DATA ANALYTICS
GLASGOW

Data Programming in Python

Project

This assignment sheet is assessed and contributes 20% to your overall grade for Data Programming in Python. You can obtain a total of 20 marks for this project (see marking scheme at end).

Please upload your answers before Monday, January 25th, 10am (UK time).

To upload your answers, log on to Moodle and go to Data Programming in Python. *Under* Week 10, *there is a link* Upload project. *Click on this link to upload the file containing your commented Python code. You can either upload a Python source file (file extension* `.py`*) or a Jupyter notebook (file extension* `.ipynb`*). Please do not upload your code in other formats.*

Background

In this project you will implement a simple simulator of an epidemic. In this model a person can have four states:

- Susceptible individuals can be infected with the disease.
- Recovered individuals are free of the disease and cannot pass it on to others. We assume that recovered individuals are not susceptible to the disease any more. We can also class individuals who are not susceptible to the disease (for example because they have built up immunity as a result of an infection with a related pathogen) as recovered, as they cannot be infected with the disease either.
- Infected individuals can spread the disease to susceptible individuals they are in contact with. They will either recover from the infection or die as a result of the infection.
- After individuals have died from the infection their state will not change.

Assume that the individuals live on a regular $m \times n$ grid, the first individual lives at $(0, 0)$, the second at $(0, 1)$, etc. with the (mn)-th individual living at $(m - 1, n - 1)$.

The speed at which the epidemic spreads depends on how much contact there is between infected individuals and susceptible individuals.

For this project we will assume that infected individuals do not reduce their contact with others. This can be the case if symptoms are only mild for the majority of cases and/or if individuals become infectious before they develop symptoms (though the latter would alter the dynamic a little).

Details

Contacts

More mathematically speaking, we will assume that at the begin of the simulation each individual is, independently of each other, either …

- infected with a probability of α_{infected},
- has already recovered or is immune with a probability of $\alpha_{\text{recovered}}$, or
- is susceptible to the disease with probability $1 - \alpha_{\text{infect}} - \alpha_{\text{recovered}}$.

If we denote these states by their first letters, we can use

```
np.random.choice(["I","R","S"], 1, True, [alpha_infected,
                                          alpha_recovered,
                                          1-alpha_infected-alpha_recovered])[0]
```

to simulate these initial states.

We assume that every day the probability that an infected person infects a susceptible person it is in contact with on that day is γ.

Each day an infected person will recover with a probability of β_{recover} or die from the infection with a probability of β_{death}. The person will be infected for at least another day with a probability of $1 - \beta_{\text{recover}} - \beta_{\text{death}}$.

One can show that the probability of dying of the infection is $\frac{\beta_{\text{death}}}{\beta_{\text{recover}}+\beta_{\text{death}}}$ and that, on average, individuals are infectious for $\frac{1}{\beta_{\text{recover}}+\beta_{\text{death}}}$ days after being infected. A susceptible person who is in contact with an infectious person has thus a probability of $\frac{\gamma}{\beta_{\text{recover}}+\beta_{\text{death}}}$ of being infected by that person. (You won't need these results for your simulation.)

