Cover_Page.md

COVID-19 Outbreaks Simulation and Analysis by an Extended SEIR Model

> Georgia Tech CSE6730 Project-2

authors: Muyang Guo, Dayu Zhu, Yibo Wang

BUILD-WITH-PYTHON3

Github Repo

Project Github Repo: https://github.gatech.edu/mguo34/COVID-19-Simulation

Authors , Contributions

- Muyang Guo , Cellular Automata model
- Dayu Zhu , ODE model
- Yibo Wang , Markov Chain Model

Models

Part 1 ODE model:
https://github.gatech.edu/mguo34/COVID-19-Simulation/blob/master/Models_Notebooks/PART-1-ODE.ipynb

Part 2 Cellular Automata Model:
https://github.gatech.edu/mguo34/COVID-19-Simulation/blob/master/Models_Notebooks/PART-2-CA.ipynb

Part 3 Markov Chain Model:
https://github.gatech.edu/mguo34/COVID-19-Simulation/blob/master/Models_Notebooks/PART-3-MarkovChain.ipynb

Table of contents:

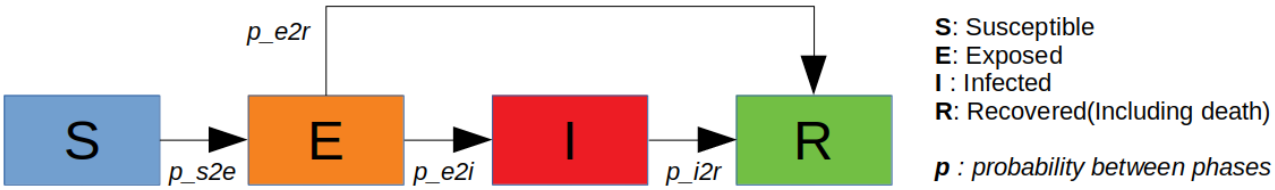
Abstract.md

Abstract

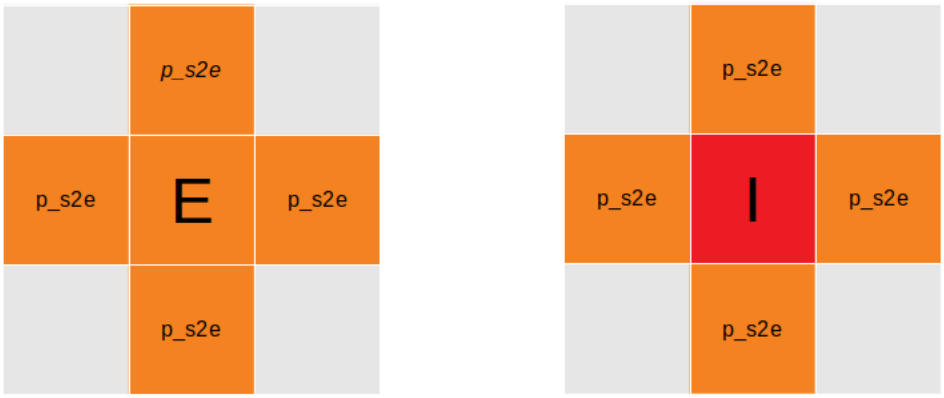
Since the first coronavirus (COVID-19) outbreaks in Wuhan, China, the COVID-19 viral disease has swept into at least 184 countries and killed more than 94,000 people.The US has become a new "center" of the outbreaks as the number of confirmed cases climbed over 450000 and the number of death reached to 16000 by 9th April 2020. It is crucial for the public to understand the transmission pattern of the disease and to take proper measures accordingly.

Groups of scientists in China, German claimed that people who are infected but do not have symptoms, or have not yet developed symptoms, can also infect others. This transmission pattern distinct among other epidemic transmission patterns such as SARS or MERS. Traditional models such as SIR or SEIR may not describe the spread correctly. Thus, an extended SEIR model was proposed and analyzed in this project paper to address the asymtomatic transmission during the incubation period. ODE, cellular automata and Markov Chain are the three approaches being used to deploy the conceptual models. The simulated results are collected, and analyzed. Real world data is also used to validate the prediction models.

Modified SEIR



Modified Infectious Pattern (CA model as an example)



Part 1: Modeling COVID-19 by continuum dynamical system : SIER system by differential equations

Table of Contents

1. SIR model

Different from most infectious dieases, the COVID-19 has a very long incubation period, which can be up to 14 days. During the incubation periods, the patients have no serve symptom, or even no symptom at all, but are still able to deliver the virus to susceptibles. Thus, a traditional susceptible-infected-recovered (SIR) model may not strictly describe the spread of COVID-19. Thus, we introduce the susceptible-exposed-infected-recovered (SEIR) model to simulate the circumstances of the pandemic.

We start from the very basic SIR model. In the SIR model, each individual has three distinct states: susceptible, infected, and recovered states. At the very begining, a few individuals in the pool are already infectious, which can be regarded as the seed of the spread, and all the rests are susceptibles. As the spread proceeds, some close contacts of the infectious are infected unfortunately, and the probability of the infection reflects the capbility of the spreading. Some infectious will recover and get the immunity and never get infected any more, while some will loss their lives. However, since the dead cases will no longer affect the distribution of S, I and R groups, and since the fatality rate is not too high to change the overall population, we will not take the death into our model.

The continuum model, which is the 'mean field' model, will only care about the overall density of the S, I and R groups without taking a close look at the exact states of individuals. Then, the continuum model will provide a general idea of the spread, so it is a handy method to model a huge population and get the mean behaviour of the spread. On the contrary, since the model is a over-simplified dynamical system, it is not straightforward to describe every details of the real world. Compared with other two methods of our study, the Markov chain and the cellular automaton methods, the dynamical system is not bounded by the size of the pool, which is easily scalable with acceptable compuytational expenses. For another, this method cannot model each cells in the system, so it is not always precise.

In SIR model, we first need to define a time-dependent variable for S, I and R state, and the variables are the fraction of the states in the overall population. That is, let

- S_t be the fraction of the population that is susceptible at time t;
- I_t be the fraction that is infected at t; and
- R_t be the fraction that is recovered at t,

where $S_t + I_t + R_t = 1$.

Besides, there are fundamental parameters defined to describe the behaviour of the virus

- $I_t S_t$ measures the fraction of total encounters that can cause disease transmission;
- τ is a parameter that represents the fraction (or probability) of such encounters transmitting disease in any time step; and
- $\frac{1}{\kappa}$ is a parameter that represents the fraction (probability) of infected individuals recovering in any time step.

Thus, we can formulate the system as a group of differential euqations, and S, I, R are collectively regarded as a vector:

$$\frac{d\vec{y}}{dt} = \frac{d}{dt}\begin{pmatrix} S(t) \\ I(t) \\ R(t) \end{pmatrix} = \begin{pmatrix} -\tau_0 I(t)S(t) \\ \tau_0 S(t)(It) - \frac{1}{\kappa_0} I(t) \\ \frac{1}{\kappa_0} I(t) \end{pmatrix} \equiv \vec{F}(\vec{y}),$$

where $\vec{y}(t)$ is the state vector and both τ_0 and κ_0 are now rate parameters, having units of "fractions per unit time."

2. Extension to SEIR model

Since COVID-19 has a very long incubation period, which makes it hard to handle. And the no-symptom patients is one of the major channel of the spread of the virus. To model the incubation peroid, we introduce another state, exposed (E), to the SIR model, and the modified system is the so-called SEIR model.

In SEIR model, the patients who get the virus will first get into the Exposed state. Then, part of the exposed patients will show symptoms and turn to the Infectious state. We use the parameter σ_0 to model the transition. Then, the system can be describe as

$$\frac{d\vec{y}}{dt} = \frac{d}{dt}\begin{pmatrix} S(t) \\ E(t) \\ I(t) \\ R(t) \end{pmatrix} = \begin{pmatrix} -\tau_0 I(t)S(t) \\ \tau_0 I(t)S(t) - \sigma_0 E(t) \\ \sigma_0 E(t) - \frac{1}{\kappa_0} I(t) \\ \frac{1}{\kappa_0} I(t) \end{pmatrix} \equiv \vec{F}(\vec{y}),$$

where $\vec{y}(t)$ is the state vector and both τ_0, σ_0 and κ_0 are now rate parameters, having units of "fractions per unit time."

We first define the function to describe the right hand side of the equations, namely E_seir, and we define x as the vector which contains the infomation of the current states. x has three components, S, E and I, and obviously R can be obtained R = 1- S - E - I

In [16]:
```
import numpy as np
import matplotlib.pyplot as plt

def F_seir (x, tau, sigma, kappa):
    # x = (s, e, i)
    x_next = x.copy ()
    ### BEGIN SOLUTION
    S, E, I = 0, 1, 2
    x_next[S] = x[S] * ( - tau*x[I])
    x_next[E] = x[S] * tau * x[I] - sigma * x[E]
    x_next[I] = sigma * x[E] + x[I] * ( - 1.0/kappa)
    ### END SOLUTION
    return x_next
```

Then the parameters and initial states are set. Here we assume initially some population are already infectious, and all the others are suspectible. Since As for time steps, we assume the time evenly distributed between 0 and 31, which means our simulation is to reflect the conditions during one month.

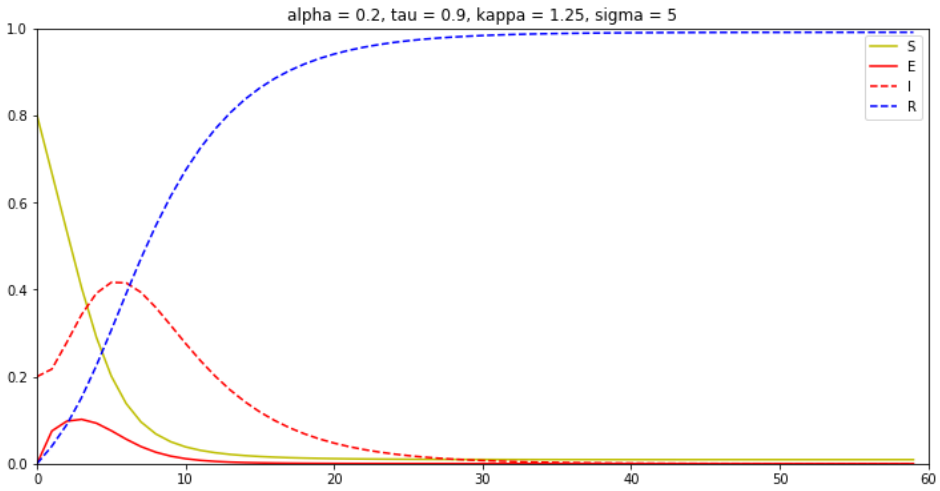
In [100]:
```
ALPHA = 0.2
TAU = 0.9
KAPPA = 1.25
SIGMA = 5

TAU_0 = TAU
KAPPA_0 = KAPPA
SIGMA_0 = SIGMA

# Initial populations, i.e., [S(0), E(0), I(0), R(0)]
y0 = np.array ([1.0 - ALPHA, 0., ALPHA, 0.])

# Time points at which to compute the solutions:
T = np.arange (60).astype (float)
```

Once upon all the equations and initial states are predefined, the following task is to solve the equations. Although formidable to human, the differential equations are easily approachable to python differential equation solvers. Here we use ode_int from scipy library. The algrithm of the solver is way beyound the content of this toturial. We only need to treat it as a automatic solver. The input is the vector of current status with explicit parameters, and it will solve for each time step. We define the function F_seir_ode to convey parameters to the solver. It just exactly looks like F_seir, but it is upon to change if we further introduce other modifications

In [101]:
```
### BEGIN SOLUTION
from scipy.integrate import odeint

############# ode for solving y; here the function is the same as F_seir, but we can change that if more functions are n
def F_seir_ode (y, t, tau0, kappa0, sigma0):
    return F_seir (y, tau0, kappa0, sigma0)

############### define the initial states
Y = np.zeros ((3, len (T)))
Y[:, 0] = y0[:3]

Y = odeint (F_seir_ode,
            Y[:, 0],
            T,
            args=(TAU_0, KAPPA_0, SIGMA_0)).T

S_ode = Y[0, :]
E_ode = Y[1, :]
I_ode = Y[2, :]
R_ode = 1.0 - S_ode - I_ode - E_ode
### END SOLUTION
```

Then we define some supporting function to output the solutions of the simulation and visulaization them. Matplotlib is a popular package for plotting, and we can easily observe the results of our results.

In [102]:
```
def summarize_sim_ode (S, E, I, T, alpha, tau_0, kappa_0, sigma_0):
    t_max = T[-1]
    print ("ODE simulation parameters:")
    print ("  - t_max = {}".format (t_max))
    print ("  - alpha = {}".format (alpha))
    print ("  - tau_0 = {}".format (tau_0))
    print ("  - kappa_0 = {}".format (kappa_0))
    print ("\nResults:")
    print ("- S_{{{}}} = {:.3f}".format (t_max, S[-1]))
    print ("- E_{{{}}} = {:.3f}".format (t_max, E[-1]))
    print ("- I_{{{}}} = {:.3f}".format (t_max, I[-1]))
    print ("- R_{{{}}} = {:.3f}".format (t_max, 1-S[-1]-I[-1] -E[-1]))

#summarize_sim_ode (S_ode, I_ode, T, ALPHA, TAU_0, KAPPA_0)

def plot_sim_ode (S, E, I, T, alpha, tau, kappa, sigma):
    t_max = T[-1]
    use_points = len (T) <= 35
    plt.plot (T, S, 'ys--' if use_points else 'y-')
    plt.plot (T, E, 'r-' if use_points else 'r-')
    plt.plot (T, I, 'r*--' if use_points else 'r--')
    plt.plot (T, 1. - S - I - E, 'bo--' if use_points else 'b--')
    plt.legend (['S', 'E' , 'I', 'R'])
    plt.axis ([0, t_max+1, 0, 1])
    plt.title ("alpha = {}, tau = {}, kappa = {}, sigma = {}".format (alpha, tau, kappa, sigma))

# Figure to compare discrete-time and continuous-time models
plt.figure (figsize=(12, 6))
plot_sim_ode (S_ode, E_ode, I_ode, T, ALPHA, TAU_0, KAPPA_0, SIGMA_0)
```



3. Validation the functionality of the SEIR model through real-world data

Upon now, we have finished all the codes for the suimulation and visualization. However, a good simulation is the one that can reflect the conditions of the real world. To test if our model can be used to describe the spread of COVID-19, we need to test it by real world data.

