传染病模型【gyj】

1. 适用范围

根据传染病的传播肌理预测疾病的传播范围、染病人数等。

2. 常见类型

- 指数传播模型：封闭区域内，不考虑人口迁移，不根据是否患病划分人群
- SI模型：考虑根据是否患病划分人群，不考虑死亡或痊愈，不考虑迁移
- SIS模型：考虑根据是否患病划分人群，考虑病人可被治愈，考虑被治愈后仍可再次感染患病。
- SIR模型：考虑根据是否患病划分人群，考虑病人可被治愈，考虑被治愈后仍可再次感染患病，考虑部分被治愈后对该病拥有免疫力而退出感染系统。

3.1 指数传播模型

3.1.1 模型假设与参数说明

- 所研究的区域是一封闭区域，在一个时期内人口总量相对稳定，不考虑人口的迁移。
- $N(t)$：在t时刻的染病人数
- λ：每个病人在单位时间内传染的人数

3.1.2 推导过程

在 $t \to t+\Delta t$ 时间内，净增加的染病人数为

$$N(t+\Delta t)-N(t)=\lambda N(t)\Delta t$$

若记 $t=0$ 时刻染病人数为 N_0 .则对上式两端同时除以 Δt ,并令 $\Delta t \to 0$,得传染病的人数的微分方程预测模型为

$$\begin{cases}\frac{dN(t)}{dt}=\lambda N(t), & t>0\\ N(0)=N_0\end{cases}$$

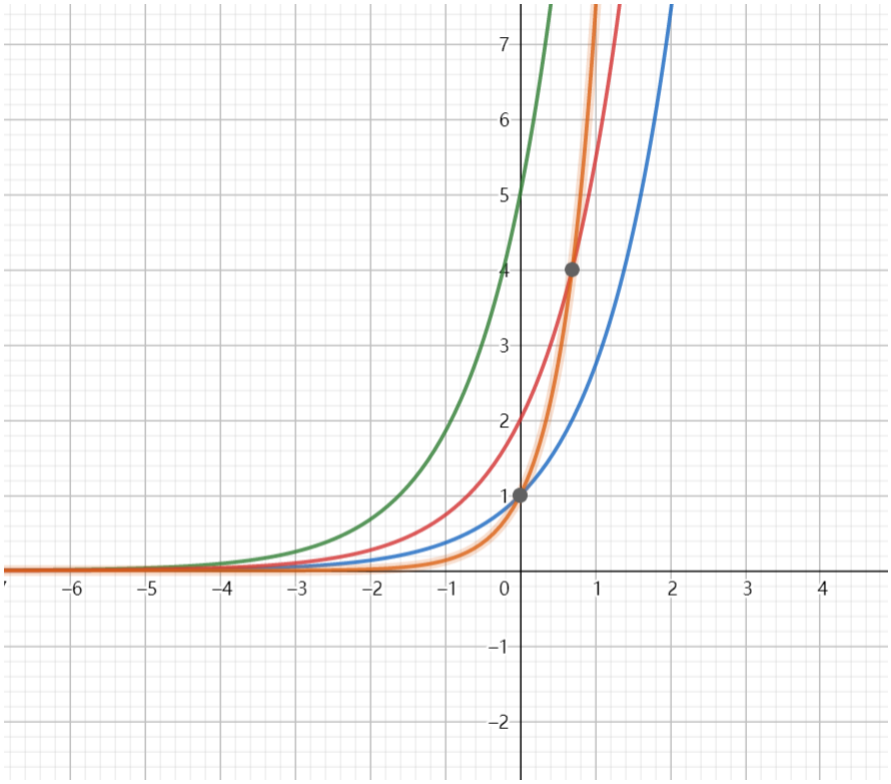
用分离变量的方法求解微分方程，得

$$N(t)=N_0e^{\lambda t}$$

3.1.3 模型表达式

$$N(t)=N_0e^{\lambda t} \tag{1}$$

3.1.4 不同取值的图像



随 t 增加 $N(t)$ 成指数模式增加

3.1.5 模型评价

本模型结果显示传染病的传播是按指数函数增加的。

- 正确的情况：一般在传染病发病初期，对传染病源和传播路径未知，以及没有任何预防控制的措施下。
- 预测失误的情况：事实上，在封闭系统的假设下，区域内人数总量是有限的，故患病人数不可能随时间一直增长。