Contacts

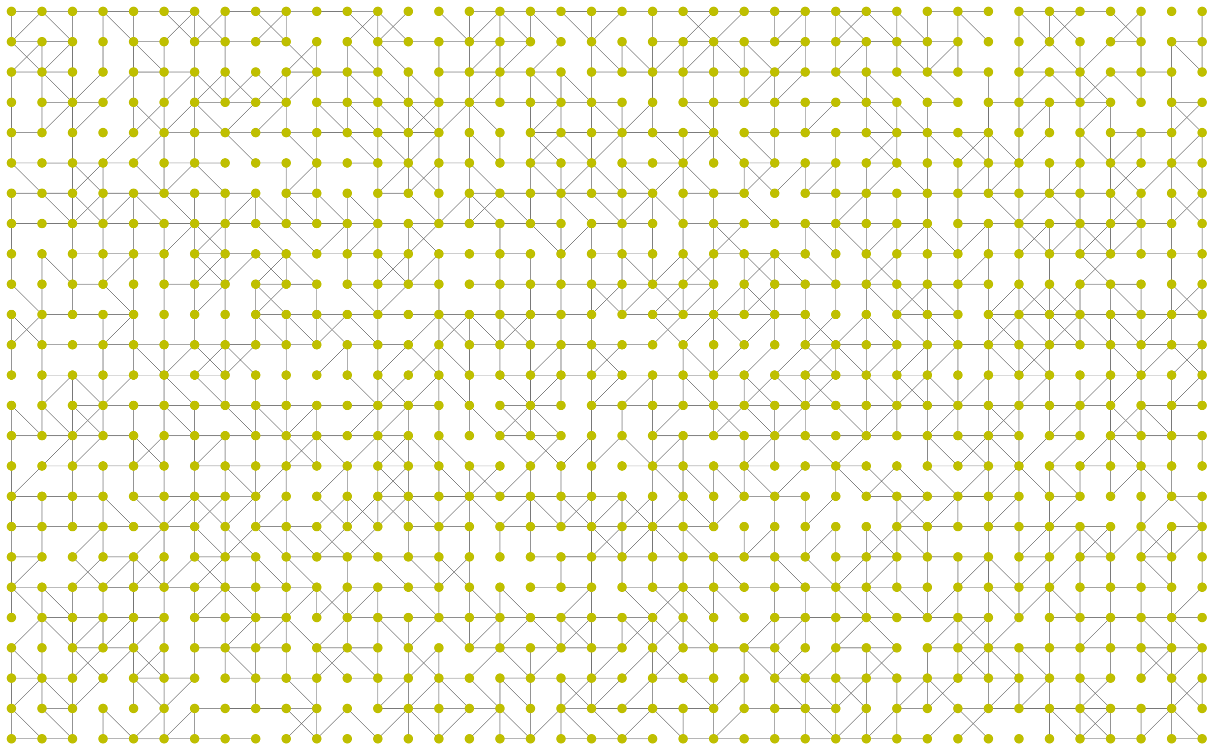
An important factor controlling how fast the disease spreads is the number of other persons each person is in contact with and also where these contacts are.

In this simulation we will simulate who is in contact with whom using the following two parameters:

- Individuals can only be in contact with each other another if their distance on the grid is at most r.
- On average, each individual is in contact with k other individuals.

(Note that the values which k can take depend on r. If $r = 1$ then individuals can only be in contact with their immediate neighbours, of which there are only four, so we cannot choose k larger than 4 in that case. The algorithm below will not work well if k is chosen close to the total number of neighbours at distance r or less.)

We can represent the contact structure using a graph. The figure below shows an example for $m = 40$, $n = 25$, $r = 2$ and $k = 4$.



This graph can be simulated as follows.

Repeat …

i. Draw a pair of individuals.

ii. If the distance between the pair is at most r and the pair is not already connected, connect the pair.

…until $\frac{m \times n \times k}{2}$ new connections have been added.

Simulation

Once the population has been initialised as set out on the previous pages, the simulation can be carried out as follows.

1. For each individual and …

   for each susceptible individual the person is in contact with, set their state to ”infected” with a probability of γ.

2. For each infected individual …

   Set the state to ”recovered” with probability β_{recover} and to ”dead” with probability β_{death}.

3. Record the current state for each individual.

The above needs to be repeated for each day.

The table below gives an overview of the parameters of the simulation.

Parameter	Interpretation	Default value
m	Number of “rows” of individuals	40
n	Number of “columns” of individuals	25
r	Individuals can only be in contact with neighbours at a distance of at most r	6
k	Average number of contacts (connections) per individual	20
α_{infected}	Probability that an indivual is infected at the begin of the simulation	0.01
$\alpha_{\text{recovered}}$	Probability that an indivual is immune to the infection at the begin of the simulation	0
β_{recover}	Probability that an infected individual recovers on any given day	0.05
β_{death}	Probability that an infected individual dies on any given day	0.005
γ	Probability that an infected individual infects a contact on any given day	0.075
N	Duration (number of simulated days) for which the simulation is run	100

Implementation

Write a class `Simulation` which implements the above simulation.

