上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

项目报告

PROJECT REPORT

项目名称：病毒传播的可视化仿真模拟

团队名称：	Python 4.0
团队成员：	翟栎、唐菘菘、余毅炫
课程名称：	程序设计
指导教师：	鲍　杨
学院(系)：	安泰经济与管理学院

目录

1 背景 **2**
1.1 选题原因 . . . 2
1.2 项目介绍 . . . 2
1.3 Library 介绍 . . . 2
1.3.1 Numpy . . . 2
1.3.2 Matplotlib . . . 2

2 代码解释 **3**
2.1 基础传染模型 . . . 3
2.1.1 类定义 . . . 3
2.1.2 位置变动 . . . 5
2.1.3 状态变动 . . . 5
2.2 外加实际因素 . . . 6
2.2.1 佩戴口罩 . . . 6
2.2.2 社交隔离 . . . 7
2.2.3 设立医院 . . . 7
2.3 可视化与运行 . . . 8
2.3.1 可视化 . . . 8
2.3.2 运行模拟器 . . . 9

3 小结与讨论 **9**
3.1 设计过程 . . . 9
3.1.1 仿真程度 . . . 9
3.1.2 算法效率 . . . 10
3.2 实际应用 . . . 10

1 背景

1.1 选题原因

去年 12 月开始爆发的新冠肺炎席卷了全球, 给世界上各国的社会与经济带来了巨大的危机. 幸运的是, 中国在有效的防疫措施下顺利的渡过了疫情. 但与中国相比, 一些西方国家在抗议方面的措施并不那么有效, 民众的配合度不高. 我们认为其中的一个原因是病毒的不可见性, 即民众并不了解病毒在人群中传播的迅速以及后果的严重性. 于是我们的想法便是编写一个病毒传播的模拟器, 以可视化的方式直观的向人们表现病毒传播的过程.

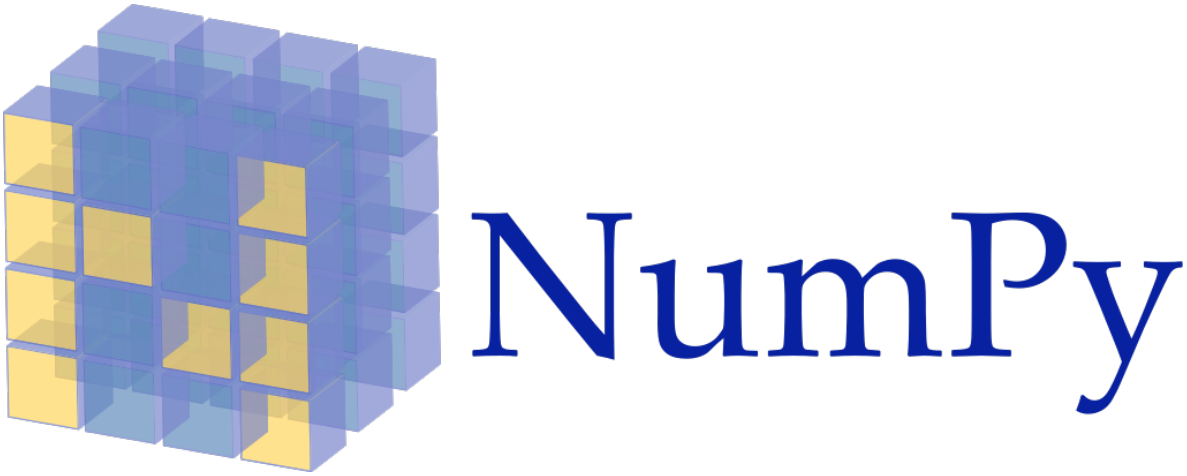
1.2 项目介绍

此项目以可视化病毒的传播为目标. 在基础传染模型中, 我们根据传染病学的 SEIRD 模型, 将处于疫情期间的人群分为 S(Suspectible)、E(Exposed)、I(Infectious)、R(Recovered)、D(Dead) 五大类, 通过数组之间的操作实现状态的转化. 随后我们有根据现实情况加入了实际因素: 佩戴口罩、社交隔离、设立医院, 使模拟更具实际意义.

1.3 Library 介绍

1.3.1 Numpy

NumPy 是 Python 语言的一个扩充链接库, 支持高阶大量的维度数组与矩阵运算, 此外也针对数组运算提供大量的数学函式函式库. NumPy 通常与 SciPy（Scientific Python）和 Matplotlib（绘图库）一起使用. Numpy 可以应用于仿真建模、数学分析等. 其优点有: 进行相同的任务速度更快、储存效率与输入输出性能远高于 Python 中等价的基本数据结构等.