The first step is to load the real-world. The dataset is loaded from the John Hoplins dataset. We can use a python package, Pandas, to load and manupilate the dataset. Here we define the function preprocess_US, to load and preprocess the data. The data is about the spread conditions in the US. The dataset keeps updated.

The dataset is denoted by states. Here we do not need to distinguish the difference between states, and regard the spread in the U.S. as a whole. The confirmed and dead cases are interested.

In [134]:
```
import pandas as pd

def preprocess_US(data_url):
    df = pd.read_csv(data_url)
    last_update_date = df.keys()[-1]
    df.drop(['UID','iso2','iso3','code3','FIPS','Admin2','Lat','Long_'], axis = 1, inplace = True)
    df_cleaned= df.groupby(['Province_State'], as_index = False).sum()
    df_sorted = df_cleaned.sort_values(by=last_update_date,ascending=False).reset_index(drop=True)
    update_total = sum(df_sorted[last_update_date])
    return df_sorted, last_update_date, update_total

### Retrieve data from CSSEGISandData Github Raw content in csv
url_confirmed_US = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_ti
url_deaths_US = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_


df_confirmed_US,last_confirmed_US,total_confirmed_US = preprocess_US(url_confirmed_US)
df_deaths_US,last_deaths_US,total_deaths_US = preprocess_US(url_deaths_US)
first_date_US = df_confirmed_US.keys()[1]
print("Data Basic Statistics: \n")
print('From {} till {} the US confirmed cases total is : {} \n'.format(first_date_US,last_confirmed_US,total_confirmed_U
print('From {} till {} the US deaths cases total is : {} \n'.format(first_date_US,last_deaths_US,total_deaths_US))
mortality_rate = float(total_deaths_US)/float(total_confirmed_US)*100
print('Mortality rate in US is being updated as {:.3f}% \n'.format(mortality_rate))
US_population_list = df_deaths_US['Population']
US_population = US_population_list.sum()
df_deaths_US.drop(['Population'], axis = 1, inplace = True)
print('COVID-19 is among US population of {:.5f}% \n'.format(float(total_confirmed_US)/float(US_population)*100))

################### change the dataframe into numpy array for manuniplation
df_confirmed_US.drop(['Province_State'],axis = 1)
array_allconfirmed = np.flip(np.array(df_confirmed_US.sum(axis = 1)))/float(US_population)
df_deaths_US.drop(['Province_State'],axis = 1)
array_alldeath = np.flip(np.array(df_deaths_US.sum(axis = 1)))/float(US_population)
```

```
Data Basic Statistics:

From 1/22/20 till 4/26/20 the US confirmed cases total is : 965785

From 1/22/20 till 4/26/20 the US deaths cases total is : 54881

Mortality rate in US is being updated as 5.683%

COVID-19 is among US population of 0.28725%
```

Here we use the numpy array array_allconfirmed and array_alldeath to represent the data of the precentage of confirmed and death cases. The we need to find the best set of parameters in SEIR model which produces the least square between the simulation and the real-world data. There is no closed form way to solve the SEIR equations since we do not know the recovered and exposed infomation. We can deliver a thorough parameter-sweep to select out the best parameters. To do this, we can conduct nested for-loop to scan the parameter spaces. It is not a very efficient way, but it is straightforward and gurantees an approximate solution.

As for the initial steup, we can set ALPHA as 2e-8, which means at the begining of the speard, only a small faction of people get infected, which serves as the seed of the pandemic. The length of the simulation should be the same as that of the dataset. The following codes are to search for the best solutions if TAU, KAPPA and SIGMA. Here we scan the three parameters all from 0 to 3, and the step of the scanning is 0.1.

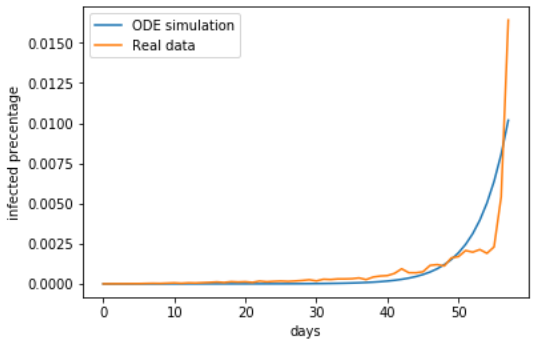
The metric for the scan is the to find the parameters which provides the least squares between the real data and simulated results of confirmed precentage (I). We define variable loss here to store the difference between the data and simulation, and the best parameters is stored in tuple best_paras.

In [130]:
```
# Initial populations, i.e., [S(0), E(0), I(0), R(0)]
ALPHA = 2e-8
n_days = len(array_alldeath)  # length of the simulation
y0 = np.array ([1.0 - ALPHA, 0., ALPHA, 0.])
T = np.arange (n_days).astype (float)
Y = np.zeros ((3, len (T)))
Y[:, 0] = y0[:3]
step = 0.1
stop = 3
loss = 5

########## the loop for parameter-scanning
for TAU_0 in np.arange(1e-7,stop,step):
    for SIGMA_0 in np.arange(1e-7,stop, step):
        for inv_kappa in np.arange(1e-7,stop,step):
            KAPPA_0 = 1/inv_kappa
            Y = odeint (F_seir_ode,
                        Y[:, 0],
                        T,
                        args=(TAU_0, KAPPA_0, SIGMA_0)).T
            I_ode = Y[2, :]
            #I_ode[:5] = 0
            #print(max(I_ode))
            #print(I_ode)
            curr_loss = np.sum(np.square(array_allconfirmed - I_ode))
            #print(curr_loss)
            if curr_loss < loss:
                loss = curr_loss
                best_paras = [TAU_0, KAPPA_0, SIGMA_0]
```

Finally, we get the best parameters: TAU_0 = 1.00, KAPPA_0 = 0.34, SIGMA_0 = 2.90; and the comparison between the simulated results and real data of the infected precentage is shown below. The simulated results can really decribe the tendency of the data, which validates the functionality of our simulation model.

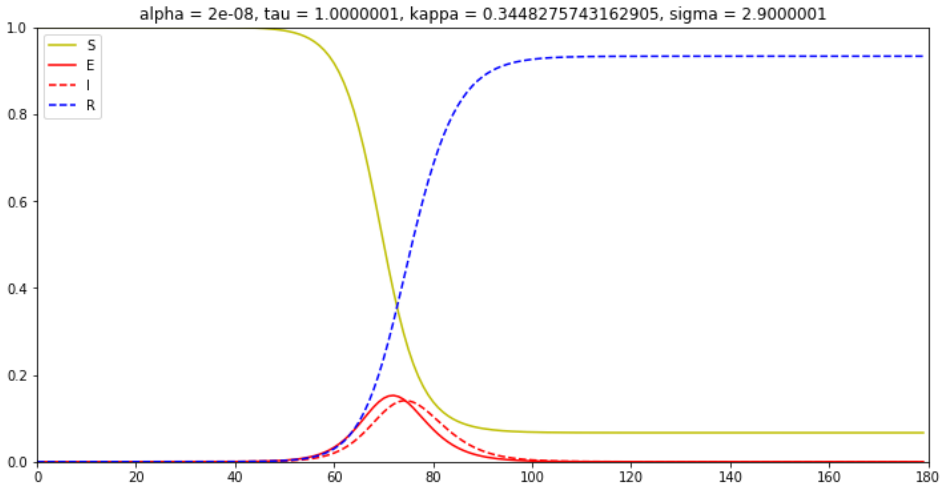
In [132]:
```
print("best parameters: ", best_paras)
TAU_0 = best_paras[0]
KAPPA_0 = best_paras[1]
SIGMA_0 = best_paras[2]


Y = odeint (F_seir_ode,Y[:, 0],T,args=(TAU_0, KAPPA_0, SIGMA_0)).T
I_ode = Y[2, :]

line_ode, = plt.plot(I_ode, label='ODE simulation')
line_data, = plt.plot(array_allconfirmed, label='Real data')
plt.xlabel('days')
plt.ylabel('infected precentage')
plt.legend(handles=[line_ode, line_data])
plt.show()
```

```
best parameters:  [1.0000001, 0.3448275743162905, 2.9000001]
```



Based on the above parameters, we can estimate the whole development of the pandemic, and then the simulation will show a reasonable guidance on the period and spread of the virus. Here we plot the spread for 180 days.

In [133]:
```
y0 = np.array ([1.0 - ALPHA, 0., ALPHA, 0.])
T = np.arange (180).astype (float)
Y = np.zeros ((3, len (T)))
Y[:, 0] = y0[:3]
Y = odeint (F_seir_ode,Y[:, 0],T,args=(TAU_0, KAPPA_0, SIGMA_0)).T

S_ode = Y[0, :]
E_ode = Y[1, :]
I_ode = Y[2, :]
R_ode = 1.0 - S_ode - I_ode - E_ode
plt.figure (figsize=(12, 6))
plot_sim_ode (S_ode, E_ode, I_ode, T, ALPHA, TAU_0, KAPPA_0, SIGMA_0)
```



The plot presents that the spread may need 4 month to terminate. We are half way, good luck everyone!

4. Conclusion and Discussion

In this part of the toturial, you should master the techniques of applying ODE in Python to solve some dynamical systems. The specific scenario is to apply ODE into analysis of COVID-19. The ODE system is called SEIR model. By solving the SEIR equations and fitting the model with real world data, we got the optimized parameters of the SEIR model. We have predicted the pandemic will come to an end in around 120 days. Keep the social distance, stay home, stay safe!

Part 2: A SEIR cellular automata model:

Table of Contents

1. Disease Spread Model: Modified SEIR with cellular automata framework

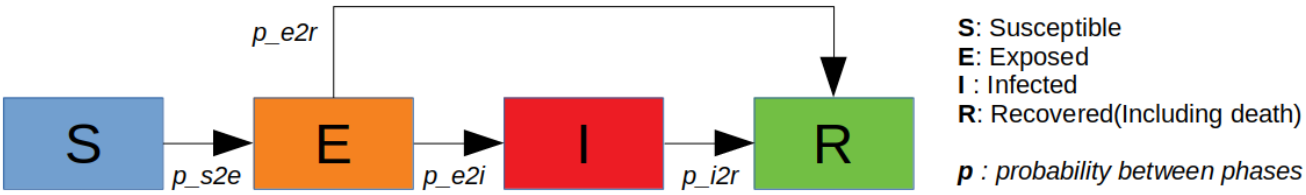
An analysis of COVID-19 publicly reported confirmed cases in the early stage of the pandemic, from Lauer SA et al. (https://annals.org/AIM/FULLARTICLE/2762808/INCUBATION-PERIOD-CORONAVIRUS-DISEASE-2019-COVID-19-FROM-PUBLICLY-REPORTED)[1], estimated that that COVID-19 has a mean incubation period approximately of 5 days, and 97.5 percent of people will develop symptons within 11.5 days of infection.
The incubation period is the time between exposure to the disease and starting to show symptoms. It is usually considered before the contagious period, when infected individual can transmit the virus to others. However, there are cases reported by Rothe, C. et al. (https://doi.org/10.1056/NEJMc2001468) [2] showing that asymptomatic persons are potential sources of 2019-nCoV infection, which means the COVID-19 could be contagious even during the late stage of incubation period. Research by Zou, Lirong, et al. (https://www.nejm.org/doi/full/10.1056/NEJMc2001737)[3] on viral load of asymptomatic infected patients also support this, and warrants a reassessment of transmission dynamics of the current outbreak.
In this simulation, a SEIR model (Susceptible, Exposed, Infectious, Recovered) was selected to describe the spread of COVID-19. Compared to the SIR model, an "Exposed" population category was added to describe the infected population during the incubation period. However, as the study suggested, it is unknown yet how large in scale can the infected people during the incubation period be infectious. To address this concern, an extra probability layer with a tunable variable was added to decribe how soon people in "E" become "I". And we assume the recovered individual will have total immune to the COVID-19.

THE **SEIR** system can be described as below:

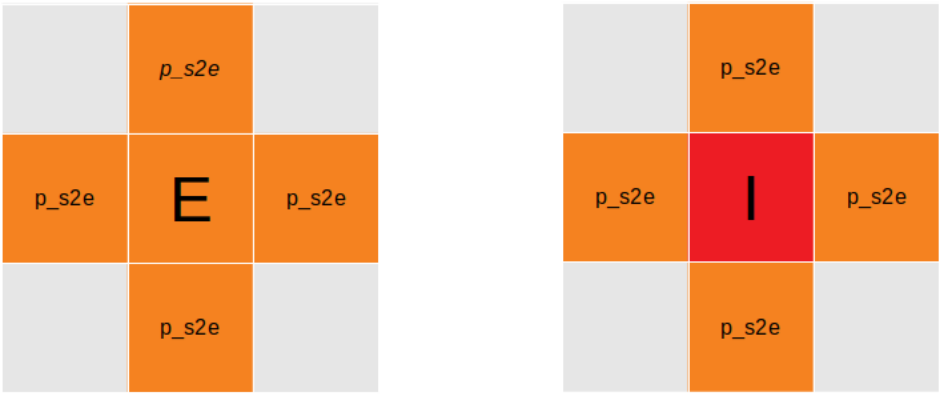
1. $N = S + E + I + R$:

   > where N is the total number of population, S is the susceptible, E is the exposed, I is the infected, R is the recovered or deaths. Here the R inclues death population, because death also stops the spread of virus, which has similar impact to the model like recovered with total immune.

