文档一：恶意代码模型

> 姓名：林绍钦
> 学号：518021910331

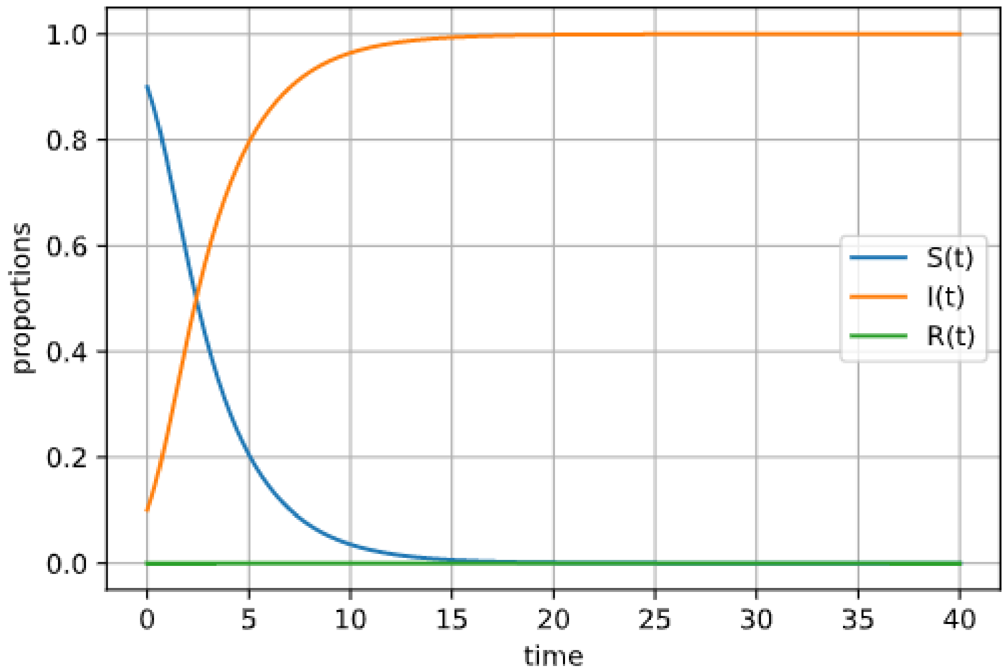
1 网络型恶意代码传播模型

- $S(t)$: 在时间t中未被感染的节点数量
- $I(t)$: 在时间t中已被感染的节点数量
- $R(t)$: 在时间t中已经被移除的节点数量
- $Q(t)$: 在时间t中可以从被感染节点中移除的节点数量
- N: 网络中节点总数
- $J(t)$: 曾经被感染的节点数量
- $\beta(t)$: 在时间t中未被感染的节点数量

1.1 随机网络传播

第一种情况，不考虑移除的情况，即**完全放任网络恶意代码的传播。**

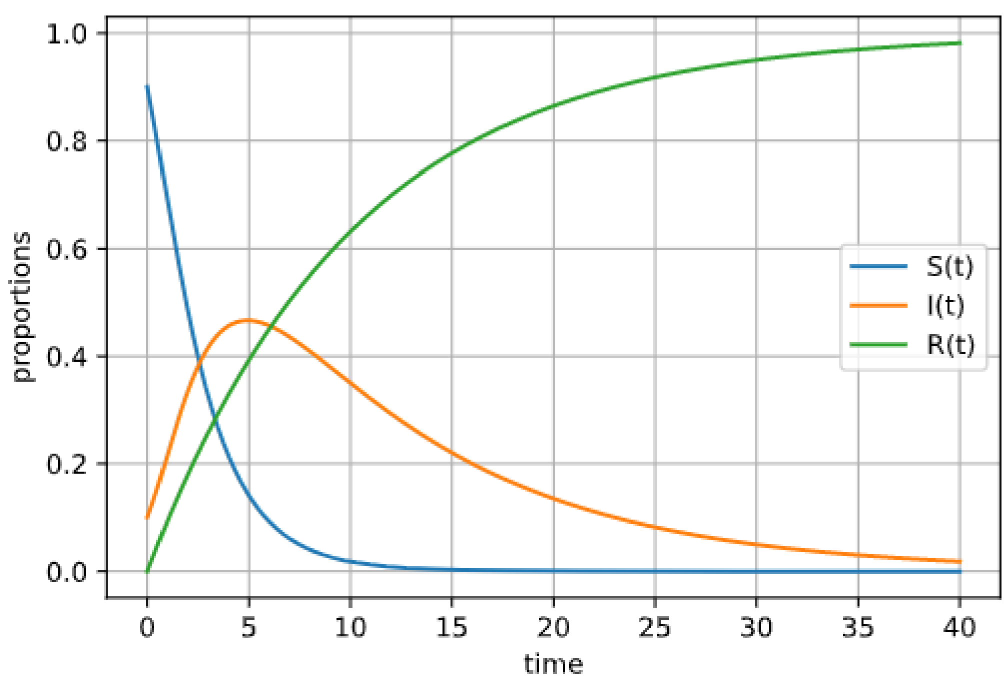
$$\begin{cases} \lambda_k = [1-(1-i)^k]\beta \\ \frac{ds}{dt} = -\lambda s, \frac{di}{dt} = \lambda s, \frac{dr}{dt} = 0 \end{cases}$$