3.2 SI模型(Susceptible - Infective)

3.2.1 模型假设与参数说明

- 将所有人群分为**易感人群**(Susceptible)和**已感人群**(Infective)，也即健康的人和病人；在传播期内所考察地区的人口总数短期内保持不变，既不考虑生死，也不考虑迁移；每个病人得病后不会痊愈，不会死亡。
- N： 人口总数(一个常数，不随时间改变)
- $s(t)$： t时刻易感人群在总人口中所占的比例
- $i(t)$： t时刻已感人群在总人口中所占的比例
  - $\ s(t)+i(t)=1\ $
- λ： 日感染率，每个病人在单位时间(每天)内接触的平均人数；常数，可使健康者受感染成为病人

3.2.2 推导过程

由参数可知，每个病人每天可使 $\lambda s(t)$ 个健康人变成病人。而t时刻的病人人数为 $Ni(t)$,故在 $t \to t+\Delta t$ 时段内，共有 $\lambda s(t)i(t)N\Delta t$ 个健康者被感染。

$$\frac{Ni(t+\Delta t)-Ni(t)}{\Delta t}=\lambda s(t)i(t)N$$

当 $\Delta t \to 0$ 时，得到微分方程 $\frac{di(t)}{dt}=\lambda s(t)i(t)$.

将 $s(t)=1-i(t)$ 带入上式，得到 $\frac{di(t)}{dt}=\lambda i(t)(1-i(t))$

假设当 $t=0$ 时，病人所占总人口的比例为 $i(0)=i_0$,于是SI 模型可描述为

$$\begin{cases}\frac{di(t)}{dt}=\lambda i(t)(1-i(t)), & t>0\\ i(0)=i_0\end{cases}$$

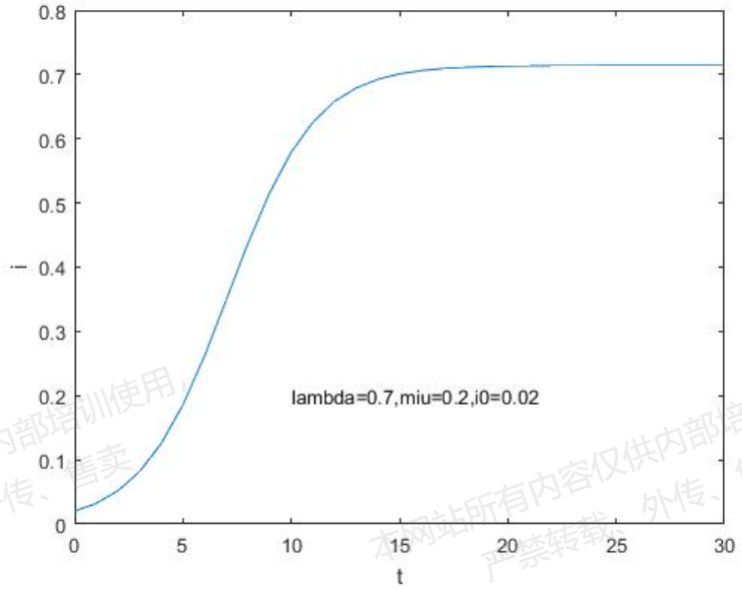
利用分离变量的方式解决次IVP(Initial Valued Problem)问题，得到解析式为

$$i(t)=\frac{1}{1+(\frac{1}{i_0}-1)e^{-\lambda t}}$$

3.2.3 模型表达式

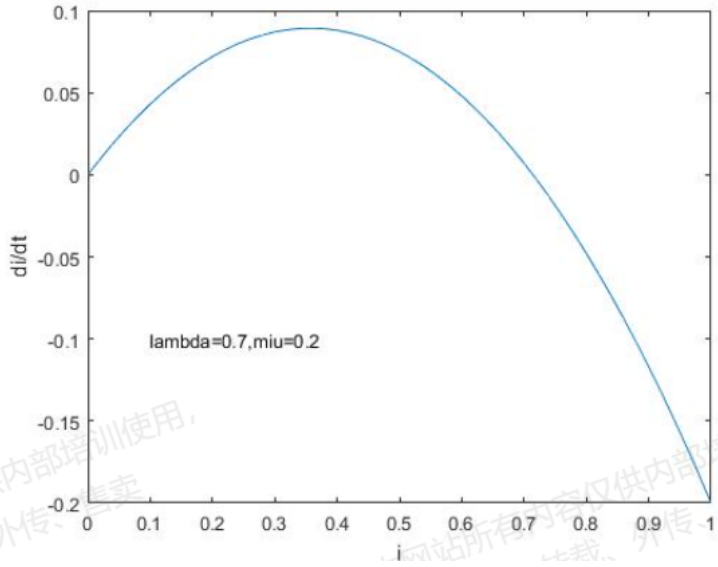
$$i(t)=\frac{1}{1+(\frac{1}{i_0}-1)e^{-\lambda t}} \tag{2}$$