1. **Modified SEIR** :
   As the study suggested, it is unknown yet how large in scale can the people during the incubation period (Status: EXPOSED) be infectious yet showing no symptoms. To address this concern, an ***extra probability layer with a tunable variable*** was added to describe how soon and how much of people in "E" could become "I", and how people in "E" could affect neighbors become "E" in a similar fashion like people in "I" behave. And an assumption made that the recovered individual will have total immune to the COVID-19. Thus, a modified SEIR model addressing the asymptoms virus shedding was created.



1. **Infectious Pattern** of Modified SEIR model in CA framework:
   A cellular Automata method was applied, to model the spread, putting one person in each grid cell of a $n \times n$ matrix, the four surrounding neighbor cells representing the persons in close contact. Each person has {S,I,E,R} four possible states.



Reference:

[1] Lauer SA, Grantz KH, Bi Q, et al. The Incubation Period of Coronavirus Disease 2019 (COVID-19) From Publicly Reported Confirmed Cases: Estimation and Application. Ann Intern Med. 2020; [Epub ahead of print 10 March 2020]. doi: https://doi.org/10.7326/M20-0504 (https://doi.org/10.7326/M20-0504)

[2] Rothe, C. et al. Transmission of 2019-nCoV infection from an asymptomatic contact in germany. N. Engl. J. Med. https://doi.org/10.1056/NEJMc2001468 (https://doi.org/10.1056/NEJMc2001468) (2020).

[3] Zou, Lirong, et al. "SARS-CoV-2 viral load in upper respiratory specimens of infected patients." New England Journal of Medicine 382.12 (2020): 1177-1179.

2. Tutorial - Build the CA framework modified SEIR model

2.1 CA framework utilities

For the celluar automata model, each cell within the NxN grid represents a person. Each person could have one of the S,E,I,R status.

When creating the NxN grid framework, we denote an outer boundary outside the framework, using EMPTY, value -1 to represent.

For all the cells within the boundary are the total populations. For example, for N = 100, a 100 x 100 grid will represent a population of 10000 people. For each person, we define the up, down, left, right side connecting cells as close contact neighbors. The neighbor cells are the cells that could be affected by the person's status.

Based on our literature review, and some trials, we estimate the transition probability for status transition. We also set a days count limit as the upper bound especially for EXPOSED and INFECTIOUS status. Meaning that if for 14 consecutive days being EXPOSED, at 15th day, the person will either be recovered or showing syptoms (classified as INFECTIOUS). But it is worth to notice that INFECTIOUS status meaning showing syptoms and can be infectious. Before a person become INFECTIOUS, the person can also be infectious when the person is in EXPOSED status. This is the main extension part of this SEIR model.

For people becoming INFECTIOUS, the day count for this person will start counting the days he/she is being INFECTIOUS. The upper limit for being INFECTIOUS is set to be 28 days, meaning after 28 days in INFECTIOUS status, a person can either be dead or fully recovered, both are classified as RECOVERED in the model.

We use this days counting thresh method to prevent people staying at 1 status forever, this could happen because we use generate random probability for each person each day. And at the same time, this days threshold method can describe the asyptoms and syptoms infections. Asytoms typically have less days to recover, but can be infectious for the first 14 days. Syptoms can be infectious during the EXPOSED period, but can also be infectious when showing syptoms becoming INFECTIOUS period. The day count thresh method can effectively find out which cell should change phase, which may continue.

For a general concern, we place the "patient zero" at the very center of the population.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


### CA framework utilities
EMPTY = -1
SUSCEPTIBLE = 0
EXPOSED = 1
INFECTED = 2
RECOVERED = 3

### Probability Setting
PROB_Exposure = 0.90 #conditional probability of getting sick, given any sick neighbors
PROB_Exposure2Recovered = 0.8
PROB_Exposure2Infectious = 1- PROB_Exposure2Recovered

### Days counter threshold
E_time_thresh = 14 # days
I_time_thresh = 28 # days

### random seed (for debugging purpose)
np.random.seed (1602034230) #


PROB_Recover = 0.96

def count (G):
    """
    Counts the number of locations in a NumPy array, `G`,
    where `np.where (G)` holds.
    """
    return len (np.where (G)[0])

def find (G):
    """
    Returns the set of locations of a NumPy array, `G`,
    where `np.where (G)` holds.
    """
    assert type (G) is np.ndarray
    return {(i, j) for i, j in zip (*np.where (G))}



#### Initial a world of nxn grid, each box contain one people. Initially dropped an infected one in the c
enter of the crowd.
def create_world(n):
    G = EMPTY * np.ones ((n+2, n+2), dtype=int)
    G[1:-1, 1:-1] = SUSCEPTIBLE
    i_mid = int ((n+2)/2)
    #All except the middle person is susceptible
    G[i_mid, i_mid] = INFECTED
    return G


def plot_initial (G, vmin=EMPTY, vmax=RECOVERED, values="states"):
    """A helper routine to visualize a 2-D world."""
    # Set color range
    assert values in ["states", "bool"]
    if values == "states":
        vticks = range (vmin, vmax+1)
        vlabels = ['Empty', 'Susceptible','Exposed','Infected','Recovered']
    else:
        vticks = [0, 1]
        vlabels = ['False (0)', 'True (1)']

    m, n = G.shape[0]-2, G.shape[1]-2
    plt.pcolor (G, vmin=vmin, vmax=vmax, edgecolor='black')
    cb = plt.colorbar ()
    cb.set_ticks (vticks)
    cb.set_ticklabels (vlabels)
    plt.axis ('square')
    plt.axis ([0, m+2, 0, n+2])
    return

## detect functions
def susceptible (G):
    return (G == SUSCEPTIBLE).astype (int)


def infected (G):
    return (G==INFECTED).astype (int)

def exposed (G):
    return (G==EXPOSED).astype (int)

def exposing_index (G):
    """
```

```
    Returns a grid whose (i, j) entry is 1 if it has
    at least 1 infected neighbor, or 0 otherwise.
    """
    E = np.zeros (G.shape, dtype=int) # exposed people
    ### BEGIN SOLUTION
    I = infected (G)+exposed(G)
    E[1:-1, 1:-1] = I[0:-2, 1:-1] | I[1:-1, 2:] | I[2:, 1:-1] | I[1:-1, 0:-2]
    ### END SOLUTION
    return E

# Visualizes your results:

def recovered (G):
    return (G==RECOVERED).astype (int)

def days_count_initial(G):
    ###
    D = np.zeros(G.shape,dtype = int)
    return D


# define a asymptoms matrix to identify whom will show no symptoms if exposed
def asymptoms_matrix(G,recover = PROB_Exposure2Recovered):
    random = np.random.uniform(size=G.shape)
    G_a = (random<recover)
    return G_a.astype(int)

def expired_day(G_asymptoms,E_time_thresh, I_time_thresh):
    # day count probability between 0-1
    random = np.random.uniform(size=G_asymptoms.shape)
    # return the day counts distribution for E could be I and E could be R
    G_symptoms = np.round((G_asymptoms<1)*random*E_time_thresh)
    G_asymptoms = np.round(G_asymptoms*(random*(I_time_thresh)+E_time_thresh))
    G_days = G_symptoms+G_asymptoms
    return G_days

def recovered_day(G_asymptoms,E_time_thresh,I_time_thresh):
    random = np.random.uniform(size=G_asymptoms.shape)
    G_infect2recov = np.round((G_asymptoms<1)*random*I_time_thresh)
    return G_infect2recov


def spreads (G,G_asymptoms_index,G_days_change,G_infect2recov,D,tau=PROB_Exposure,theta=PROB_Recover, E_t
ime = E_time_thresh, I_time=I_time_thresh):
    #### update counter, new iteration update now, all start at 0 at iteration = 0. existing E and I +=1.
    D_update = D+infected(G)+exposed(G)


    ### find the getting exposed surroundings of infected, including being exposed by I.
    random_draw_s2e = np.random.uniform (size=G.shape)
    random_draw_i2r = np.random.uniform (size=G.shape)

    G_e = (susceptible (G) * exposing_index (G) * (random_draw_s2e < tau))
    # for the pecent that turned to I from E
    G_i = exposed(G)*(G_asymptoms_index<1)*(G_days_change==D_update)
    G_r_from_e = exposed(G)*G_asymptoms_index*(G_days_change==D_update)
    G_r_from_i = infected(G)*((G_infect2recov+G_days_change)==D_update)
    D_final =D_update - D_update*G_e

    G = G-G*(G_e+G_i+G_r_from_e+G_r_from_i)+G_e*EXPOSED+G_i*INFECTED+(G_r_from_e+G_r_from_i)*RECOVERED
    confirmed = np.sum(G_i)
    return G, D_final,confirmed
```

2.2 Simulate over time steps (days) utilities functions

In this block, two utilities functions are provided, one is to save each time step plot frame while getting the data, the other one is to get the data without plotting and saving figures. The output figures are found at "ca_outputs" folder, we intentionaly ignored this outputs in our github repo, as they take up too much space, but they can be generated locally.

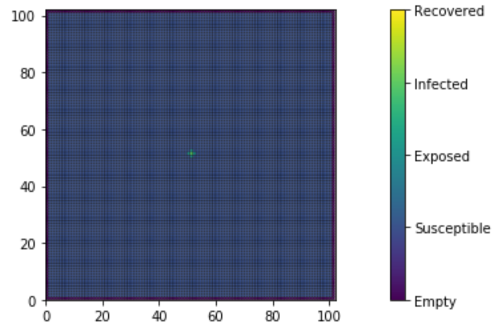
In [2]:
```
def simulate_ca_modified_seir(max_iteration,initial_test,D):
    container = {"S":[],"E":[],"I":[],"R":[],"Confirmed":[]}
    container["S"].append(np.sum(susceptible(initial_test)))
    container["E"].append(np.sum(exposed(initial_test)))
    container["I"].append(np.sum(infected(initial_test)))
    container["R"].append(np.sum(recovered(initial_test)))
    container["Confirmed"].append(np.sum(infected(initial_test)))

    population = np.sum(container.values())
    for i in range(max_iteration):
        initial_test, D,confirmed = spreads(initial_test, G_asymptoms_index,G_days_change,G_infect2recov,
D)
        container["S"].append(np.sum(susceptible(initial_test)))
        container["E"].append(np.sum(exposed(initial_test)))
        container["I"].append(np.sum(infected(initial_test)))
        recovered_total = np.sum(recovered(initial_test))
        container["R"].append(recovered_total)
        container["Confirmed"].append(confirmed)

        fig = plt.figure ()
        plot_initial(initial_test)
        plt.savefig('./ca_outputs/test_'+str(i)+'.png',dpi=600)
        if recovered_total == population:
            return initial_test, D,container
    return initial_test, D,container


def simulate_ca_modified_seir_without_saving_plots(max_iteration,initial_test,D):
    container = {"S":[],"E":[],"I":[],"R":[],"Confirmed":[]}
    container["S"].append(np.sum(susceptible(initial_test)))
    container["E"].append(np.sum(exposed(initial_test)))
    container["I"].append(np.sum(infected(initial_test)))
    container["R"].append(np.sum(recovered(initial_test)))
    container["Confirmed"].append(np.sum(infected(initial_test)))

    population = np.sum(container.values())
    for i in range(max_iteration):
        initial_test, D,confirmed = spreads(initial_test, G_asymptoms_index,G_days_change,G_infect2recov,
D)
        container["S"].append(np.sum(susceptible(initial_test)))
        container["E"].append(np.sum(exposed(initial_test)))
        container["I"].append(np.sum(infected(initial_test)))
        recovered_total = np.sum(recovered(initial_test))
        container["R"].append(recovered_total)
        container["Confirmed"].append(confirmed)
        if recovered_total == population:
            return initial_test, D,container
    return initial_test, D,container
```

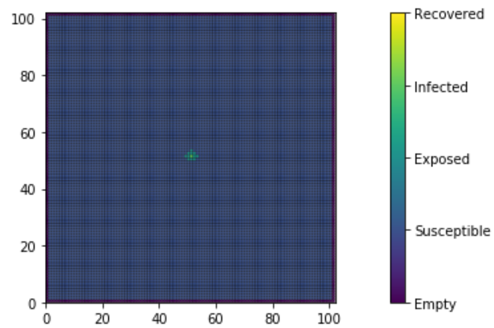
2.3 Simulate one run, with 100x100 ca model, max iteration thresh at 30.

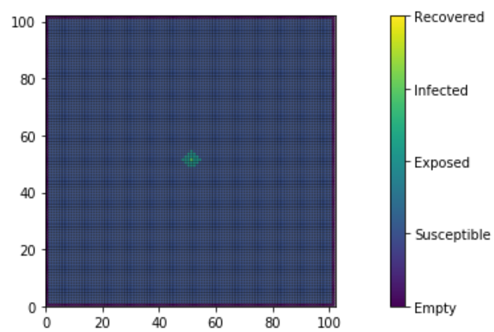
Due to the limit of the pages, I will use a 30 max iterations for 100x100 CA model. Normally a 100x100 will stop around 120 days when all people are recovered. Later section will explore that part further in a statistical way.

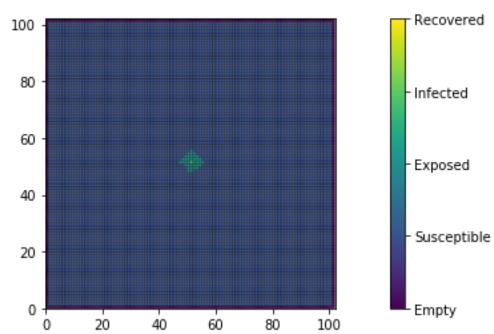
In [3]:
```
### Reclaim variables values setting for your convienience for later change.