1.3.2 Matplotlib

Matplotlib 是一个综合库, 用于在 Python 中创建静态, 动画和交互式可视化, 是 Python 数值数学扩展包 NumPy 的可视化操作界面, 可以用于绘制曲线图、直方图、散点图、3D 图等. 其

优点有: 图形绘制相较 Gnuplot 更加美观、与 Python 深度集成等.

2 代码解释

※ 分工情况: 翟栎、唐菘菘、余毅炫分别负责 2.1、2.2、2.3. 在翟栎同学完成基础传染模型之后, 唐菘菘同学除了完成实际因素的添加, 还就基础模型的代码做了算法的优化, 提高了代码性能.

```
import numpy as np

import matplotlib.pyplot as plt
```

首先, 在编写模拟器之前, 我们先导入我们所需的第三方类库: Numpy 与 Matplotlib.

2.1 基础传染模型

我们首先建立一个基础的传染病模型来模拟病毒的传播.

2.1.1 类定义

```
class People(object):
    """This class stand for a community of people.

    Attributes:
    round: An integer count of how many times infection has been carried out.
    total_num: An integer indicating the population of the community.
    location: A numpy array indicating the Every individual's coordinates.
    initial_infection: A integer indicating the number of 'Patient Zero'.
    scale_of_movement: A integer indicating the range of personal movement.
    """
    def __init__(self, total_num, initial_infection):
        self.round = 0
        self.total_num = total_num
        self.location = np.random.normal(0, 100, (total_num,2))
        self.initial_infection = initial_infection
        self.scale_of_movement = 5
```

为了便捷起见, 我们将人群定义为一个类: `People`, 并赋予其下列类变量:

- round: 代表传染进行的轮数.
- total_num: 代表People所代表的人群的总人口数.
- location: 代表由人群中每个人的 x, y 坐标构成的数组.
- initial_infection: 代表疫情爆发最开始感染症数量.
- scale_of_movement: 代表人群中的个人位置变化范围大小.

其中round的初始值设置为 0, 表示模拟未开始. 此外, 我们将人口分布假设为正态分布, 即坐标值依正态分布概率论密度函数随机生成. 对于总人群, 我们将均值设置为 0, 标准差设置为 100.

```
'''Here we define the different type of person's status as diferent properties
according to our SERID model.
'''
@property   #For displaying the suspectible people
def S(self):
    return self.location[self.status == 0]

@property   #For displaying the exposed people
def E(self):
    return self.location[self.status == 1]

@property   #For displaying the infectious people
def I(self):
    return self.location[self.status == 2]

@property   #For displaying the recovered people
def R(self):
    return self.location[self.status == 3]

@property   #For displaying the dead people
def D(self):
    return self.location[self.status == 4]
```

接着我们使用@property装饰器将人的属性与状态值绑定, 即: S, E, I, R, D 五类人分别对应的状态值为 0, 1, 2, 3, 4.

```
def initialize(self):
    """This method is responsible for initializing all the arrays needed to operate
    the emulator. The timer array is used to calculate how many days a certain patient
    has been infected so that we can change their status from E to I. At the beginning,
    all alements in timers are assigned a value of -1, indicating that the timer has
    not been activated.
    """
    self.timer = np.array([-1] * self.total_num)
    self.status = np.array([0] * self.total_num)
    index_initial = np.random.randint(0, self.total_num - 1, (self.initial_infection, 1))
    for i in index_initial:
        self.status[int(i)] = 1
        self.timer[int(i)] = self.round
```

我们定义了initialize方法来初始化People. 其中timer用于记录单个人累计感染天数的计数器, 其初始值均设置为 -1 代表计时未开始. 而status表示人的健康状况, 其具体对于情况见上文的绑定部分, 初态时所有人的状态值设置为 0. 随后我们根据初始病人数随机生成" 零号病人" 所对应的序号, 并将他们的状态值初始化为 1.

2.1.2 位置变动

```
def update_location(self):
    loca_change = np.random.normal(0, self.scale_of_movement, (self.total_num, 2))
    self.location = self.location + loca_change
```

对于单个人的位置变动, 我们定义了update_location方法. 我们仍然假设其服从正态分布, 即下一时刻较上一时刻的位置该变量依照正态分布概率密度函数随机生成, 并为我们将这一分布的均值设置为 0, 标准差设置为 5. 在生成了位置变化量的数组之后, 返回加上变化量之后的位置数组, 以此方式达成位置更新.