3.2.4 不同取值的图像



3.2.5 模型评价

- 上述图显示，患病人数在总人口数的比例随时间的流逝呈先增加后不变的趋势；通过绘制 $\frac{di(t)}{dt}$ 图，可知传染病人的增加率与时间的关系



由图可知，由于当 $i=\frac{1}{2}$ 时，$\frac{d^2(t)}{dt^2}=0$，故意味着当有50%的人口患病时，此时是传染病传染的高峰期。我们可根据公式(2)算出易得传染病高峰到来的时刻：

$$t_m=\frac{1}{\lambda}ln(\frac{1}{i_0}-1) \tag{2.1}$$

- 公式2.1在医学上具有重要意义：因为 t_m 与 λ 成反比，所以当 λ 增大时(即每个病人可传染的人越多时)，t_m 将会减小，预示着传染病高峰期来的越早。同样，若已知日接触率 λ，即可预报传染病高峰到来的时间 t_m
- 预测失误的情况：公式(2)显示，当 $t\to 0$ 也即 $i(t)\to 1$，预示着群体中的每个人都会生病，这显然不符合实际情况。原因是假设中未考虑病人得病后可以痊愈，人群中的健康者只能变成病人，但病人缺不能变成健康的人，故这一模型的预测结果也不够准确。

3.2.6 代码

python:

```
# -*- coding: utf-8 -*-

import scipy.integrate as spi
import numpy as np
import pylab as pl

beta=1.4247
"""the likelihood that the disease will be transmitted from an infected to a
susceptible
individual in a unit time is 尾"""
gamma=0
#gamma is the recovery rate and in SI model, gamma equals zero
I0=1e-6
#I0 is the initial fraction of infected individuals
ND=70
#ND is the total time step
TS=1.0
INPUT = (1.0-I0, I0)

def diff_eqs(INP,t):
    '''The main set of equations'''
    Y=np.zeros((2))
    V = INP
    Y[0] = - beta * V[0] * V[1] + gamma * V[1]
    Y[1] = beta * V[0] * V[1] - gamma * V[1]
    return Y   # For odeint

t_start = 0.0; t_end = ND; t_inc = TS
t_range = np.arange(t_start, t_end+t_inc, t_inc)
RES = spi.odeint(diff_eqs,INPUT,t_range)
"""RES is the result of fraction of susceptibles and infectious individuals at
each time step respectively"""
print(RES)

#Ploting
pl.plot(RES[:,0], '-bs', label='Susceptibles')
pl.plot(RES[:,1], '-ro', label='Infectious')
pl.legend(loc=0)
pl.title('SI epidemic without births or deaths')
pl.xlabel('Time')
pl.ylabel('Susceptibles and Infectious')
pl.savefig('2.5-SI-high.png', dpi=900) # This does increase the resolution.
pl.show()
```

3.3 SIS模型

3.3.1 模型假设与参数说明

- 将所有人群分为**易感人群**(Susceptible)和**已感人群**(Infective)，也即健康的人和病人；在传播期内所考察地区的人口总数短期内保持不变，既不考虑生死，也不考虑迁移；每个病人得病后会痊愈。病人被治愈后成为仍可被感染的健康者。
- N： 人口总数(一个常数，不随时间改变)
- $s(t)$： t时刻易感人群在总人口中所占的比例
- $i(t)$： t时刻已感人群在总人口中所占的比例
  - $\ s(t)+i(t)=1\ $
- λ： 日感染率，每个病人在单位时间(每天)内接触的平均人数；常数，可使健康者受感染成为病人
- μ： 每天被治愈的病人人数占总病人总数的比例。
- δ： 传染强度
  - $\delta = \frac{\lambda}{\mu}$

