武漢理工大學

数学建模暑期培训论文

第 1 题

基于 xxxxxxxx 模型

第 10 组

姓名	方向
刘子川	编程
程宇	建模
祁成	写作

2020 年 8 月 20 日

摘要

控制高压油管的压力变化对减小燃油量偏差，提高发动机工作效率具有重要意义。本文建立了基于质量守恒定理的微分方程稳压模型，采用二分法、试探法以及自适应权重的蝙蝠算法对模型进行求解。

针对问题一，针对问题一，建立 SEIR 模型并使用免疫差分进化算法估计模型参数，最后调整模型参数以分析卫生措施效果。本节首先根据新冠病毒疫情改进了传统的 SEIR 模型，并通过免疫差分进化算法拟合预测结果与实际统计数据，估计未统计模型参数。根据美国实施居家令和提前或延后湖北省采用的严格隔离措施调整模型参数，分析预测结果的变化规律，可得出——提前采取严格隔离将大幅度降湖北省低疫情损失，反之将大幅度提高损失；若即时实施居家令将大幅度降低美国疫情损失。

针对问题二，建立基于质量守恒定律的泵-管-嘴系统动态稳压模型，将燃油进入和喷出的过程动态化处理。考虑柱塞和针阀升程的动态变动，建立喷油嘴流量方程和质量守恒方程。为提高角速度求解精度，以凸轮转动角度为固定步长，转动时间变动步长，采用试探法粗略搜索与二分法精细搜索的方法求解，求得凸轮最优转动角速度 **0.0283rad/ms（转速 270.382 转/分钟）**，并得到该角速度下高压油管的密度、压力周期性变化图。对求解结果进行误差分析与灵敏度分析，考察柱塞腔残余容积变动对高压油管压力稳态的影响。

针对问题三，对于增加一个喷油嘴的情况，改变质量守恒方程并沿用问题二的模型调整供、喷油策略，得到最优凸轮转动角速度为 **0.0522rad/ms（498.726 转/分钟）**；对于既增加喷油嘴又增加减压阀的情况，建立基于自适应权重的蝙蝠算法的多变量优化模型，以凸轮转动角速度、减压阀开启时长和关闭时长为参数，平均绝对偏差 MAD 为目标，在泵-管-嘴系统动态稳压模型的基础上进行求解，得到最优参数：**角速度 0.0648 rad/ms（619.109 转/分钟）**、减压阀的开启时长 **2.4ms** 和减压阀的关闭时长 **97.6ms**。

本文的优点为：1. 采用试探法粗略搜索与二分法精细搜索结合的方法，降低了问题的求解难度。2. 以凸轮转动角度为固定步长，对不同角速度按照不同精度的时间步长求解，大大提高了求解的精确度。3. 针对智能算法求解精度方面，采用改进的蝙蝠算法，使速度权重系数自适应调整，兼顾局部搜索与全局搜索能力。

关键词： 微分方程　微分方程　微分方程　微分方程

目录

1 问题重述

1.1 问题背景

新型冠状病毒肺炎（Corona Virus Disease 2019，COVID-19），简称“新冠肺炎”，世界卫生组织命名为“COVID-19”，是指 2019 新型冠状病毒感染导致的肺炎。2020 年 3 月 11 日，世界卫生组织总干事谭德塞宣布，世卫组织认为当前新冠肺炎疫情可被称为全球大流行（pandemic）。目前，COVID-19 疫情仍在世界各地蔓延，已超过 1630 万人感染，65 万余人死亡，给世界各国的经济发展和人民生活带来了极大影响，甚至从一定程度上改变了人类的工作生活方式。

在当前信息快速传播的社会中，在传染病传播周期内，人群一般都会经历“不重视-自我保护”2 个阶段：在第一阶段，由于群众对疾病具体情况不知情, 对疫情的认知程度有限，对疾病传播缺乏有效的防备措施. 在此阶段中的疾病传播过程可认为符合基本再生数为常数的经典 (Susceptible-Exposed-Infectious-Recovered，SEIR) 模型[1]。在第二阶段，由于政府和群众高度重视，各种防治措施与资源被相继引入[2]，加上对疾病传播途径已有较为准确的认知，人群会逐渐采取有效且科学的防控措施，使得基本再生数不断下降，直到其下降到小于 1，并持续一段时间，最终达到对疾病的根除。