2.1.3 状态变动

```
def change_state(self, recovery_death_rate, proportion):
    '''This method is responsible of status change that only involves one person,
    like E -> I and I -> R/D.
    '''
    self.timer[(self.status == 1) & (self.timer == -1)] = self.round
    self.status[(self.status == 1) & (self.round - self.timer > 14)] = 2

    for i in range(self.total_num):
        if self.status[i] == 2:
            if np.random.normal() <= recovery_death_rate:
                self.status[i] = 3 if np.random.normal() <= proportion else 4
                if self.isolated[i]:
                    self.hospital_occupation -= 1
            elif self.hospital_occupation < self.hospital_capacity and not self.isolated[i]:
                self.isolated[i] = True
                self.hospital_occupation += 1
```

对于 E 人群随时间自发地转化为 I 人群的过程以及 I 人群随时间自发地转化为 R/D 人群的过程, 我们定义了change_status方法来实现. 我们先按照总体概率recovery_death_rate随机选取一部分 I 人群, 再按照一定的比例proportion随即决定他们是转化为 R 人群还是 D 人群. 依概率筛选的机制我们使用正态分布的分布函数, 例如:

```
if np.random.normal() <= recovery_death_rate:
```

便表示满足条件语句的概率为服从标准正态分布的随机变量 X 的观测值小于recovery_death_rate的概率, 在之后的筛选中我们也会反复用到这一机制.

```
def infection(self, infection_rate_E, infection_rate_I, safe_distance):
    '''This method is responsible for carrying out status changes that involves two
    people, like S + E -> E + E and S + I -> E + I. The posibility to infect
    is base on the normal distributon, which is P(X <= x).
    '''
    sqr_safe_distance = safe_distance ** 2
    S_list = np.where(self.status == 0)[0]
    S_location_list = self.S

    for person in self.E:
        dm_E = (S_location_list - person) ** 2
        d_E = dm_E.sum(axis = 1)
        for i in S_list[d_E < sqr_safe_distance]:
            if (self.round >= self.weartime[i]
            and np.random.normal() <= infection_rate_E - self.drop_rate
            or  self.round <  self.weartime[i]
            and np.random.normal() <= infection_rate_E):
                self.status[i] = 1

    for person in self.Iui:
        dm_I = (S_location_list - person) ** 2
        d_I = dm_I.sum(axis = 1)
        for i in S_list[d_I < sqr_safe_distance]:
            if (self.round >= self.weartime[i]
            and np.random.normal() <= infection_rate_I - self.drop_rate
            or  self.round <  self.weartime[i]
            and np.random.normal() <= infection_rate_I):
                self.status[i] = 1
```

对于携带病毒的 E 人群和 I 人群传染健康的 S 人群的过程, 我们定义了`infection`方法来实现. 由于两种传染过程的唯一区别是传染概率不同, 我们不妨以 E 人群传染 S 人群为例. 我们先使用 numpy 中的`where`方法筛选出 S 人群, 将其单独设为`S_list`作为待传染的对象. 具体的传染机制为: 对于每一个 E 人群中的个人, 首先用两点间距离公式计算他与所有人的距离, 即代码中的`sqrt(d_E)`; 其次, 筛选出`S_list`中距离小于安全距离`safe_distance`的人群作为新的待感染对象; 随后, 用前文提到的改了筛选机制随机选出被传染的对象, 将其状态值 +1. 若对象佩戴口罩, 则筛选概率为`infection_rate - self.drop_rate`; 若不然, 则传染率为`infection_rate`.

2.2 外加实际因素

随后在基础传染模型的基础之上, 我们加入以下三个方法为病毒传播模型添加了一些实际因素. 它们可以被分别调用, 也可以同时调用.

2.2.1 佩戴口罩

```
def facemasking(self, number_of_wearing = 0, st = 0, ed = 1, drop_rate = 3):
    self.weartime = np.array([201] * self.total_num)
    for i in range(number_of_wearing):
```

```
        self.weartime[int(i)] = np.random.randint(st, ed)
        self.drop_rate = drop_rate
```

对于佩戴口罩我们定义了facemasking方法来实现. 该方法能够模拟令指定数量的人口, 在指定的时间区间内, 陆续开始戴上口罩的状况, 佩戴口罩的效果是令这个个体被传染的概率下降. 其中各个参数分别代表:

- number_of_wearing: 总的佩戴口罩人数.
- st: 开始佩戴口罩的时间.
- ed: 结束佩戴口罩的时间.
- drop_rate: 佩戴口罩导致的传染概率下降值.