3.3.2 推导过程

将SI模型修正为SIS模型

$$\begin{cases} \frac{di(t)}{dt} = \lambda i(t)(1-i(t)) - \mu i(t), & t > 0 \\ i(0) = i_0 \end{cases}$$

通过进行微分方程的求解，可得解析式：

$$i(t) = \begin{cases} [\frac{\lambda}{\lambda-\mu} + (\frac{1}{i_0} - \frac{\lambda}{\lambda-\mu})e^{-(\lambda-\mu)t}]^{-1}, & t > 0 \\ i(0) = i_0 \end{cases}$$

利用 δ 的定义，将上面两式改写为

$$\begin{cases} \frac{di}{dt} = -\lambda i[i - (1 - \frac{1}{\delta})], & t > 0 \\ i(0) = i_0 \end{cases}$$

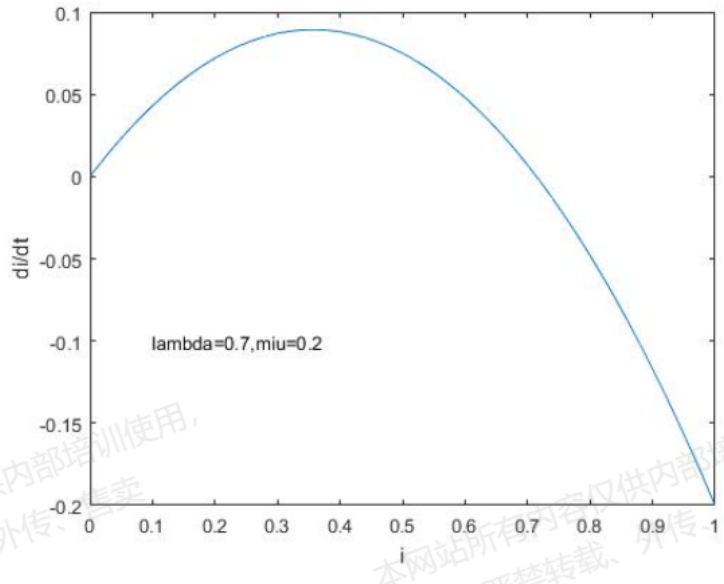
$$i(t) = \begin{cases} [\frac{1}{1-\frac{1}{\delta}} + (\frac{1}{i_0} - \frac{1}{1-\frac{1}{\delta}})e^{-\lambda(1-\frac{1}{\delta})t}]^{-1}, & \delta \neq 1 \\ (\lambda t + \frac{1}{i_0})^{-1}, & \delta = 1 \end{cases}$$

3.3.3 模型表达式

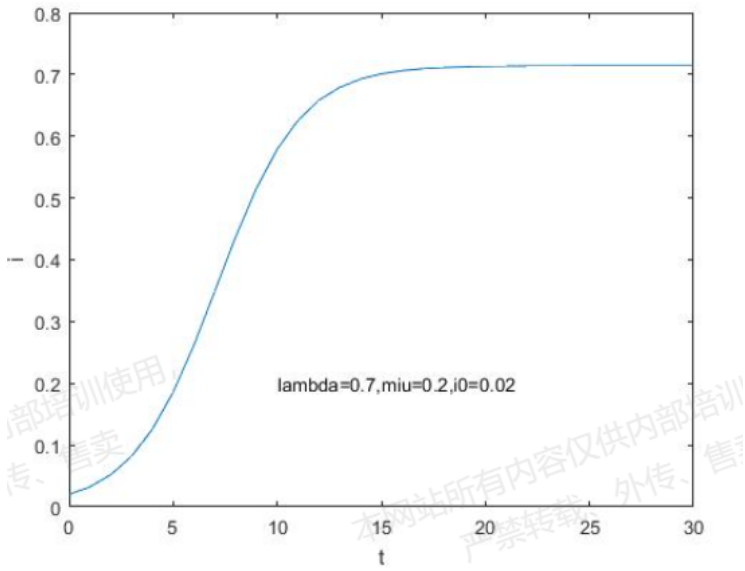
$$i(t) = \begin{cases} [\frac{1}{1-\frac{1}{\delta}} + (\frac{1}{i_0} - \frac{1}{1-\frac{1}{\delta}})e^{-\lambda(1-\frac{1}{\delta})t}]^{-1}, & \delta \neq 1 \\ (\lambda t + \frac{1}{i_0})^{-1}, & \delta = 1 \end{cases} \tag{3}$$