如图，从模拟结果可以看出，随机网络的传播性是较强的，在15个时间单位后网络中所有的节点都被感染。

进一步考虑移除的情况，即**人为控制网络恶意代码的传播**，表现为节点有概率γ转化为移除（免疫）节点（取值为0.3）。

$$\begin{cases} \lambda_k = [1-(1-i)^k]\beta \\ \frac{ds}{dt} = -\lambda s - \gamma s, \frac{di}{dt} = \lambda s - \gamma i, \frac{dr}{dt} = \gamma(s+i) \end{cases}$$



如图，从模拟结果可以看出，在有一定人为控制的情况下，感染率在第5个时间单位到达48%的巅峰，随后逐渐趋向零，所有的主机最终都获得免疫。

1.2 无标度网络传播

该情况即1.1.2中节点类型的推广，模拟情况在定性上和1.1.2情况的趋势类似，故不再详细叙述。

2 蠕虫传播模型分类

基于以上对网络恶意代码传播的基础模型，结合网络蠕虫的特性，可以逐渐逼近完善的蠕虫传播模型。

2.1 流行病传播模型

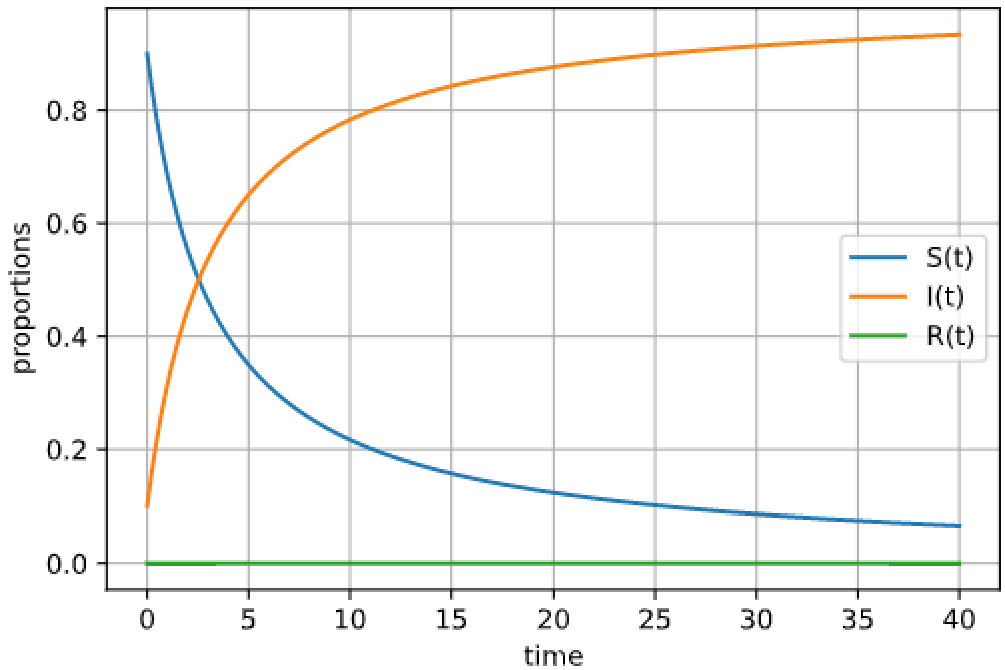
该情况是最简单的互联网蠕虫传染情况，假设系统均匀分布，节点等概率连接，将节点分为可感染状态和感染状态（同1.1.1），其中传染。

$$\begin{cases} \frac{dS}{dt} = -\beta S(t)[N - I(t)] \\ \frac{dI}{dt} = \beta S(t)[N - I(t)] \\ \frac{dR}{dt} = 0 \end{cases}$$

归一化后为：

$$\begin{cases} \frac{ds}{dt} = -\beta s(t)[1 - i(t)] \\ \frac{di}{dt} = \beta s(t)[1 - i(t)] \\ \frac{dr}{dt} = 0 \end{cases}$$