### CA framework utilities
EMPTY = -1
SUSCEPTIBLE = 0
EXPOSED = 1
INFECTED = 2
RECOVERED = 3

### Probability Setting
PROB_Exposure = 0.90 #conditional probability of getting sick, given any sick neighbors
PROB_Exposure2Recovered = 0.8
PROB_Exposure2Infectious = 1- PROB_Exposure2Recovered

### Days counter threshold
E_time_thresh = 14 # days
I_time_thresh = 28 # days

### random seed (for debugging purpose)
np.random.seed (1602034230) #


PROB_Recover = 0.96


### N describe the population, NxN is the total population in CA framework,
### MAX_iteration holds the thresh iteration times to stop the simulation
N = 100
MAX_iteration = 30

scale_test = create_world (N)
# plot_initial (scale_test)
D_counter = days_count_initial(scale_test)
G_asymptoms_index = asymptoms_matrix(scale_test)
G_days_change = expired_day(G_asymptoms_index, E_time_thresh,I_time_thresh)
G_infect2recov = recovered_day(G_asymptoms_index,E_time_thresh,I_time_thresh)

result_test, counter,container = simulate_ca_modified_seir(MAX_iteration,scale_test,D_counter)
```

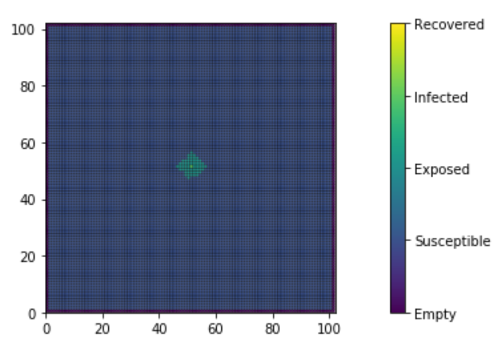
```
/home/muyangguo/.local/lib/python2.7/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20
figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are re
tained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```