3.3.4 不同取值的图像

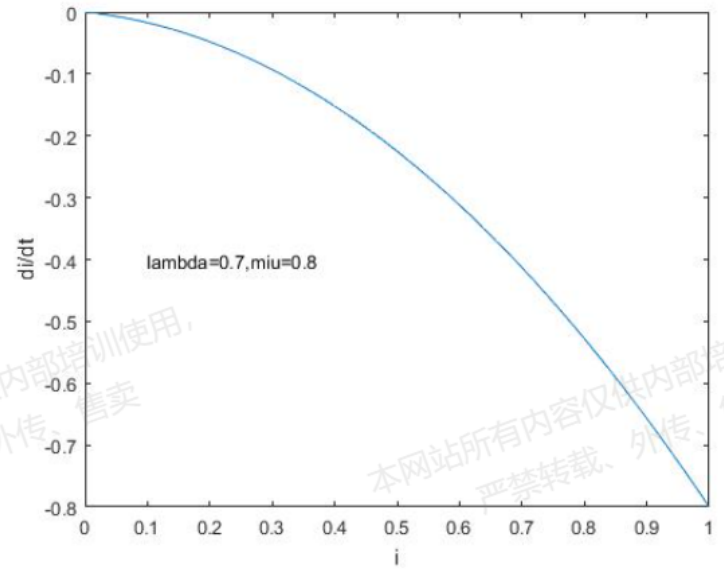
$\delta > 1$ 的 $di/dt - i(t)$ 图:



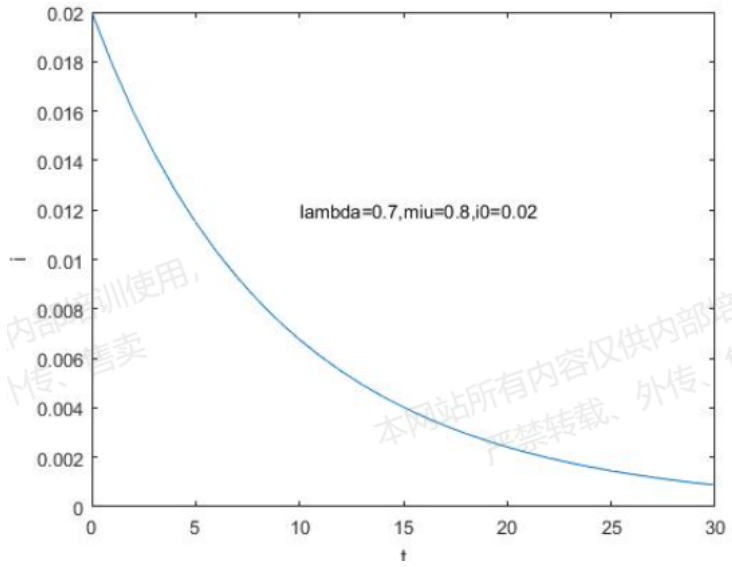
$\delta > 1$ 的 $i(t) - t$ 图:



$\delta < 1$ 的 $di/dt - i(t)$ 图:



$\delta < 1$ 的 $i(t) - t$ 图:



3.3.5 模型评价

- 由公式(3)可知，当 $t \to \infty$ 时，有

$$i(\infty) = \begin{cases} 1 - \frac{1}{\delta}, & \delta > 1 \\ 0, & \delta \leq 1 \end{cases} \tag{3.1}$$

  - 当 $\delta \leq 1$ 时，表示日接触率小于治愈率，故随时间的推移，患病人数比例越来越小，当t趋近于无穷时患病人数清零，最终所有病人都会被治愈。
  - 当 $\delta > 1$ 时，表示日接触率大于治愈率，故随时间的推移，总人口中总有一定比例的人口会被传染而成为病人。
- SIS模型在 $\delta > 1$时的结果与SI模型类似，但本模型额外考虑了病人治愈率与被传染率之间的关系，故更加科学
- 预测会失误的情况：有一些疾病(如水痘、天花等)在患过该疾病后就不容易再感染疾病，本模型并没有考虑到这一种情况。故要引入SIR模型。