归一化的模拟结果如下：

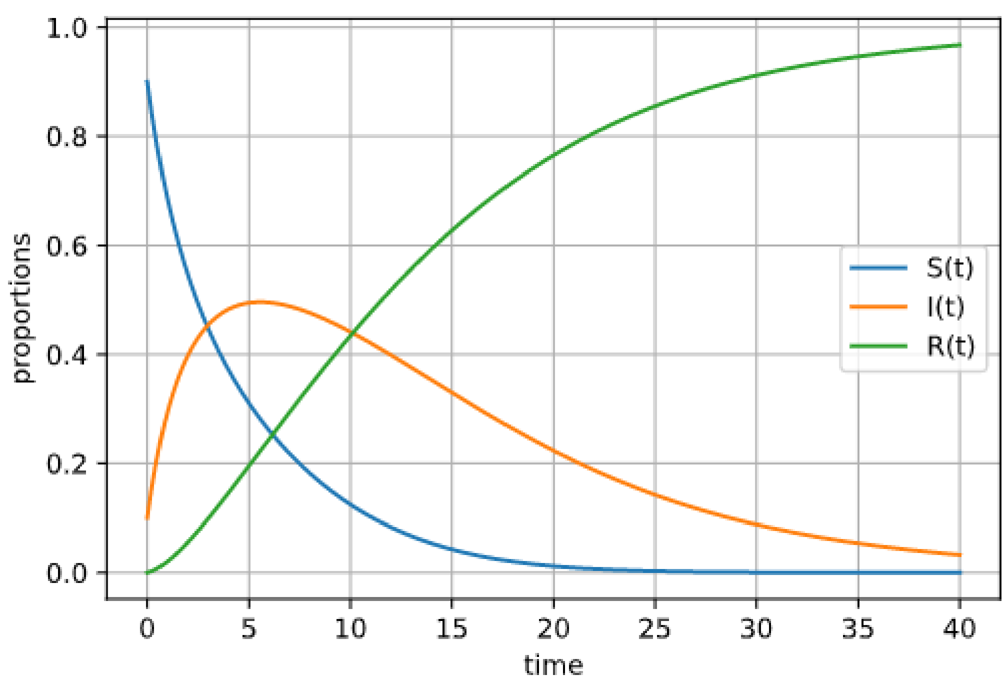


与1.1.1中SI模型对比，可以发现该模型的趋势和速率基本不变，变化的是曲线的平滑度，原因是1.1.1中模型有考虑节点的最大连接数量，而2.1.1中没有考虑k参数。

2.2 KM互联网传播模型

进一步将节点分为可感染状态、感染状态和被移除状态（同1.1.2）。

$$\begin{cases} \frac{di}{dt} = \beta s(t)[1 - i(t)] \\ \frac{dr}{dt} = \gamma i(t) \\ \frac{ds}{dt} = -\frac{di+dr}{dt} \end{cases}$$



与1.1.2中SIR模型对比，在参数一致的情况下模型的趋势和速率仍然基本不变，变化的依旧是是曲线的平滑度。

> 在Internet中应用程序进行通信时，可以认为Internet是个完全连接网络，蠕虫的传播完全受网络的拓扑结构影响，所以可以认为Internet蠕虫符合传染病模型。

> SEIR 模型

在计算机系统中有易感染的程序传入并运行，就会造成恶意代码在系统中的传播，现作如下假设：

- 恶意代码潜伏期短
- 已经被感染的文件具有免疫力
- S类，易感者（Susceptible），指尚未感染的可执行程序
- E类，潜伏者（Exposed），指已经感染的程序，还未开始传播恶意代码
- I类，感病者（Infective），指已经感染的程序，并且开始传播恶意代码
- R类，移出者（Removal），指一定时间内不会运行的可执行程序（文件被隔离或删除）

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate

def plotfuc(func):
    S0 = 0.9
    I0 = 0.1
    R0 = 0.0
    t = np.linspace(0, 40, 10000)
    res = np.array(scipy.integrate.odeint(func, [S0, I0, R0], t,args=(0.35,0.1)))
    plt.figure(figsize=[6, 4])
    plt.plot(t, res[:, 0], label='S(t)')
    plt.plot(t, res[:, 1], label='I(t)')
    plt.plot(t, res[:, 2], label='R(t)')
    plt.legend()
    plt.grid()
    plt.xlabel('time')
    plt.ylabel('proportions')
    plt.show()


def code_1_1_1(y, t, beta, gamma):
    s, i, r= y
    lambda_k = (1-(1-i)**4)*beta
    ds_dt = -lambda_k * s
    di_dt = lambda_k * s
    dr_dt = 0
    return ([ds_dt, di_dt, dr_dt])

def code_1_1_2(y,t,beta, gamma):
    s, i, r = y
    lambda_k = (1-(1-i)**4)*beta
    ds_dt = -lambda_k*s-gamma*s
    di_dt = lambda_k*s-gamma*i
    dr_dt = gamma*(s+i)
    return ([ds_dt, di_dt, dr_dt])

def code_2_1_1(y,t,beta, gamma):
    s, i, r  = y
    ds_dt = -beta*s*(1-i)
    di_dt = beta*s*(1-i)
    dr_dt = 0
    return ([ds_dt, di_dt, dr_dt])

def code_2_1_2(y,t,beta, gamma):
    s, i, r= y
    ds_dt = -beta*s*(1-i)
    di_dt = beta*s*(1-i)-gamma*i
    dr_dt = gamma*i
    return ([ds_dt, di_dt, dr_dt])
```

```
plotfuc(code_1_1_1)

plotfuc(code_1_1_2)

plotfuc(code_2_1_1)

plotfuc(code_2_1_2)
```