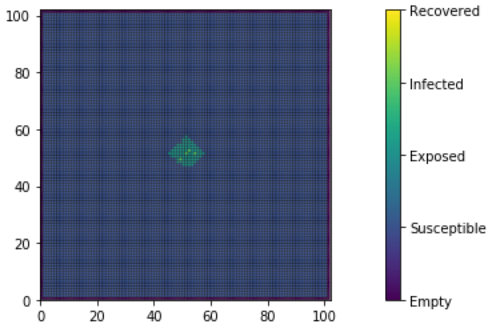


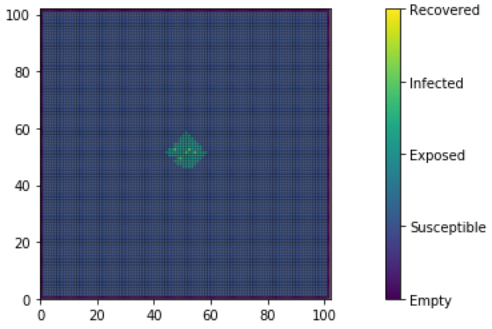


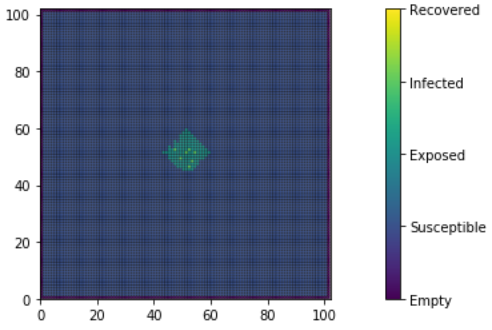


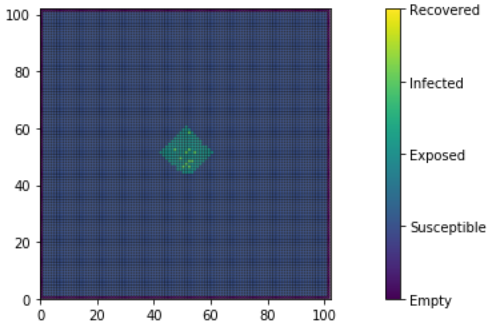


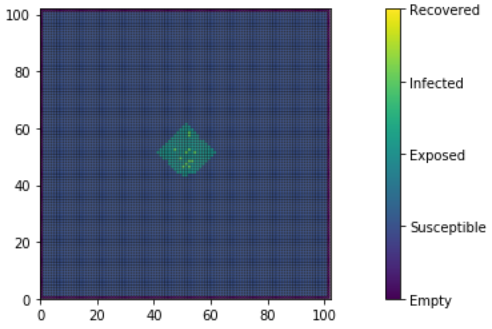


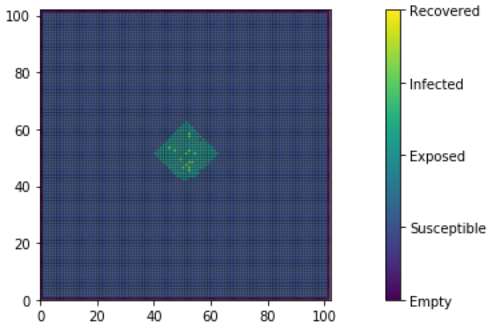

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

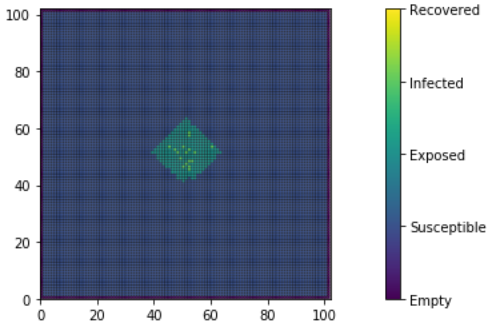

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

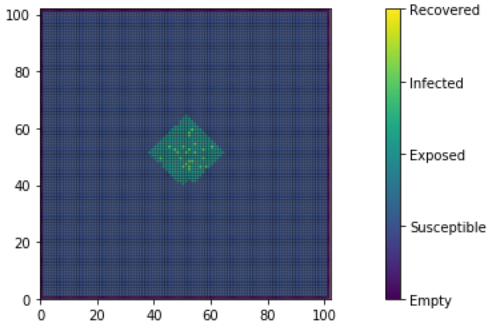

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

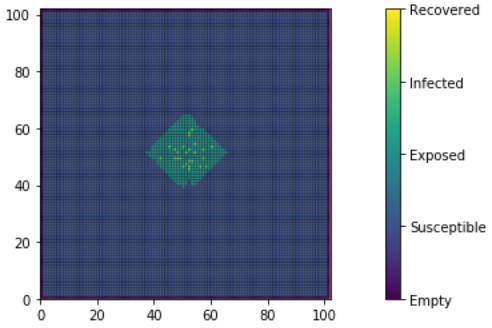

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

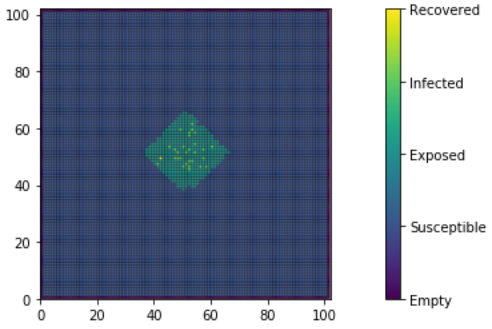

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

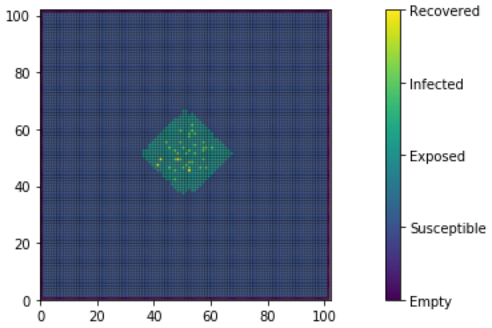

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

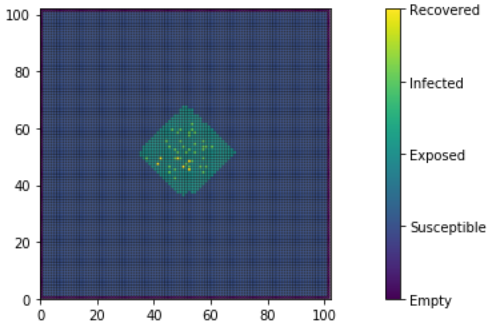

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

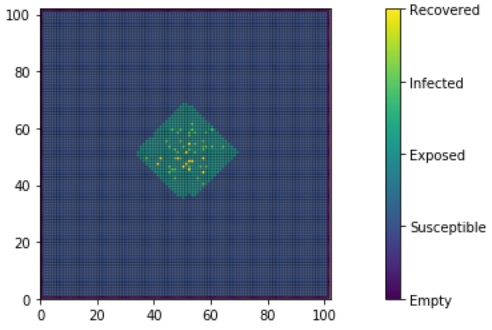

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

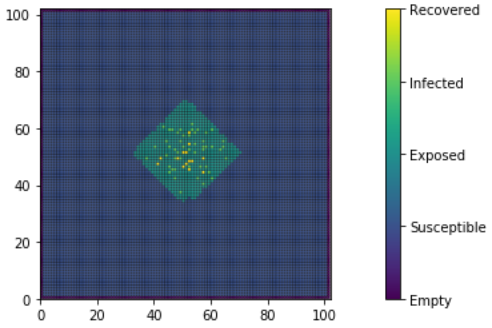

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

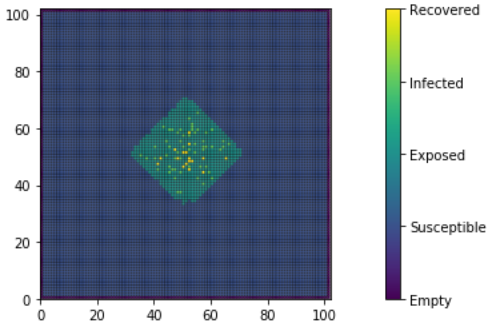

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

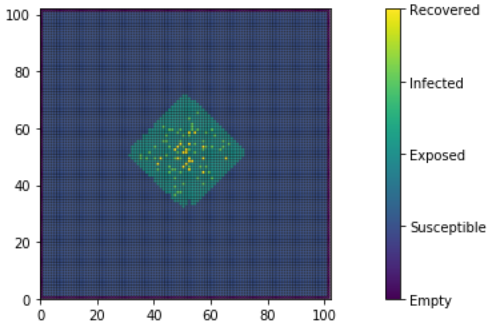

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

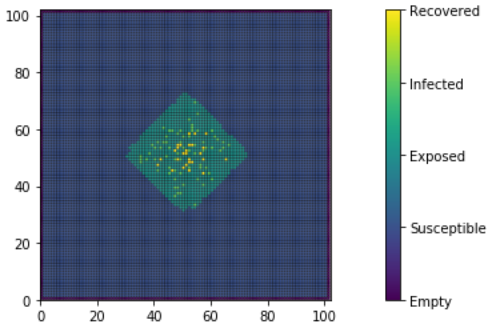

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

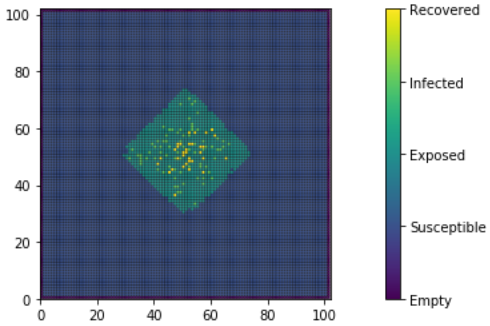

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

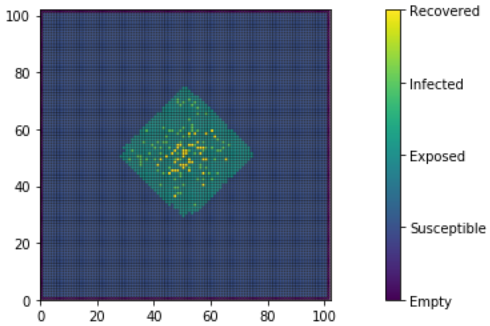

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

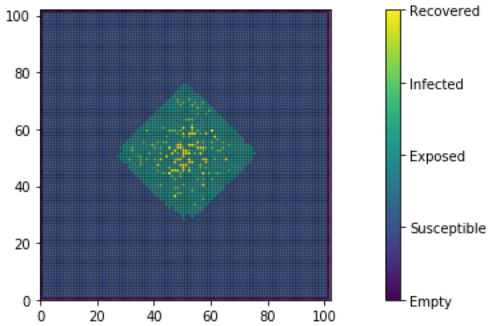

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

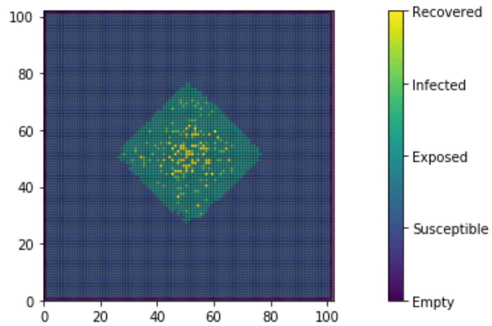

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

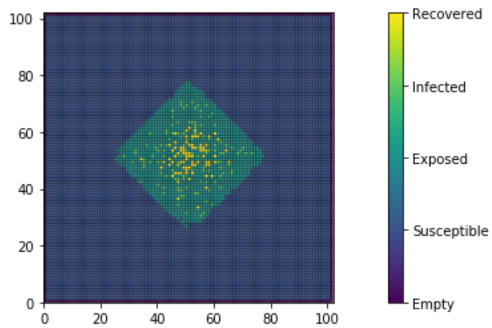

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

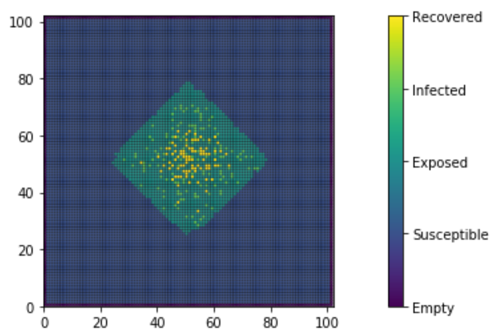

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

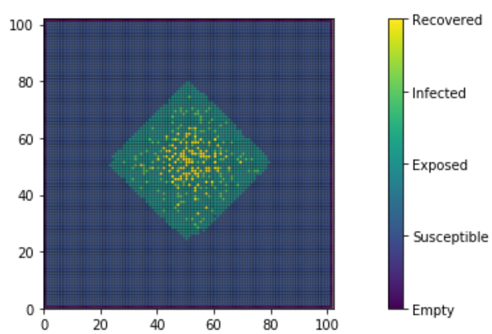

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100

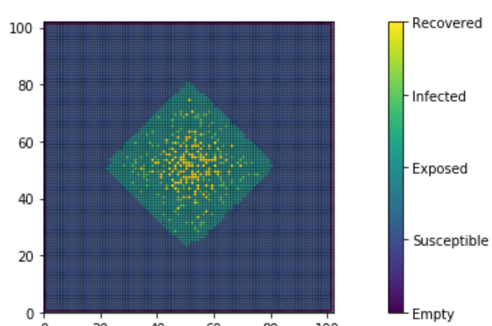

Recovered
Infected
Exposed
Susceptible
Empty
0
20
40
60
80
100


0
20
40
60
80
100
Recovered
Infected
Exposed
Susceptible
Empty


0
20
40
60
80
100
Recovered
Infected
Exposed
Susceptible
Empty


0
20
40
60
80
100
Recovered
Infected
Exposed
Susceptible
Empty


0
20
40
60
80
100
Recovered
Infected
Exposed
Susceptible
Empty


0
20
40
60
80
100
Recovered
Infected
Exposed
Susceptible
Empty

2.4 Simulate multiple runs, with 100x100 ca model, max iteration thresh at 100.

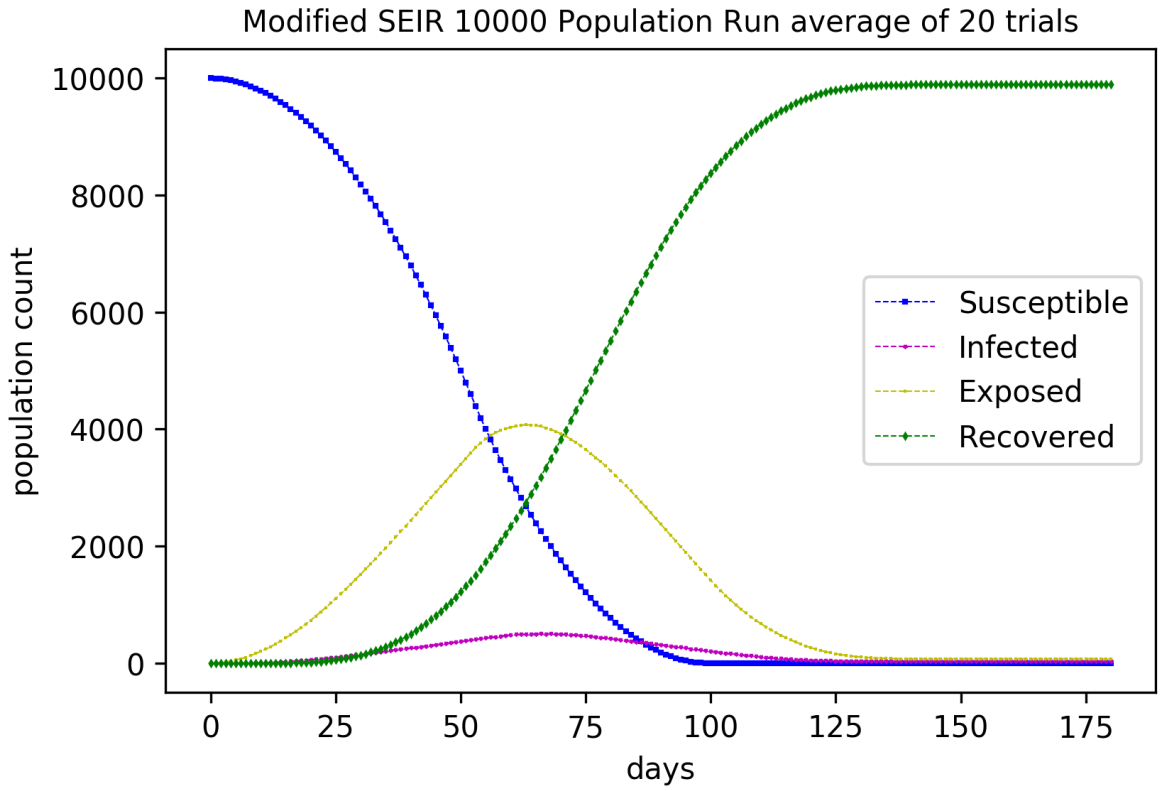
Depending on memoty limitation, multiple runs with large ca can be time consuming. In this example, we run 20 times and average the results of total runs. To save time and memory, we use the simulate function without savign figures for each frame. The parameters setting inheritaged from the previous single run trial. The data of the SEIR daily count averaged are plotted over time.

This time, we chose a MAX_Iteration of 180 days, in order to see how many days for the population to become fully recovered, and what will be the peak percentage, and when will that happen.

In [4]:
```
#### RUN Multiple times and Plot average


RUN_TIMES = 20
N = 100
MAX_iteration = 180

container = {"S":[],"E":[],"I":[],"R":[],"Confirmed":[]}
for i in range(RUN_TIMES):

    scale_test = create_world (N)
    single_container = {"S":[],"E":[],"I":[],"R":[],"Confirmed":[]}
    single_container["S"].append(np.sum(susceptible(scale_test)))
    single_container["E"].append(np.sum(exposed(scale_test)))
    single_container["I"].append(np.sum(infected(scale_test)))
    single_container["R"].append(np.sum(recovered(scale_test)))
    single_container["Confirmed"].append(np.sum(infected(scale_test)))

    D_counter = days_count_initial(scale_test)
    G_asymptoms_index = asymptoms_matrix(scale_test)
    G_days_change = expired_day(G_asymptoms_index, E_time_thresh,I_time_thresh)
    G_infect2recov = recovered_day(G_asymptoms_index,E_time_thresh,I_time_thresh)

    result_test, counter,single_container = simulate_ca_modified_seir_without_saving_plots(MAX_iteration,
scale_test,D_counter)
    if i == 0:
        container = single_container
        print("Initilized with run 0")
    else:
        container["S"] = [a + b for a, b in zip(container["S"],single_container["S"])]

        container["E"] = [a + b for a, b in zip(container["E"],single_container["E"])]
        container["I"] = [a + b for a, b in zip(container["I"],single_container["I"])]
        container["R"] = [a + b for a, b in zip(container["R"],single_container["R"])]
        container["Confirmed"] =[a + b for a, b in zip(container["Confirmed"],single_container["Confirme
d"])]
        print("Done run time: ", i)


#### Plot the averaged data

container["S"] = [a/RUN_TIMES for a in container["S"]]
container["E"] = [a/RUN_TIMES for a in container["E"]]
container["I"] = [a/RUN_TIMES for a in container["I"]]
container["R"] = [a/RUN_TIMES for a in container["R"]]
container["Confirmed"] = [a/RUN_TIMES for a in container["Confirmed"]]


S = container["S"]
I = container["I"]
E = container["E"]
R = container["R"]
Confirmed = container["Confirmed"]

# function to find cumulative sum of array


def cumulativeSum(input):
    output = []
    for i in range(len(input)):
        prefixsum = sum(input[0:i+1])
        output.append(prefixsum)
    return output


Confirmed = cumulativeSum(Confirmed)

fig, ax = plt.subplots( dpi = 300)
ax.plot(range(len(S)),S,'bs--', label='Susceptible' ,linewidth=0.5,markersize=1)
ax.plot(range(len(S)),I,'m*--', label='Infected',linewidth=0.5,markersize=1)
ax.plot(range(len(S)),E,'y+--', label='Exposed',linewidth=0.5,markersize=1)
ax.plot(range(len(S)),R,'gd--', label='Recovered',linewidth=0.5,markersize=1)
ax.set_title("Modified SEIR 10000 Population Run average of "+str(RUN_TIMES)+" trials",fontsize = 10)
plt.xlabel('days', fontsize=10)
plt.ylabel('population count', fontsize=10)


# ax.plot(range(len(S)),Confirmed,'r', label='Confirmed Cases')
ax.legend()
```

```
Initilized with run 0
('Done run time: ', 1)
('Done run time: ', 2)
('Done run time: ', 3)
('Done run time: ', 4)
('Done run time: ', 5)
('Done run time: ', 6)
('Done run time: ', 7)
('Done run time: ', 8)
('Done run time: ', 9)
('Done run time: ', 10)
('Done run time: ', 11)
('Done run time: ', 12)
('Done run time: ', 13)
('Done run time: ', 14)
('Done run time: ', 15)
('Done run time: ', 16)
('Done run time: ', 17)
('Done run time: ', 18)
('Done run time: ', 19)
```

Out[4]: <matplotlib.legend.Legend at 0x7fb82a6dcd10>



Discussion

As can been seen from the plot, the model with the current setting predict that around 120 days, all population will be in RECOVERED status. The Peak EXPOSURE percentage will happen at around 60 days. And the EXPOSED rate could be as high as 40% at that peak. The Infected peak is also at that time. The EXPOSED cases are more than the INFECTED cases, this is because we have considered the asymptoms populations. They did not show any symptoms yet could be infectious as well. And we can see RECOVERED population soon become larger than the INFECTED cases, it is because so many people are EXPOSED but being asyptoms and over some days they become RECOVERED directly. Only a fraction of the EXPOSED actually showed syptoms and become INFECTED status. This plot fits all of our expectations.

3. Real Data Acquisitions, Processing, Validating

In this section, we will first pull the latest public data, and gain a general understanding of the real data curves, with some basic statistics. We will most use the national death rate to estimate our death rate in our projection model. We also will use real data to align with our scaled projection data. For example, the CA model we use predict a 10000 population outbreak. If scaled to a NEW YORK population level, at about 8.623 million people, we need to scale our projection for 862.3 times. Thus the first and only one infected person at day 0 in the CA frame will be scaled to 862 people. Then we tried to find the a day from NEW YORK 's accumulated daily confirmed cases data around that 862 number, and align our projection's starting date as that date.

3.1 Data Source:

Coronavirus COVID-19 Global Cases by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University (JHU)

- Blog (https://systems.jhu.edu/research/public-health/ncov/)
- Dashboard (https://www.arcgis.com/apps/opsdashboard/index.html#/bda7594740fd40299423467b48e9ecf6)

3.2 Data used:

CSSEGISandData (https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_time_series)

1. time_series_covid19_confirmed_US.csv
3. time_series_covid19_deaths_US.csv

3.3 Data Pre-processing, basic statistics and insights:

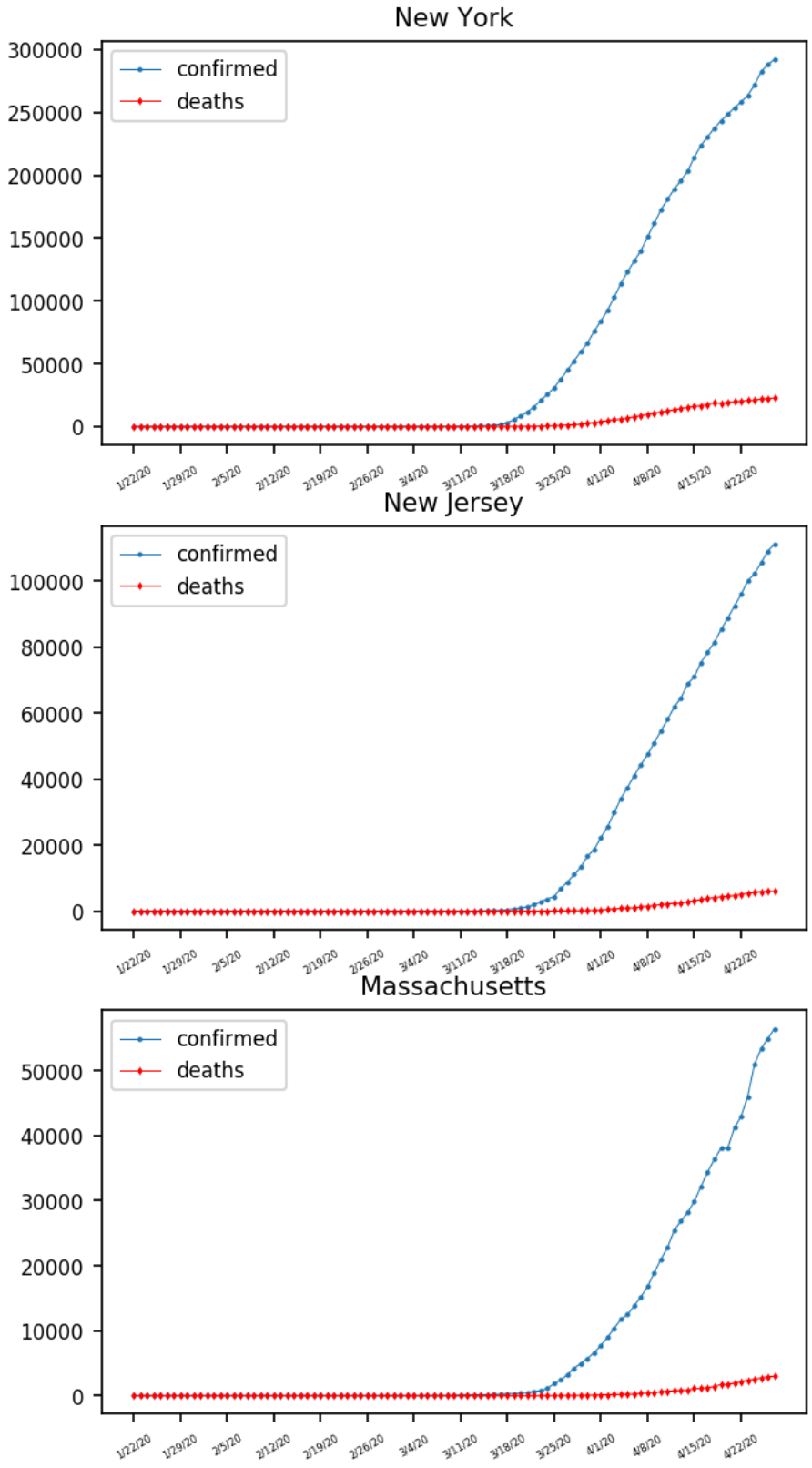
In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