- The `__init__` method should take the settings as argument and initialise the simulation by creating the contact graph and the initial state of each individual.
- The `run` method should carry out the simulation as described above.
- The `plot_state` method should take a time as argument and plot the state of the population at that point in time. Each individuals should be represented as a dot. Different colours should be used to represent the different states (susceptible, infected, recovered and dead) of the individuals. Individuals who are in contact with each other should be connected using a line.
- The function `chart` should draw a line plot showing how the number of susceptible, infected, recovered and dead evolve over time.
- Write a method `max_infected` which returns the maximal number of infected individuals.
- Write a method `peak_infected` which returns the day at which the number of infected individuals was maximal.
- As the simulation is random, each run will give slightly different results. Write a class method (or static method) `averaged_chart` which performs the above simulation N times and averages the curves from the simulations. The function should also return the averages of the quantities calculated by the methods `max_infected` and `peak_infected` for each run. The implementation of `averaged_chart` should not duplicate code but call already implemented for `N` replicated simulations. The method `averaged_chart` should take `N` as well as the settings of the simulation as arguments. The settings should be optional with the above defaults used otherwise.

Example

Below is an example of how the class could be used (using the above defaults, but $r = 2$ and $k = 4$, i.e. strong social distancing).

Run simulation:

```
s = Simulation(settings)
s.run()
```

Plot state at time 100:

```
s.plot_state(100)
plt.show()
```

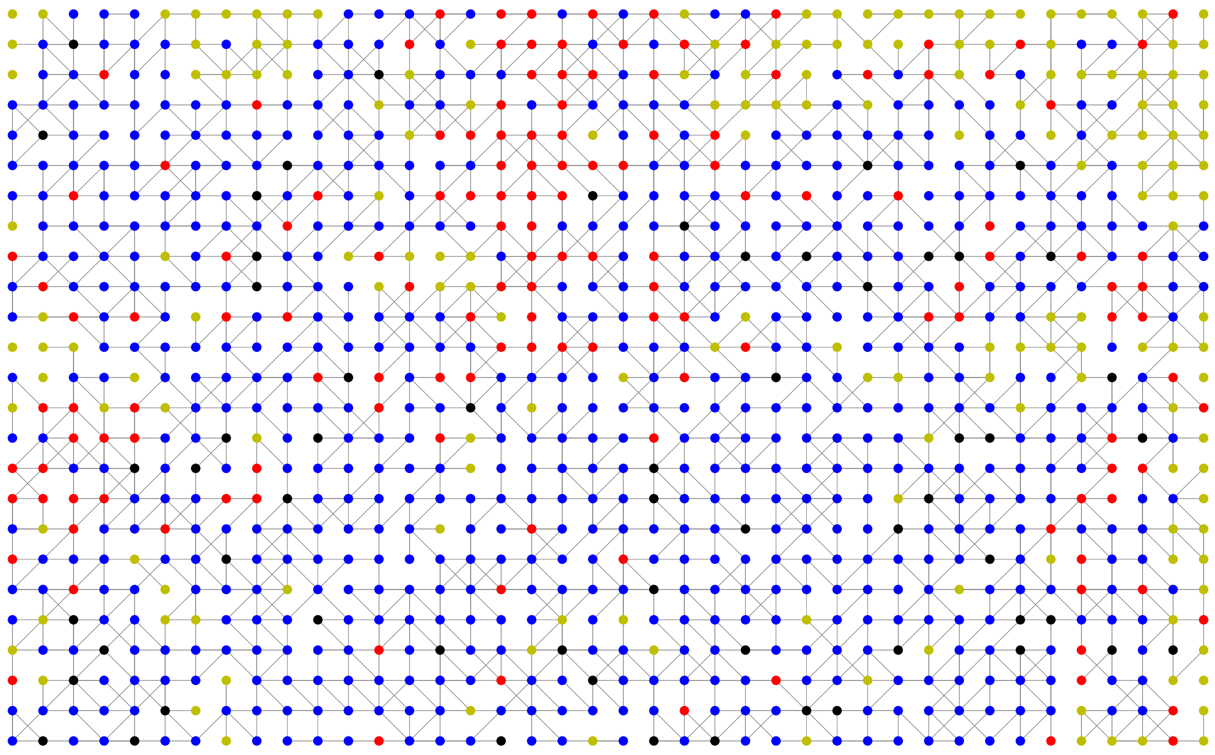
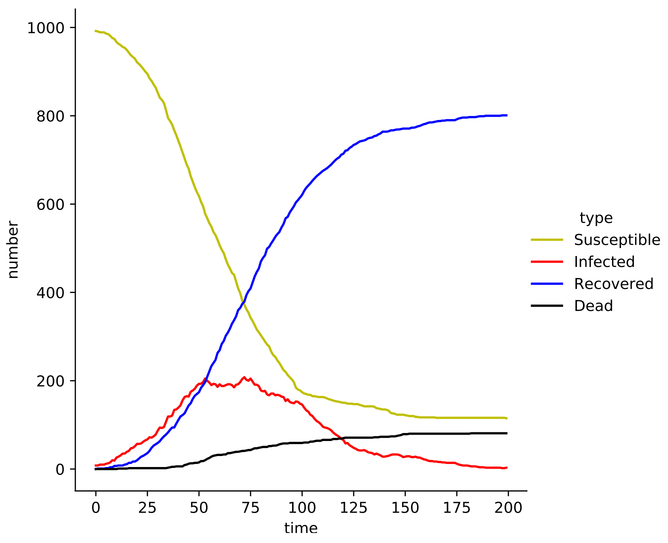