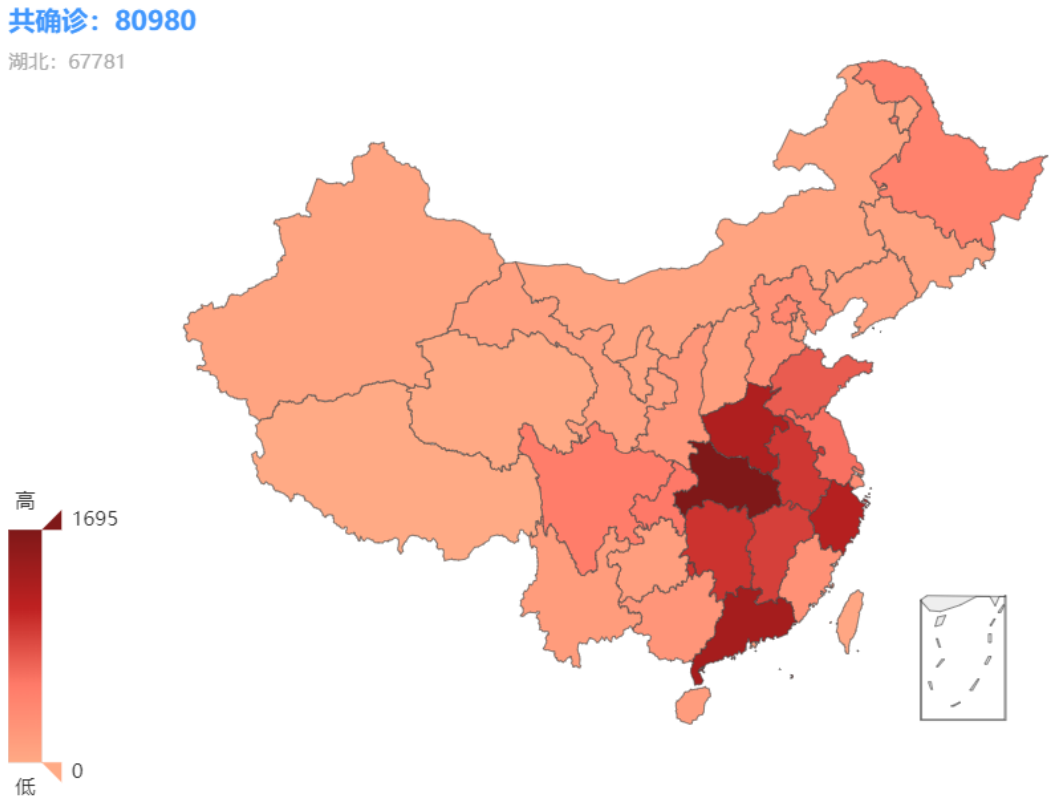


图 **1**　新型冠状病毒肺炎在中国蔓延情况

1.2 问题概述

围绕相关附件和条件要求，定量地研究传染病的传播规律，利用所给（不限于）资料和数据，作出预测并给出控制传染病蔓延的对策建议，具体要求如下：

问题一：建立模型，预测不同国家或地区（至少预测两个国家或地区）确诊病例和死亡病例数的变化。

问题二：收集 COVID-19 对经济某个方面影响的数据（须说明数据获取方式或来源），建立相应的数学模型并进行预测。

问题三：结合你们模型和预测数据，给相关国家或地区的卫生部门写一篇短文，对该国或该地区的疾病防控给出对策建议。

2 模型假设

(1) 假设所有感染者与潜伏者分别具有相同的疾病传播能力，且感染者的传染能量强于潜伏者。

(2) 假设实施严格的隔离措施与实施居家令分别能有效的提高隔离比例和降低人与之间的接触次数。

(3) 假设所有康复者具有病毒抗体，不会出现二次感染的情况。

(4) 假设股票数据真实可靠，并且所选公司能反映国家的经济情况。

3 符号说明

符号	说明
S	易感者的累计总数
E	潜伏者的累计总数
I	感染者的累计总数
S_q	隔离易感者的累计总数
E_q	隔离潜伏者的累计总数
H	住院患者的累计总数
α	有效接触率
β	传染率
ρ	有效接触系数
d	病死率

注：表中未说明的符号以首次出现处为准

4 问题一模型的建立与求解

4.1 问题描述与分析

问题一要求建立至少两个地区的确诊病例数与死亡数的预测模型，并基于模型对这些地区的卫生部门所采取的措施做出评论。收集整理文献可知[2, 3]，灰度预测、时间序列等完全基于数据的预测模型难以更改环境参数，即难以得出卫生部门采取措施对疫情数据的影响。故本节使用可灵活调整参数的动力学模型预测确诊病例数与死亡数的变化趋势。