def preprocess_US(data_url):
    df = pd.read_csv(data_url)
    last_update_date = df.keys()[-1]
    df.drop(['UID','iso2','iso3','code3','FIPS','Admin2','Lat','Long_'], axis = 1, inplace = True)
    df_cleaned= df.groupby(['Province_State'], as_index = False).sum()
    df_sorted = df_cleaned.sort_values(by=last_update_date,ascending=False).reset_index(drop=True)
    update_total = sum(df_sorted[last_update_date])
    return df_sorted, last_update_date, update_total
```

In [6]:
```
### Retrieve data from CSSEGISandData Github Raw content in csv
url_confirmed_US = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_US.csv'
url_deaths_US = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_deaths_US.csv'



### Get useful insights of the updated data (data is updated daily)
df_confirmed_US,last_confirmed_US,total_confirmed_US = preprocess_US(url_confirmed_US)
df_deaths_US,last_deaths_US,total_deaths_US = preprocess_US(url_deaths_US)
first_date_US = df_confirmed_US.keys()[1]
print("Data Basic Statistics: \n")
print('From {} till {} the US confirmed cases total is : {} \n'.format(first_date_US,last_confirmed_US,total_confirmed_US))
print('From {} till {} the US deaths cases total is : {} \n'.format(first_date_US,last_deaths_US,total_deaths_US))
mortality_rate = float(total_deaths_US)/float(total_confirmed_US)*100
print('Mortality rate in US is being updated as {:.3f}% \n'.format(mortality_rate))
US_population_list = df_deaths_US['Population']
US_population = US_population_list.sum()
df_deaths_US.drop(['Population'], axis = 1, inplace = True)
print('COVID-19 is among US population of {:.5f}% and fatal rate is {:.5f}% \n'.format(float(total_confirmed_US)/float(US_population)*100, float(total_deaths_US)/float(US_population)*100))
```

```
Data Basic Statistics:

From 1/22/20 till 4/27/20 the US confirmed cases total is : 988197

From 1/22/20 till 4/27/20 the US deaths cases total is : 56259

Mortality rate in US is being updated as 5.693%

COVID-19 is among US population of 0.29391% and fatal rate is 0.01673%
```

In [7]:

```
### Plot the top severe states daily increasing cases and death plots

def get_top_rank(sorted_df,compare_df,rank):
    # Create an empty list
    container = {}
    rows_list = []
    rows_compare_list=[]
    state_sorted_order = []
    # Iterate over each row
    for index, row in sorted_df.iterrows():
        if index <rank:
            row_list = row.to_list()
            container[row_list[0]] = [row_list[1:]]
            state_sorted_order.append(row_list[0])
    for index_compare, row_compare in compare_df.iterrows():
        row_compare_list = row_compare.to_list()
        if row_compare_list[0] in container:
            container[row_compare_list[0]].append(row_compare_list[1:])
    return container, state_sorted_order


### Let us plot the top 3 severe states to have a general idea of the situation
container, state_list = get_top_rank(df_confirmed_US,df_deaths_US,3)
fig, ax = plt.subplots(ncols = 1, nrows = len(state_list), figsize = (5,10), dpi = 150)
date = df_confirmed_US.keys()[1:]

for i in range(len(state_list)):
    ax[i].plot(date,container[state_list[i]][0],'o-',label="confirmed", linewidth=0.5,markersize=1)
    ax[i].plot(date,container[state_list[i]][1],'rd-',label= "deaths", linewidth=0.5,markersize=1)
    ax[i].set_title(state_list[i],fontsize = 10)
    ax[i].tick_params(axis='x',labelsize=4)
    ax[i].tick_params(axis='y',labelsize=8)
    ax[i].legend(fontsize = 8)
    ax[i].set_xticks(date[::7])
    ax[i].set_xticklabels(date[::7], rotation=30)
```



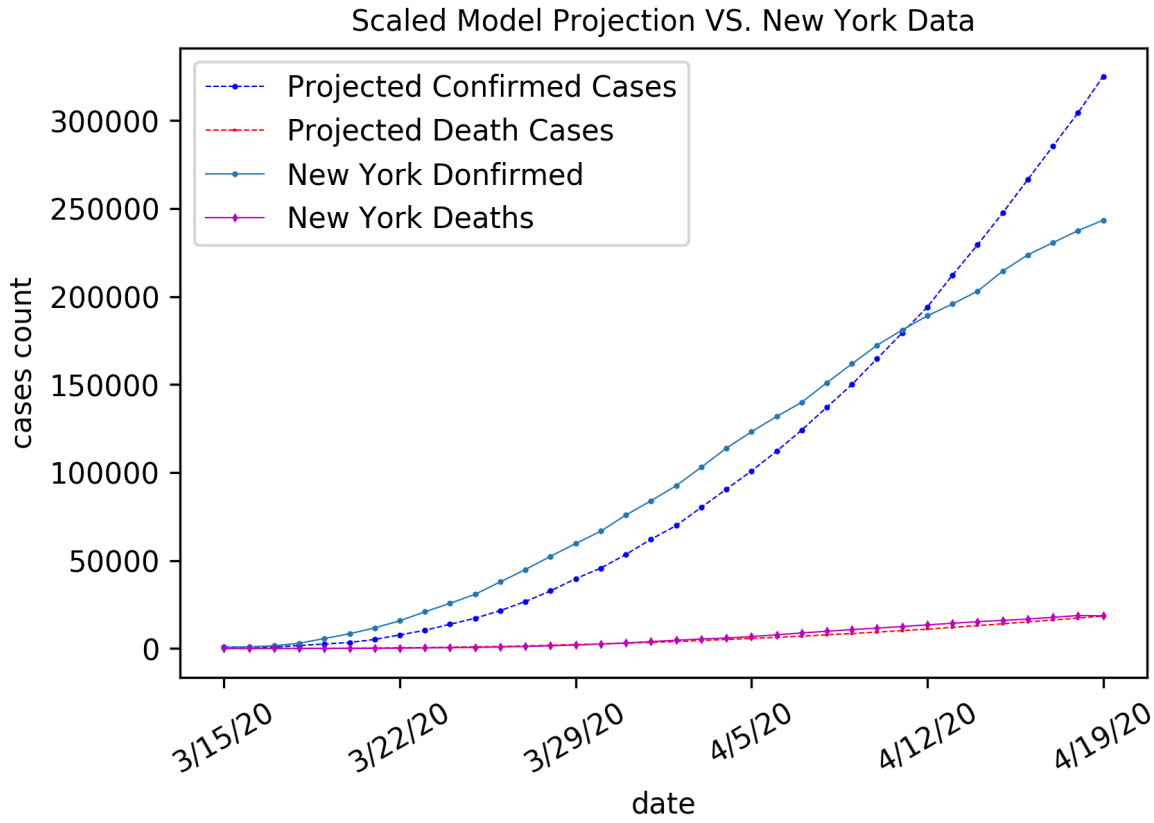
4 Validation

4.1 Compairison between selected REAL data and Simulated data

In this practice, we could used the averaged simulated data after multiple runs, previously stored in the "container" DICT, to compare with the real world data. The real data we selected is the NEW YORK accumulated confirmed data, and the Death data. The whole country scaled morality rate, is the DEATH_RATE that we used to predict the deaths from our SEIR model, since the SEIR model we had does not specifically indicate the DEATH portion.

In [8]:
```
### Example Pick New York States as an example, New York is known to have 8.623 million people, and we co
uld use
### a daily updated retrieved morality rate as 5.683% (April 27th)

POPULATION = 862.3
DEATH_RATE = 0.05683


fig2, ax2 = plt.subplots( dpi = 300)
ax2.plot(range(len(Confirmed[4:40])),[POPULATION
*x for x in Confirmed[4:40]],'bo--',label='Projected Confirmed Cases',linewidth=0.5,markersize=1)

Death = np.round([862.3
*float(i)*DEATH_RATE for i in Confirmed[4:40] ])
ax2.plot(range(len(Death)),Death,'r+--',label='Projected Death Cases',linewidth=0.5,markersize=1)

container_top1, state_list_top1 = get_top_rank(df_confirmed_US,df_deaths_US,1)

check_len = len(Death)

date_ny = df_confirmed_US.keys()[1:]

date_ny = date_ny[53:(53+(check_len))]