由于在本项目中, 不存在其他根据人的下标来进行随机化处理的代码, 为了方便起见, 我们直接令下标为 [0, number_of_wearing) 的人佩戴口罩. 单个个体佩戴口罩的起始时间, 记录在实例变量weartime数组内. 对于下标为i的人, 若被指定不佩戴口罩, 则令weartime[i] = 201, 表示其佩戴口罩的起始时间为无穷大; 否则, 其佩戴口罩的起始时间weartime[i]服从 [st, ed) 区间内的均匀分布. 基础模型部分的代码反映了口罩佩戴的降低传染率效果, 体现在infection方法中.

2.2.2 社交隔离

```
def social_distancing(self, new_scale_of_movement = 5):
    self.scale_of_movement = new_scale_of_movement
```

我们又定义了social_distancing用于实现社交距离的模拟. 输入参数new_scale_of_movement, 以更新update_location方法中人群移动量所服从的正态分布的标准差.

2.2.3 设立医院

```
@property   #For displaying the unisolated infectious
def Iui(self):
    return self.I[self.isolated[np.where(self.status == 2)[0]] == False]
```

加入医院这一实际因素后, infectious 人群便分化为了两种: 进入医院接受医学隔离的, 和没有机会入住医院的. 为此, 我们单独定义一种属性 Iui, 将其与状态值 5 绑定, 一次代表虽然出现症状但仍未住院隔离的人群.

```
    def isolation(self, capacity = 0):
        self.hospital_capacity = capacity
        self.hospital_occupation = 0
        self.isolated = np.array([False] * self.total_num)
```

最后, isolation用于实现医学隔离的模拟, 医院只能隔离已经出现症状的患者, 即 I 类人群. 相关的变量如下:

- hospital_capacity =capacity: 医院的最大容量.
- hospital_occupation: 已被占用的医院容量.
- isolated: 布尔数组, 表示一个人是否接受了隔离.

医学隔离操作的主体部分, 位于基础模型内的change_state方法中. 每个 I 状态的病人, 可能在此方法内转化为 R, D. 若病人没有转化为后两种状态, 且医院仍有空位, 则其立即接受医学隔离, 若已经接受医学隔离的病人, 转化为了 R, D 状态, 则他/她的隔离结束, 释放一个医院空间. 除此之外, infection方法中, 能够感染他人的病人只能是未被隔离的病人, 因此在循环中使用从新定义的 Iui 数组参与传染. 在之后的可视化过程中, 为了不引起混淆, 也不显示已被医学隔离的病人.

2.3 可视化与运行

2.3.1 可视化

```
def visualize(self):
    plt.cla()
    plt.grid(False)
    p1 = plt.scatter(self.S[:, 0], self.S[:, 1], s = 1)
    p2 = plt.scatter(self.E[:, 0], self.E[:, 1], s = 1, c = 'orange')
    p3 = plt.scatter(self.Iui[:, 0], self.Iui[:, 1], s = 1, c = 'red')
    p4 = plt.scatter(self.R[:, 0], self.R[:, 1], s = 1, c = 'green')

    plt.legend([p1, p2, p3, p4], ['Susceptible', 'Exposed', 'Infectious', 'Recovered'],
    loc='upper right', scatterpoints=1)
    t = "Round: %s, Susceptible: %s, Exposed: %s, Infectious: %s, Recovered: %s,
         Dead: %s, NHS capacity: %s/%s" % \
         (self.round, len(self.S), len(self.E), len(self.I), len(self.R),
          len(self.D), self.hospital_occupation, self.hospital_capacity)
    plt.title(t)
```

我们定义了visualize方法来完成可视化, 主要分为两个部分: 建图部分与图例解释与标题部分. 首先对于建图部分, 我们使用plt.scatter()方法来在图上呈现People的坐标, 其中:

- plt.cla()用于清除当前图形中的当前活动轴.
- plt.grid(False)用于关闭背景网格线.

其次对于图例解释与标题部分, plt.legend()方法用于帮助展示不同数据点对应的图像名称, 让读者更好地认识数据结构; 而plt.title()方法用于用于设置标题 (本例先赋予t一个字符串, 并在直接使用plt.title(t)进行标题的建立).

