 5th order Runge-Kutta methods. Also, the July model uses a generation matrix approach to determine the basic reproduction number. This model was implemented to see the effects of not social distancing and/or using certain hygiene measures.

Finally, a paper from August of 2020 [3] uses a slight variation to the SEIR model: the SEIAR model, where the 'A' represents asymptomatic individuals. This model also incorporates population migration, and the results aim to verify the effectiveness of interventions like social distancing and migration control in mitigating the spread of COVID-19.

References

[1] Emily Burkhead and Jane Hawkins. "A cellular automata model of Ebola virus dynamics". In: *Physica A: Statistical Mechanics and its Applications* 438 (2015), pp. 424–435.

[2] José M Carcione et al. "A simulation of a COVID-19 epidemic based on a deterministic SEIR model". In: *Frontiers in public health* 8 (2020), p. 230.

[3] Min Chen et al. "The introduction of population migration to SEIAR for COVID-19 epidemic modeling with an efficient intervention strategy". In: *Information Fusion* 64 (2020), pp. 252–258.

[4] Samuel Mwalili et al. "SEIR model for COVID-19 dynamics incorporating the environment and social distancing". In: *BMC Research Notes* 13.1 (2020), pp. 1–5.

[5] Bernhard Pfeifer et al. “A cellular automaton framework for infectious disease spread simulation”. In: *The open medical informatics journal* 2 (2008), p. 70.

[6] Peter Sloot, Fan Chen, and Charles Boucher. “Cellular automata model of drug therapy for HIV infection”. In: *International Conference on Cellular Automata*. Springer. 2002, pp. 282–293.

[7] S Hoya White, A Martın Del Rey, and G Rodrıguez Sánchez. “Modeling epidemics using cellular automata”. In: *Applied mathematics and computation* 186.1 (2007), pp. 193–202.