3.3.6 代码

python:

```
# -*- coding: utf-8 -*-

import scipy.integrate as spi
import numpy as np
import pylab as pl

beta=1.4247
gamma=0.14286
I0=1e-6
ND=70
TS=1.0
INPUT = (1.0-I0, I0)

def diff_eqs(INP,t):
    '''The main set of equations'''
    Y=np.zeros((2))
    V = INP
    Y[0] = - beta * V[0] * V[1] + gamma * V[1]
```

```
    Y[1] = beta * V[0] * V[1] - gamma * V[1]
    return Y   # For odeint

t_start = 0.0; t_end = ND; t_inc = TS
t_range = np.arange(t_start, t_end+t_inc, t_inc)
RES = spi.odeint(diff_eqs,INPUT,t_range)

print(RES)

#Ploting
pl.plot(RES[:,0], '-bs', label='Susceptibles')
pl.plot(RES[:,1], '-ro', label='Infectious')
pl.legend(loc=0)
pl.title('SIS epidemic without births or deaths')
pl.xlabel('Time')
pl.ylabel('Susceptibles and Infectious')
pl.savefig('2.5-SIS-high.png', dpi=900) # This does increase the resolution.
pl.show()
```

3.4 SIR模型

3.4.1 模型假设与参数说明

- 将所有人群分为**易感人群**(Susceptible)、**已感人群**(Infective)、和**病愈后因具有免疫力而退出系统者**(Remove).在传播期内所考察地区的人口总数短期内保持不变，不考虑迁移；每个病人得病后会痊愈。病人被治愈后成为仍可被感染的健康者。
- N： 人口总数(一个常数，不随时间改变)
- $s(t)$： t时刻易感人群在总人口中所占的比例
- $i(t)$： t时刻已感人群在总人口中所占的比例
- $r(t)$： t时刻具有免疫力而退出系统的人在总人口中所占比例
  - $s(t)+i(t)+r(t)=1$
- λ： 日感染率，每个病人在单位时间(每天)内接触的平均人数；常数，可使健康者受感染成为病人
- μ： 日治愈率，每天被治愈的病人人数占总病人总数的比例。
- δ： 传染强度
  - $\delta=\frac{\lambda}{\mu}$

3.4.2 推导过程

对所有人群，有 $s(t)+i(1)+r(t)=1$

对系统移出者，有$\ N\frac{dr}{dt}=\mu Ni\ $

对病人，有 $N\frac{di}{dt}=\lambda Nsi-\mu Ni$

对健康者，有 $N\frac{ds}{dt}=-\lambda Nsi$

联立上述公式，可获得SIR模型

$$\begin{cases}\frac{di}{dt}=\lambda si-\mu i\\ \frac{ds}{dt}=-\lambda si\\ \frac{dr}{dt}=\mu i\\ i(0)=i_0,s(0)=s_0,r(0)=r_0\end{cases}$$

3.4.3 模型表达式

$$\begin{cases} \frac{di}{dt} = \lambda si - \mu i \\ \frac{ds}{dt} = -\lambda si \\ \frac{dr}{dt} = \mu i \\ i(0) = i_0, s(0) = s_0, r(0) = r_0 \end{cases} \tag{4}$$

3.4.4 模型评价

- SIR模型是一个典型的系统力学模型，其突出特点是模型形式为关于多个相互关联的系统变量之间的常微分方程组。
    - 类似的问题包括：河流中水体各类污染物质的好氧、复氧、反应、迁移、吸附、沉降等；食物在人体中的分解、吸收、排泄；污水处理过程中的污染物降解、微生物和细菌的增长或衰减等。
- 此类问题很难求得解析式，可以使用软件求数值解。

3.4.5 代码

python:

```
# -*- coding: utf-8 -*-

import scipy.integrate as spi
import numpy as np
import pylab as pl

beta=1.4247
gamma=0.14286
TS=1.0
ND=70.0
S0=1-1e-6
I0=1e-6
INPUT = (S0, I0, 0.0)

def diff_eqs(INP,t):
    '''The main set of equations'''
    Y=np.zeros((3))
    V = INP
    Y[0] = - beta * V[0] * V[1]
    Y[1] = beta * V[0] * V[1] - gamma * V[1]
    Y[2] = gamma * V[1]
    return Y   # For odeint

t_start = 0.0; t_end = ND; t_inc = TS
t_range = np.arange(t_start, t_end+t_inc, t_inc)
RES = spi.odeint(diff_eqs,INPUT,t_range)

print(RES)

#Ploting
pl.plot(RES[:,0], '-bs', label='Susceptibles')  # I change -g to g--  #
RES[:,0], '-g',
pl.plot(RES[:,2], '-g^', label='Recovereds')  # RES[:,2], '-k',
pl.plot(RES[:,1], '-ro', label='Infectious')
pl.legend(loc=0)
pl.title('SIR epidemic without births or deaths')
pl.xlabel('Time')
```

```
pl.ylabel('Susceptibles, Recovereds, and Infectious')
pl.savefig('2.1-SIR-high.png', dpi=900) # This does, too
pl.show()
```