2.3.2 运行模拟器

```
def update(self):
    self.change_state(-2.3, 1.28)
    self.infection(-0.255, 0.255, 2)
    self.update_location()
    self.round += 1
    self.visualize()
```

为了简化代码起见, 我们定义了update方法来达成集成化的传染模块, 其中包含: 完整的传染过程、位置的更新和散点图的更新.

```
if __name__ == '__main__':
    np.random.seed(1)
    plt.figure(figsize=(10, 10), dpi=85)
    plt.ion()

    p = People(10000, 3)

    p.initialize()
    p.facemasking()          #recommended parameter: number_of_wearing=7000, ed=60
    p.social_distancing()    #recommended parameter: new_scale_of_movement=2
    p.isolation()            #recommended parameter: capacity=2000

    for i in range(200):
        p.update()
        plt.pause(.1)
        plt.pause(3)
```

最后便是模拟器的运行部分, 首先我们先设置绘图窗口, 使用plt.fugure()生成单独的绘图窗口, 并使用plt.ion()打开交互模式; 接着定义了代表“人数为 10000、最初有 3 人感染”的People类变量p; 随后, 是控制各类实际因素的代码, 初始时默认不设置参数, 代表实际因素不发挥作用; 最后, 基于之前定义的变量和实际条件的设置情况, 将传染过程p.update()循环 200 次已完成完整的病毒传播过程.

3 小结与讨论

3.1 设计过程

3.1.1 仿真程度

经过讨论, 我们淘汰了基于人际网络的图模拟, 选择根据正态分布模拟人群的分布与移动, 人和人之间只有在距离小于安全阈值时, 才依概率传染病毒, 这一基础模型与现实的相似程度比较高, 实际运行也证明了这一点.

我们的程序设计思想, 源于元胞自动机算法, 后者强调用上一个时间单位的元素状态及其周围元素的状态, 来确定目前时间单位的元素状态. 而在实际编写代码的过程中, 我们不慎忽视了这

一点, `infection`方法事实上是根据已经被`change_state`函数改变了的新状态来进行计算的, 违背了元胞自动机原则, 逻辑上也存在着漏洞.

3.1.2 算法效率

整个程序过程的效率瓶颈位于基础模型中的`infection`部分.

最开始, 我们采用的是对于每一位 I, E 病人 (person), 计算全部人与他/她的当前距离, 存入数组, 并与安全阈值比较, 筛选出小于安全阈值的人进行概率判断, 这么做, 在这个部分的时间复杂度大约是 $O($`total_num`$\times($`self.I + self.E`$))$, 随着感染人数的上升, 运行速度将会越来越慢.

随后, 简单地改变了计算方法, 首先从所有人中选出健康人`self.S`, 对于每一位 I, E 病人, 只计算全部健康人与他/她的当前距离. 并且将距离公式中的开方转换为对安全阈值进行平方, 从而避免了对于大量数据的开方运算, 缩小了代码常数. 这么做, 在这个部分的时间复杂度大约是 $O($`self.S`$\times($`self.I + self.E`$))$, 这样, 在健康人数与感染人数相当的时候, 程序运行最慢, 此前和此后程序运行的速度都非常快, 整体的算法效率提升了.

3.2 实际应用

对于病毒传播的计算机模拟以及可视化, 有利于使我们更加直观地了解病毒传播的过程, 并且通过调整其中的参数, 我们能够开展一些比较性的研究, 为政策制定提供一些建议.

比如, 它为我们观察各项防疫举措的有效性提供了实用的工具. 通过单独调用三个实际因素方法, 我们发现了三者各自对于疫情的影响具有明显不同的特点.

我们发现, 在前 60 个时间单位内有 7000 人陆续戴上口罩, 能够有效地抑制病毒在人群中的传播, 虽然这不能防止病毒传播的地域的扩大; 通过实行充分的社交距离, 如将新的人群移动规模设置为 1, 能够将病毒传播有力地控制在局部区域内, 并且减少一些感染总人数; 而简单的医学隔离, 表现出了最差的防疫效果——即使医院有能力立刻隔离出现症状的感染者, 并且拥有高达总人口 30% 的最大容量, 医学隔离的存在却仍然几乎无法降低感染总人数.

这警示我们, 对于在潜伏期也具有一定传染性的病毒, 被动地隔离出现症状者几乎无济于事, 必须开展流行病学调查, 排查密切接触者并对其进行医学观察, 才能够真正有效地防控疫情.

对于当下的新冠肺炎疫情, 我们的模型同样可以在一定程度上做出一些解释. 中国在两个月内口罩佩戴率迅速提升到了较高水平, 人们在疫情期间进行充分的居家隔离, 方舱医院等的建设、流行病学调查的开展实现了有效的隔离医学观察. 三种因素并施, 是中国得以迅速平息冠状病毒大流行的重要原因