Newyork_confirmed = container_top1["New York"][0][53:(53+(check_len))]
Newyork_death = container_top1["New York"][1][53:(53+(check_len))]
ax2.plot(date_ny,Newyork_confirmed,'o-',label="New York Donfirmed", linewidth=0.5,markersize=1)
ax2.plot(date_ny,Newyork_death,'md-',label= "New York Deaths", linewidth=0.5,markersize=1)
ax2.set_xticks(date_ny[::7])
ax2.set_xticklabels(date_ny[::7], rotation=30)
ax2.legend()
ax2.set_title("Scaled Model Projection VS. New York Data",fontsize = 10)
plt.ylabel('cases count', fontsize=10)
plt.xlabel('date', fontsize=10)
```

Out[8]: Text(0.5,0,'date')



Discussion

As can be seen from the plots, after the scaling, and aligning the projection data with NEW YOPK's real data. The projected death curve and the NEW YORK death curve aligned well, and there is some difference between the projected confirmed cases, and confirmed cases. Our syptoms INFECTED curve seemed more steep than the real data, this could be due to the fact NEW YORK has taken measures to intervene the spread, yet our model does not count the intervene measures into consideration.

However,as our model has taken into account of the fact that the asyptoms people can infect others (assuming EXPOSED 80% could be asyptoms). There are days at the first 30 days that our projected infected data is below the real infected data curve, implying the fact that there should exist a quite large number of people being exposed yet not showing syptoms and just become recovered over some days in real world.

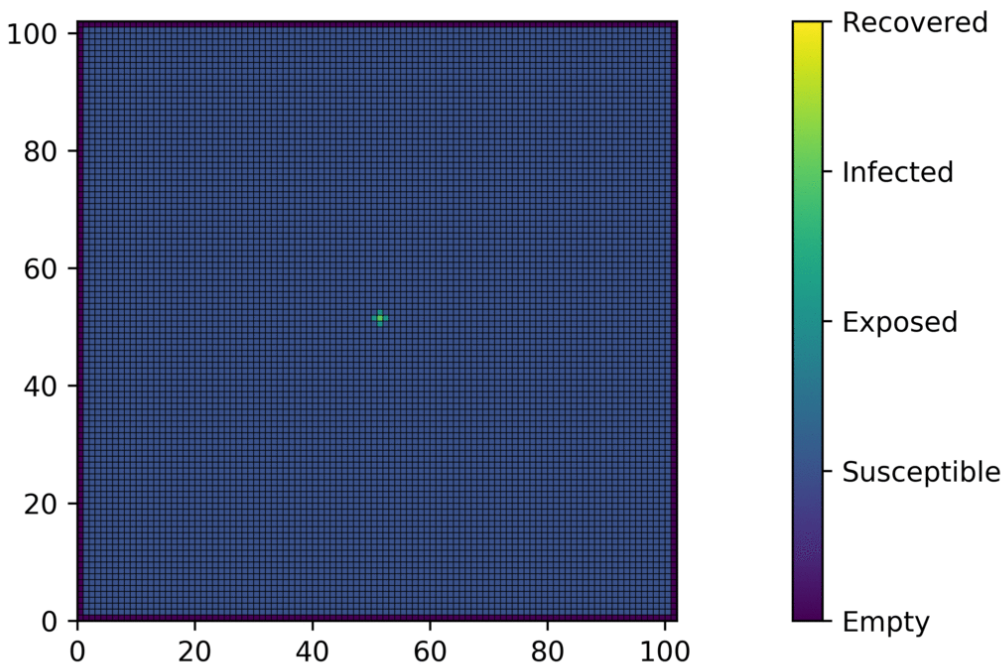
5 Visualization (Due to the HTML content, the gif animation might not be able to be shown on pdf report, please go to github repo, or open the jupyter notebook)

In this case, we showed some of animations for single trials. We stored the frames plot locally but not included in the github repo, so here are some processed animations as visualizations.

Clearly we could see that the EXPOSED status spread very quickly, but not all of the EXPOSED people become a confirmed case -- INFECTED status. Those without any symptoms will just become RECOVERED.

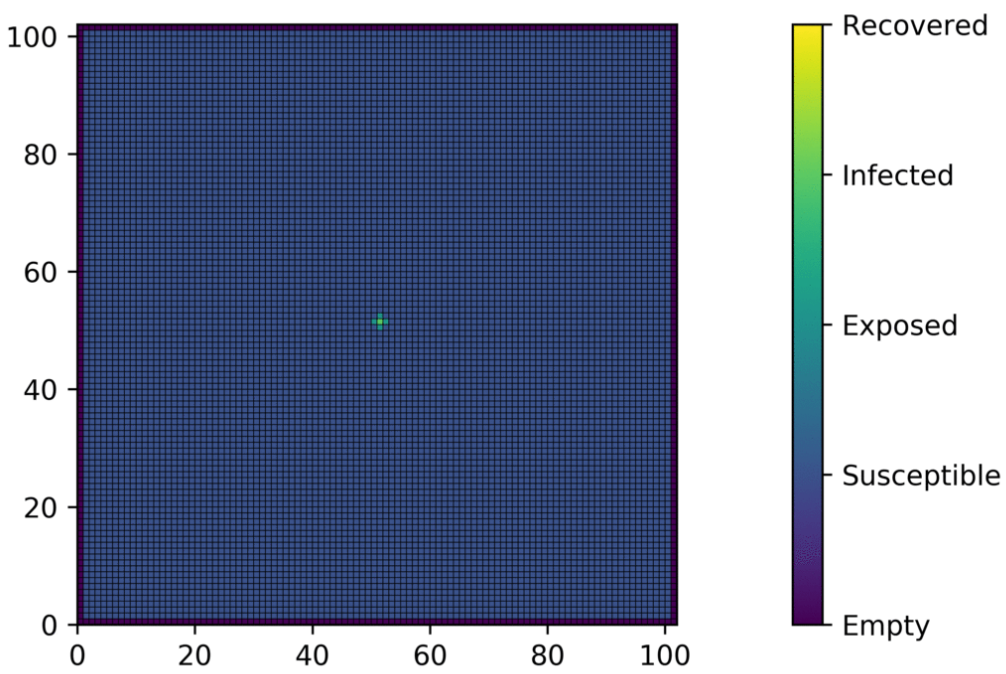
In [9]:
```
from IPython.display import HTML
HTML('<img src="https://raw.githubusercontent.com/MUYANGGUO/COVID19-Simulation/master/Models_Notebooks/ca_visuals/ca-modified-SEIR-10000.gif">')
```

Out[9]:



In [10]:
```
HTML('<img src="https://raw.githubusercontent.com/MUYANGGUO/COVID19-Simulation/master/Models_Notebooks/ca_visuals/ca-modified-SEIR-10000-version2.gif">')
```

Out[10]:



In [ ]:

Part 3: Markov Chain Analysis Based on One Dimensional CA model:

Table of Contents

1. Introduction of S-E-I-R Model

SEIR model is a common simulation model in computer simulation of epidemics. This SEIR models people in four states, which are susceptible, exposed, infected, and resistant. People will have their own state at each unit time and each state also represents the amount of people in this state.

In the study of COVID-19 using SEIR model and Markov Chain, there are assumptions. The first one is that people who recoverd will be immutable. The second assumption is death and birth rate are not considered in this model.

S - the total amount of susceptible people
E - the total amount of exposed people
I - the total amount of infected people
R - the total amount of recovered people

The mathmatical expression of SEIR model is described below.

$$\frac{dS}{dt} = -\frac{\beta S I}{N}$$
$$\frac{dE}{dt} = \frac{\beta S I}{N} - \alpha E$$
$$\frac{dI}{dt} = \alpha E - \lambda I$$
$$\frac{dR}{dt} = \lambda I$$
$$N = S + E + I + R$$

SEIR represents people's different states. S represents people who are able to contract with the disease. E represents people who are infected but not yet infectious. I represents people who are infected and infectious. R represents people who are recovered. N represents the total amount of people. β represents the probability that a suscepptible person is infected by a infected person. α represents the probability that a exposed person becomes infectious. λ represents the probability that a infected person recovered.

$$P_j = \sum_{i=0} j = nP_i * Pij$$

2. Introduction of Markov Chain Model

In order to implement and analyse the SEIR model, Markov Chain method is implemented. In this section, COVID-19 progression is assumed to follow a discrete time Markov chain with stationary transition probabilities[1], which is represented as: $X(t), t \in T$ with time index set $T = 0, 1, \ldots$ and a finite state space $S = S_1, S_1, \ldots, S_N$. The Markov Chain model satisfies the rules that the probability of the random variable X being in the state S_i at time t depends on the X's state at time t-1.

In the study of COVID-19, $\sum$ is used to represents the space of all states. A 1-D system is built to represent the state in the Markov Chain Model. Each element in the array, which is of length n, represents a person and the value of the element represents the state of that person.

In the Markov Chain System, there are n people in the system and they are distributed in a one dimensional array. $\sum$ is the state space and $\sigma \in \sum$ represents a state. $\sigma = (g_1, g_2, \ldots, g_n m)$ and $g_i \in (S, E, I, R)$. S means that the person is susceptible. E represents the person has been exposed but not tested. I means that the person has been infected and tested positive. R means that the person has recovered. In the system, an n array are used to represent people. Each person in the matrix will has their own state, which could be susceptible, exposed, infected or recovered. Then the matrix that contains everyone's state is called a state of Markov Chain.

A sample state is given below. Assume there are 5 people and an array with length 5 is built.

$$\sigma = \begin{bmatrix} 0 & 1 & 2 & 1 & 3 \end{bmatrix}$$

$S = 0, E = 1, I = 2, R = 3.$

Reference:

[1] A. Vivanco-Lira, “Predicting covid-19 distribution in mexico through a discrete and time-dependent markov chain and an sir-like model,” 2020.

[2] Bendtsen Cano, S. Cano Morales, and C. Bendtsen, “Covid-19 modelling: the effects of socialdistancing,”medRxiv, 2020

2.1 Assumptions of Markov Chain Analysis

1. An infected person only infects his adjacent neighbours.

2. People who recovered from the disease will be immutable.

2.2 State in Markov Chain

For a 1-D system, there are n people and each person will have 4 posible states. Therefore, the states of Markov Chain model will be 4^n.

As you can see, the n people are in the one dimensional array. And the number of states limit the Markov Chain's scalability. This model is not applicable to the large scale problem.

In [257]:
```
import numpy as np
import scipy as sp
import scipy.sparse
import itertools
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import matplotlib.ticker as mtick
from matplotlib.ticker import PercentFormatter
import random
# define parameters
S = 0
E = 1
I = 2
R = 3
n = 4

#get possible values of a cell
values = list (range(S,R+1));
# Generate Sigma
sigma = set (itertools.product(values, repeat=n));

#print the basic information of this example
print ("There are ", n, " people in this case. They are distributed in a ",n," elements array.\n")
print ("There are ",len(sigma)," states.\n")

#transfer a matrix to a string
def convert_state_to_string (state):
    str1 = ""
    for ele in state:
        ele = int(ele)
        str1 += str(ele)
    return str1

#transfer a string to a matrix
def convert_string_to_state (str):
    return [int(x) for x in str if x.isdigit()]

#generate all the states and save them into a hash map
map = {}
index2str = {}
str2index = {}
i = 0
for state in sigma:
    map[convert_state_to_string(state)] = state
    index2str[i] = convert_state_to_string(state)
    str2index[convert_state_to_string(state)] = i
    i += 1
```

```
There are  4  people in this case. They are distributed in a  4  elements array.

There are  256  states.
```

2.3 Transition Matrix

The key of Markov Chain Model is the transition matrix and transition matrix is a square matrix. Each column/row is a state. In another word, the matrix will have 4^n rows and 4^n columns if there are n person in this system. Each element in the transition matrix represents the probability that a state goes to another state.

The limit of markov chain model is that the problem can not be scaled. In order to compute the transition matrix, the time complexity is $O(4^n)$, which is very expensive. Therefore, it can only be applied to small scale problem.

Let use A to denote the transition matrix.

$$A = \begin{bmatrix} \sigma_{1,1} & \sigma_{1,2} & \cdots & \sigma_{1,4^n} \\ \sigma_{2,1} & \sigma_{2,2} & \cdots & \sigma_{2,4^n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{4^n,1} & \sigma_{4^n,2} & \cdots & \sigma_{4^n,4^n} \end{bmatrix}$$

Based on the analysis of real world data, the parameters of the model are predefined. Since the Markov Chain Analysis is based on the one dimensional CA model, which is discussed in the Part2, most of the parameters are lined up with what are in the Part2. These predefined parameters are derived from the real world data.

The meaning of these parameters should be clarified. In our simulation, we define that the time unit of one state transition is one day. Therefore, the meaning of these parameters is the probability that a person changes from one state to another state in one day.

In [260]:
```
P_E2R = 0.40
P_E2I = 0.20
P_I2R = 0.90
P_SByE = 0.8
P_SByI = 0.9

Trans_Matrix = np.zeros(shape=(len(sigma),len(sigma)))

def checkE (i, src):
    count = 0
    if i != 0:
        if (src[i-1] == E):
            count += 1
    if i != n-1:
        if (src[i+1] == E):
            count += 1
    return count

def checkI (i, src):
    count = 0
    if i != 0:
        if (src[i-1] == I):
            count += 1
    if i != n-1:
        if (src[i+1] == I):
            count += 1
    return count


def get_prob(src, dest, i, prob):
    s = src[i]
    d = dest[i]
    ##check edge case
    if s == R and d != R:
        prob = 0
        return prob
    elif s == I and d < I:
        prob = 0
        return prob
    elif s == E and d < E:
        prob = 0
        return prob
    elif s == S and d == R:
        prob = 0
        return prob
    elif s == S and d == I:
        prob = 0
        return prob

    ##check neighbor
    countE = checkE(i,src)
    countI = checkI(i,src)
    if s == R:
        prob = prob * 1
    elif s == I and d == I:
        prob = prob * (1 - P_I2R)
    elif s == I and d == R:
        prob = prob * P_I2R
    elif s == E and d == E:
        prob = prob * (1 - P_E2R - P_E2I)
    elif s == E and d == I:
        prob = prob * P_E2I
    elif s == E and d == R:
        prob = prob * P_E2R
    elif s == S and d == E:
        prob = prob * (1-(1-P_SByE)**countE * (1-P_SByI)**countI)
    elif s == S and d == S:
        prob = prob * (1-P_SByE)**countE * (1-P_SByI)**countI

    return prob

#Given conditional probabilities, a transition matrix will be computed.
for sc in range(len(sigma)):
    for dc in range(len(sigma)):
        src = map[index2str[sc]]
        dest = map[index2str[dc]]
        prob = 1
        for i in range(len(src)):
            prob = get_prob(src, dest, i, prob)
        Trans_Matrix[sc,dc] = prob
```

3. Simulation

In this section, we will implement Markov Chain Analysis on the one dimensional case and there are n people in the one dimensional space. Because limit of the number of states and the size of transition matrix, the computation of our simulation is evey expensive. To avoid the scalability problem, we choose the scenario that only a small number of people are involved, like home or office.

Based on the transition matrix, the Markov Chain Simulation can be implemented. There are two main simulations we focused on. One is the simulation the change of number of current infected prople in the small community as time passes by. Another is to discusse the probability that all people get infected.

Initial State: One infected person will be chosen among the n people and will be randomly assign to a position. Steady State: After running many times, the simulation will reach a steady state.

The probability matrix at steady state means the probability of the markov chain state. Let use v denote the initial state, which is a 4^n vector. A is the transition matrix. S is probability vector at steady state.

$$S = vA^n$$

First, we need to generate the inital state randomly. We assume that there is only one exposed person in the community we studied. And the position of this person is randomly generated.

In [261]:
```
#generate initial state
poll = [0] * n
for i in range(len(poll)):
    poll[i] = i
initial_infected = random.choice(poll)
temp = [0] * n
temp[initial_infected] = 2
initial_state_index = str2index[convert_state_to_string(temp)]
v = np.zeros(shape=(len(sigma),1))
v[initial_state_index] = 1
res = v.transpose()
tres = v.transpose()
#res = v*Trans_Matrix
i = 0
while i < 80:
    tres = np.matmul(tres,Trans_Matrix)
    i += 1

print("Initial State: ",map[convert_state_to_string(temp)])
print("Steady State: ")
for i in range(len(tres[0])):
    print(" state: ",map[index2str[i]], " probability: ", tres[0,i]);

print("Initial State: ",map[convert_state_to_string(temp)])
print("Steady State: ")
for i in range(len(tres[0])):
    if ( tres[0,i] > 0.00001):
        print(" state: ",map[index2str[i]], " probability: ", tres[0,i]);
```

In [264]:
```
#res = v*Trans_Matrix
def countInfected(list):
    countInf = 0
    for i in range(len(list)):
        if(list[i] == 1 or list[i] == 2):
            countInf += 1
    return countInf


def unittest_StateBase(days, Trans_Matrix, temp, res):
    print("Initial State: ",map[convert_state_to_string(temp)])

    tempres = res
    dcount = 0
    prob_numInf = {}
    X_days = [i+1 for i in range(days)]
    for i in range(n):
        tempArray = [0 for i in range(days)]
        prob_numInf[i] = tempArray


    while dcount < days:
        tempres = np.matmul(tempres,Trans_Matrix)

        proOfnumInfected = [0 for i in range(n)]
        for i in range(len(tempres[0])):
            index = countInfected(map[index2str[i]]) - 1
            proOfnumInfected[index] += tempres[0,i]

        for i in range(n):
            tempArray = prob_numInf[i]
            tempArray[dcount] = proOfnumInfected[i]
        dcount += 1