matlab:

```
%%传染病SIR模型
clear;clc
%日接触率、日治愈率均定义在ill.m文件中

ts=0:50;              %天数范围
x0=[0.02,0.98];       %初始感染人群比例

[t,x]=ode45('ill',ts,x0);
[t,x]

plot(t,x(:,1),t,x(:,2)),grid
legend('感染者','健康者')
figure
plot(x(:,2),x(:,1)),grid
title('相轨线')

%需要另外一个'ill.m'来实现ill
function y=ill(t,x)

a=1;    %日接触率
b=0.3;  %日治愈率

y=[a*x(1)*x(2)-b*x(1),-a*x(1)*x(2)]';
```

C++:

```
/*************************************************************************
传染病SIR模型
*************************************************************************/
#include<stdio.h>
#include<math.h>
#define steplength 1 //步长
#define FuncNumber 2 //微分方程数
#define L 1 //日接触率
#define U 0.3   //日治愈率
int main()
{
    float x[200],Yn[20][200],reachpoint;
    int i;
    x[0]=0;          //起点
    Yn[0][0]=0.02;
    Yn[1][0]=0.98;         //初值条件
    reachpoint=30;       //终点
    void rightfunctions(float x ,float *Auxiliary,float *Func);
    void Runge_Kutta(float *x,float reachpoint, float(*Yn)[200]);
    Runge_Kutta(x ,reachpoint, Yn);
    printf("date\t\tinfective\tsusceptible\n");
    for(i=0;i<=(reachpoint-x[0])/steplength;i++){
        printf("%f\t",x[i]);
```

```
        printf("%f\t",Yn[0][i]);
        printf("%f \n",Yn[1][i]);
    }
    return 0;
}
void rightfunctions(float x ,float *Auxiliary,float *Func)//微分方程组
{
    Func[0]=L*Auxiliary[0]*Auxiliary[1]-U*Auxiliary[0];
    Func[1]=-1*L*Auxiliary[0]*Auxiliary[1];
}
void Runge_Kutta(float *x,float reachpoint, float(*Yn)[200])
{
    int i,j;
    float Func[FuncNumber],K[FuncNumber][4],Auxiliary[FuncNumber];
        for(i=0;i<=(reachpoint-x[0])/steplength;i++)
    {
        for(j=0;j<FuncNumber;j++)
            Auxiliary[j]=*(Yn[j]+i);
        rightfunctions(x[i],Auxiliary,Func);
        for(j=0;j<FuncNumber;j++)
        {
            K[j][0]=Func[j];
            Auxiliary[j]=*(Yn[j]+i)+0.5*steplength*K[j][0];
        }
        rightfunctions(x[i],Auxiliary,Func);
        for(j=0;j<FuncNumber;j++)
        {
            K[j][1]=Func[j];
            Auxiliary[j]=*(Yn[j]+i)+0.5*steplength*K[j][1];
        }
        rightfunctions(x[i],Auxiliary,Func);
        for(j=0;j<FuncNumber;j++)
        {
            K[j][2]=Func[j];
            Auxiliary[j]=*(Yn[j]+i)+steplength*K[j][2];
        }
        rightfunctions(x[i],Auxiliary,Func);
        for(j=0;j<FuncNumber;j++)
            K[j][3]=Func[j];
        for(j=0;j<FuncNumber;j++)
        Yn[j][i+1]=Yn[j][i]+(K[j][0]+2*K[j][1]+2*K[j][2]+K[j]
[3])*steplength/6.0;
        x[i+1]=x[i]+steplength;
    }

}
```

4. 参考资料

1. 数模官网 - 传染病模型
2. 《数学建模算法与应用》书p.153~157