Chart:

```
s.chart()

## /home/chaitanya/.local/lib/python3.6/site-packages/seaborn/_decorators.py:43: FutureWarnin
##   FutureWarning

plt.show()
```



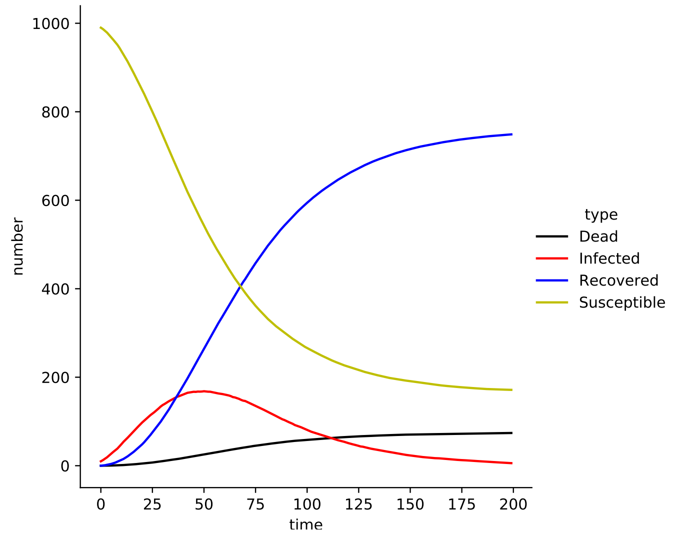
number
time
type
Susceptible
Infected
Recovered
Dead
0
200
400
600
800
1000
0
25
50
75
100
125
150
175
200

Averaged chart ($N = 100$ replications):

```
Simulation.chart_average(100, settings)

## {'max': 191.1, 'peak': 53.54}
##
## /home/chaitanya/.local/lib/python3.6/site-packages/seaborn/_decorators.py:43: FutureWarnin
##   FutureWarning

plt.show()
```



Assessment

The following scheme will be used to assess your submission.

Criterion / "Unit test"	Marks if correct
Initialisation correct?	2 marks
Network of contacts simulated correctly?	3 marks
Dynamics implemented correctly?	3 marks
Data recorded correctly?	2 marks
`plot_state` as required?	2 marks
`chart` as required?	2 marks
`max_infected` and `peak_infected` as required?	1 mark
`averaged_chart` implemented as required?	3 marks
Code clear and well organised?	2 marks