    ##plot
    fig2, ax2 = plt.subplots( dpi = 150)
    for i in range(n):
        plt.plot(X_days, prob_numInf[i])
    labels = ["Number of Infected people is: "+str(i+1) for i in range(n)]
    plt.legend(labels)
    ax2.yaxis.set_major_formatter(PercentFormatter(1))
    ax2.xaxis.set_major_locator(MaxNLocator(integer=True))
    ax2.set_title("Probability of the  number of currently infected people vs Number of Days",fontsize = 10)
    plt.ylabel('Probability of the number of currently infected people', fontsize=10)
    plt.xlabel('Number of Days', fontsize=10)

    maxState = map[index2str[0]]
    maxProb = tempres[0,0]
    for i in range(len(map)):
        if (tempres[0,i] > maxProb):
            maxState = map[index2str[i]]
            maxProb = tempres[0,i]

    for i in range(days):
        print("At day ",i,", the probability of ",n," people are currently infected is ",prob_numInf[n-1][i])
```

The above code are used to simulate the relationship between the number of current infected people with the number of days.

In [265]:
```
def countInfected_AllIncluded(list):
    countInf = 0
    for i in range(len(list)):
        if(list[i] > 0):
            countInf += 1
    return countInf


def unittest_AllInfected(days, Trans_Matrix, temp, res):
    print("Initial State: ",map[convert_state_to_string(temp)])

    tempres = res
    dcount = 0
    prob_numInf = {}
    X_days = [1+i for i in range(days)]
    for i in range(n):
        tempArray = [0 for i in range(days)]
        prob_numInf[i] = tempArray


    while dcount < days:
        tempres = np.matmul(tempres,Trans_Matrix)

        proOfnumInfected = [0 for i in range(n)]
        for i in range(len(tempres[0])):
            index = countInfected_AllIncluded(map[index2str[i]]) - 1
            proOfnumInfected[index] += tempres[0,i]

        for i in range(n):
            tempArray = prob_numInf[i]
            tempArray[dcount] = proOfnumInfected[i]
        dcount += 1

    ##plot
    fig2, ax2 = plt.subplots( dpi = 150)
    for i in range(n):
        plt.plot(X_days, prob_numInf[i])
    labels = ["Number of people who got infected is: "+str(i+1) for i in range(n)]
    plt.legend(labels)
    ax2.yaxis.set_major_formatter(PercentFormatter(1))
    ax2.xaxis.set_major_locator(MaxNLocator(integer=True))
    ax2.set_title("Probability of the  number of people who got infected vs Number of Days",fontsize = 10)
    plt.ylabel('Probability of the  number of people who got infected', fontsize=10)
    plt.xlabel('Number of Days', fontsize=10)

    maxState = map[index2str[0]]
    maxProb = tempres[0,0]
    for i in range(len(map)):
        if (tempres[0,i] > maxProb):
            maxState = map[index2str[i]]
            maxProb = tempres[0,i]

    for i in range(days):
        print("At day ",i,", the probability of ",n," people who got infected is ",prob_numInf[n-1][i])
    print("The most likly state is : ",maxState, " and the probability: ", maxProb);
```

The above code are used to simulate the relationship between the number of people who got infected with the number of days.

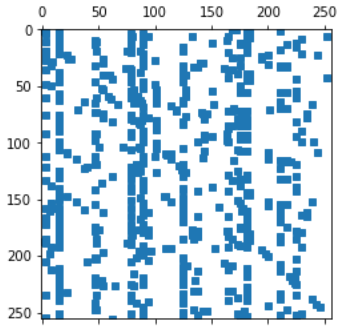
4 Analysis

4.1 Visualization of Transition Matrix

The transition matrix is plot below.

In [266]:
```
plt.spy (Trans_Matrix, markersize=5, precision=.1)
```

Out[266]:
```
<matplotlib.lines.Line2D at 0x11bd788d0>
```



As you can see the transision matrix, the x-axis and y-axis represent the id of states. The meaning of element $P(i, j)$ in this transition matrix is that the probability state i jumps to state j.

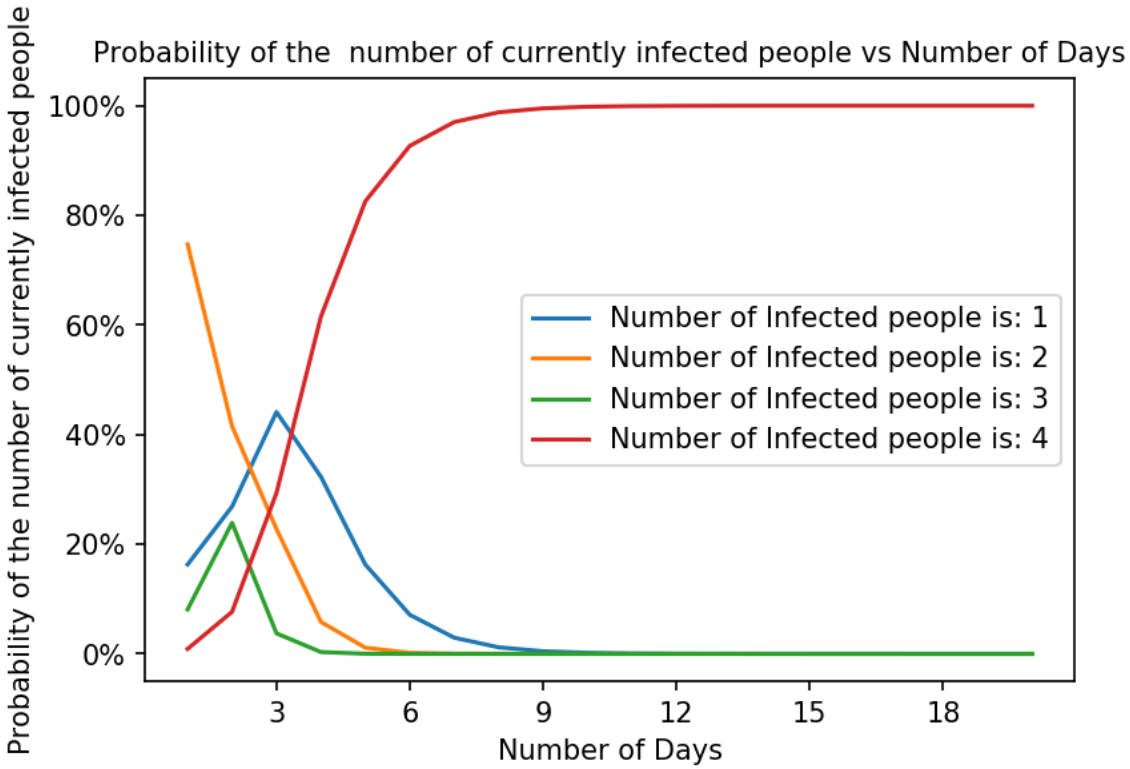
4.2 Infection Rate

In this case, we calculate relationship between the number of currently infected people and time. The transition probability will be updated once per day. By adding up the probability of all states with i infected person, the probability of i people are currently infected can be calculated.

As you can see in the following fiture, the curve reaches to the steady state really fast. It means that the infection rate of COVID-19 is really high and it only take fews days the entire family or people in a office will be infected if any one of them was exposed in the beginning.

In [267]:
```
unittest_StateBase(20,Trans_Matrix,temp,res)
```

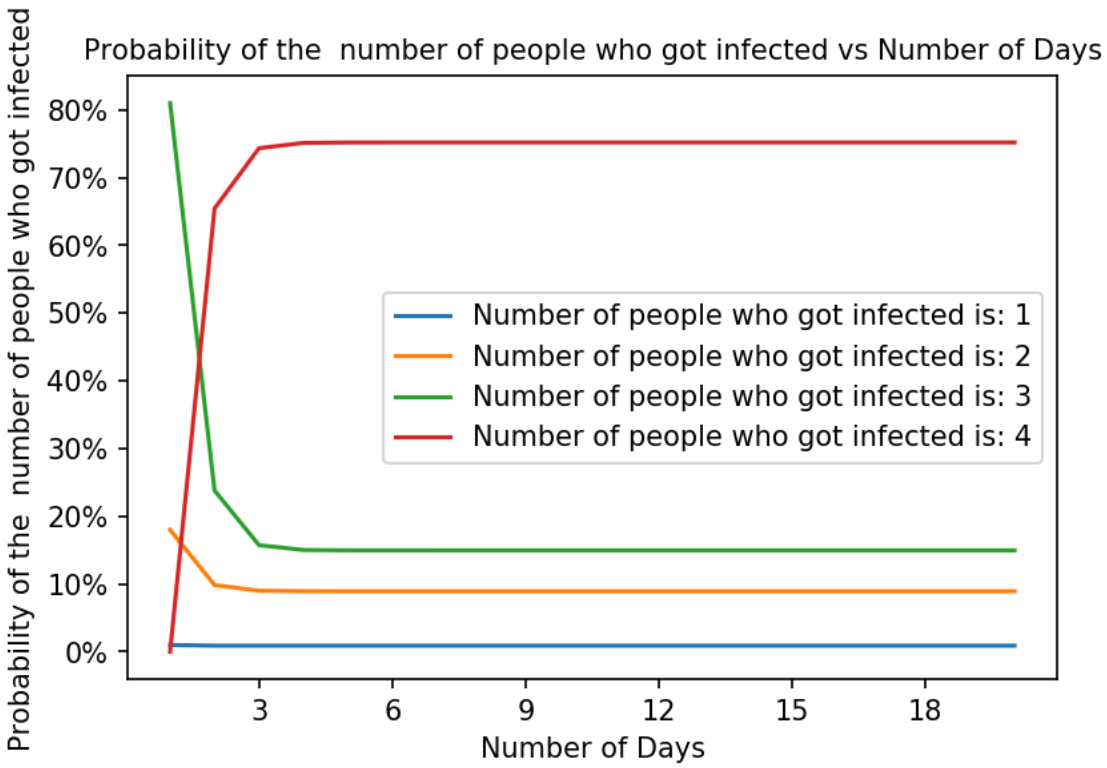
```
Initial State:  (0, 0, 2, 0)
At day  0 , the probability of  4  people are currently infected is  0.008999999999999996
At day  1 , the probability of  4  people are currently infected is  0.07666019999999998
At day  2 , the probability of  4  people are currently infected is  0.29362590828
At day  3 , the probability of  4  people are currently infected is  0.6152146243427521
At day  4 , the probability of  4  people are currently infected is  0.8256947070189521
At day  5 , the probability of  4  people are currently infected is  0.9268266147959962
At day  6 , the probability of  4  people are currently infected is  0.9701801401135591
At day  7 , the probability of  4  people are currently infected is  0.98798578299405
At day  8 , the probability of  4  people are currently infected is  0.9951808153210284
At day  9 , the probability of  4  people are currently infected is  0.9980702065645658
At day  10 , the probability of  4  people are currently infected is  0.9992277482351931
At day  11 , the probability of  4  people are currently infected is  0.999691046322046
At day  12 , the probability of  4  people are currently infected is  0.9998764101107493
At day  13 , the probability of  4  people are currently infected is  0.9999505627035008
At day  14 , the probability of  4  people are currently infected is  0.9999802248675089
At day  15 , the probability of  4  people are currently infected is  0.9999920899128467
At day  16 , the probability of  4  people are currently infected is  0.9999968359596804
At day  17 , the probability of  4  people are currently infected is  0.9999987343829996
At day  18 , the probability of  4  people are currently infected is  0.9999994937530604
At day  19 , the probability of  4  people are currently infected is  0.9999997975012018
```



4.3 Steady State

In [268]:
```
unittest_AllInfected(20,Trans_Matrix,temp,res)
```

```
Initial State:  (0, 0, 2, 0)
At day  0 , the probability of  4  people who got infected is  0.0
At day  1 , the probability of  4  people who got infected is  0.6544800000000001
At day  2 , the probability of  4  people who got infected is  0.7434973799999998
At day  3 , the probability of  4  people who got infected is  0.7513019263115998
At day  4 , the probability of  4  people who got infected is  0.7519363476701179
At day  5 , the probability of  4  people who got infected is  0.7519872343360171
At day  6 , the probability of  4  people who got infected is  0.7519913069228374
At day  7 , the probability of  4  people who got infected is  0.7519916327495585
At day  8 , the probability of  4  people who got infected is  0.751991658815926
At day  9 , the probability of  4  people who got infected is  0.7519916609012387
At day  10 , the probability of  4  people who got infected is  0.7519916610680639
At day  11 , the probability of  4  people who got infected is  0.7519916610814096
At day  12 , the probability of  4  people who got infected is  0.7519916610824776
At day  13 , the probability of  4  people who got infected is  0.7519916610825634
At day  14 , the probability of  4  people who got infected is  0.7519916610825698
At day  15 , the probability of  4  people who got infected is  0.7519916610825701
At day  16 , the probability of  4  people who got infected is  0.7519916610825701
At day  17 , the probability of  4  people who got infected is  0.7519916610825701
At day  18 , the probability of  4  people who got infected is  0.7519916610825701
At day  19 , the probability of  4  people who got infected is  0.7519916610825703
The most likly state is :  (3, 3, 3, 3)  and the probability:  0.7519914800803487
```



Markov Chain model provides us another view of analysis. Markov Chain does not focus on the process, while it more focus on the state and the probability of the state, especially the steady states.

After 20 days, the state with the largest probability is the state that everyone get recovered or killed. As you can see in the printing above, everyone's state is recovered/death.

In this case, we talk about the steady state of the simulation. If there are four people in this small community and one of them is exposed, the probability of all of them get infected after 6 days is around 75 percent.

5. Conclusion

In this part, we talked about the Markov Chain Analysis on COVID-19 with SEIR model. By analysing the this problem, we can see that COVID-19 has fast infection rate and a small family could all get infected within few days.Due to the limitation of Markov Chain model, our implementation only focused on small community.

Author Contributions

All team member contributes equally to the project. Muyang Guo is mainly responsible to the cellular automata method, Dayu Zhu works on the ordinary differential equation method, and Yibo Wang researches on the Markov chain model.