本文改进了动力学模型中的 SEIR 模型，基于新冠病毒的特点，添加无症状感染者元素，且使得潜伏者具有传染特性。即总方程组中包含易感者、隔离易感者、潜伏者、隔离潜伏者、感染者以及隔离感染者与无症状感染者等变量。针对方程中的未统计参数，如接触感染率与潜伏者数量等参数，本文使用免疫差分进化算法拟合预测曲线与统计数据以求解该类未知参数。

用改进的动力学模型预测湖北省与美国的确诊病例数与死亡数后，我们通过修改模型参数以探讨当地卫生部门采取的措施对疫情趋势的影响。针对湖北省，我们分别提前和延后 5 天调整模型的隔离比例以分析提前或延后采取严格的隔离措施，对疫情传播所造成的影响。针对美国，我们通过调整日平均接触人数以讨论实施或取消居家令和就地避难令对疫情传播的影响。最后给予部分参数细微震荡，对整体模型进行灵敏度分析。

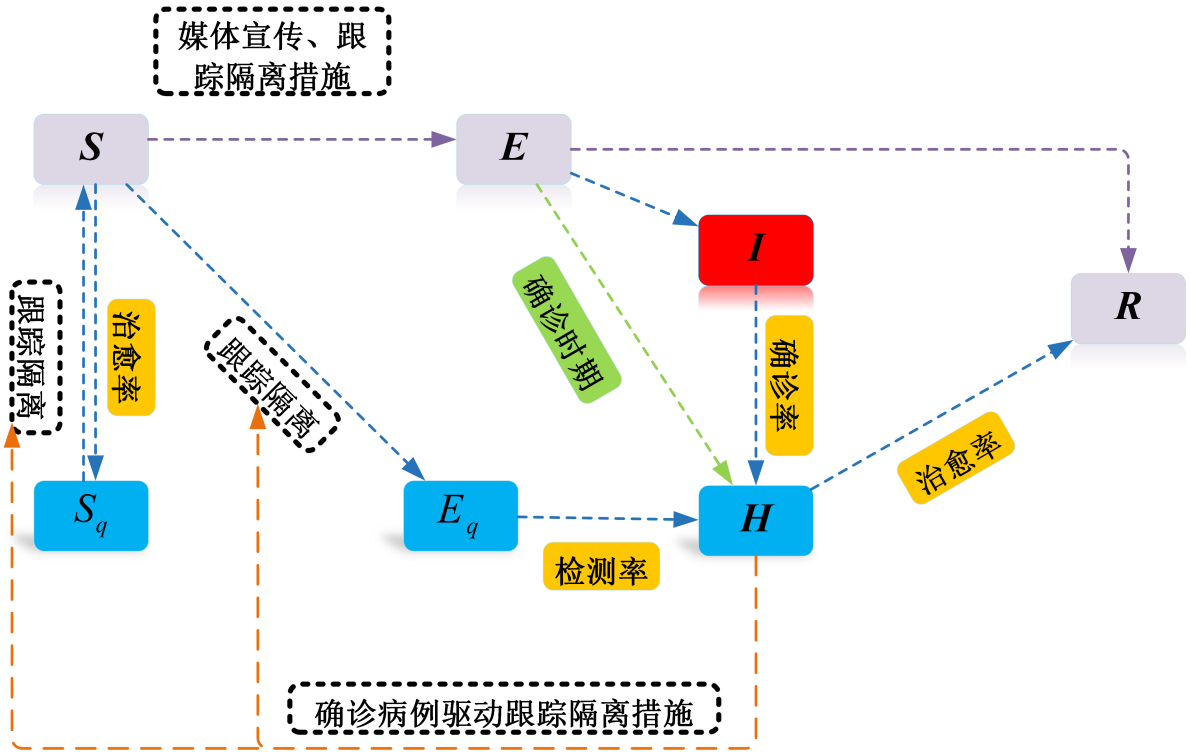


图 2　问题一人群转化关系图

4.2 传染病动力学模型

本节介绍改进的 $SEIR$ 模型. 研究对象是感染者、潜伏者、易感者、痊愈者等，我们使用如下记号来代表每个人群的人数:

- $S(t)$：t 时刻易感者的累计总数；
- $E(t)$：t 时刻潜伏者的累计总数；
- $I(t)$：t 时刻感染者的累计总数；
- $S_q(t)$：t 时刻隔离易感者的累计总数；
- $E_q(t)$：t 时刻隔离潜伏者的累计总数；
- $H(t)$：t 时刻住院患者的累计总数；
- $R(t)$：t 时刻痊愈者的累计总数。

模型有以下前提：

1. 潜伏者在出现明显症状前会经历 7 天的潜伏期, 一旦出现症状，潜伏者将寻求治疗，从而转为确诊的感染者；

2. 由于政府干预控制措施，部分感染者在潜伏期内尚未出现症状已被隔离，成为隔离潜伏者，在被隔离了平均 14 天后出现症状，成为确诊的感染者。

可建立模型如下：

定义有效接触率 α、传染率 β 和有效接触系数 ρ，α 是易感人群在随机混合人群中的占比；β 是有效接触。分析可知易感者 S 有三种转化途径：向隔离易感者 S_q、隔离潜伏者 E_q 和潜伏者 E 的转化速率 (单位时间 Δt 内，转化数量 Δn 与同类群体的个数 n 的比值) 分别为 $\rho(1-\beta)q$，$\rho\beta q$ 和 $\rho\beta(1-q)$。此外，确认隔离期 t_d 无症状后，隔离易感者 S_q 也可向易感者 S 以 λS_q 的速率转化。由以上分析可建立易感者 S 转化方程：

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\alpha\rho[\beta + q(1-\beta)]S(I+\theta E) + \lambda S_q, \tag{1}$$

其中，θ 是潜伏者相对于感染者传播能力的比值。$\lambda = 1/14$ 是隔离解除速率，数值取隔离期的倒数。

潜伏者可以向感染者转化，易感者可向潜伏者转化，可列出潜伏者 S 的转化方程：

$$\frac{\mathrm{d}E}{\mathrm{d}t} = \alpha\rho\beta(1-q)(I+\theta E) - \sigma E, \tag{2}$$

其中，σ 为潜伏者向感染者的转化速率。

潜伏者可以转化为感染者，感染者的流向有死亡、被治愈和被隔离。定义病死率 d、感染者恢复率 ς_I 和隔离速率 σ_I，对感染者有：

$$\frac{\mathrm{d}I}{\mathrm{d}t} = \sigma E - (\sigma_I + d + \varsigma_I)I. \tag{3}$$

对于被隔离的群体，隔离易感者 S_q 与易感者相互转化；隔离潜伏者 E_q 来源于易感者，可转化为隔离感染者。定义 δ_q 是隔离潜伏者向隔离感染者的转化速率，则有

$$\frac{\mathrm{d}S_q}{\mathrm{d}t} = \alpha\rho q(1-\beta)(I+\theta E) - \lambda S_q, \tag{4}$$

$$\frac{\mathrm{d}E_q}{\mathrm{d}t} = \rho\alpha\beta q(I+\theta E) - \delta_q E_q. \tag{5}$$

对住院患者，感染者和隔离的潜伏者向住院患者的转化速率分别是 δ_I 和 δ_q，住院患者流向为死亡和康复，对应系数为死亡率 d 和住院患者恢复率 ς_H。

$$\frac{\mathrm{d}H}{\mathrm{d}t} = \delta_I I + \delta_q E_q - (d+\varsigma_H)H. \tag{6}$$

对死亡病例，有

$$\frac{\mathrm{d}D}{\mathrm{d}t} = D + d(H+I). \tag{7}$$

最后，痊愈者来源有感染者和住院患者，故对痊愈者有

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \varsigma_I I + \varsigma_H H. \tag{8}$$

模型的总表达为：

$$\begin{cases} \dfrac{\mathrm{d}S}{\mathrm{d}t} = -\alpha\rho[\beta+q(1-\beta)]S(I+\theta E) + \lambda S_q, \\ \dfrac{\mathrm{d}E}{\mathrm{d}t} = \alpha\rho\beta(1-q)(I+\theta E) - \sigma E, \\ \dfrac{\mathrm{d}I}{\mathrm{d}t} = \sigma E - (\sigma_I + d + \varsigma_I)I, \\ \dfrac{\mathrm{d}S_q}{\mathrm{d}t} = \alpha\rho q(1-\beta)(I+\theta E) - \lambda S_q, \\ \dfrac{\mathrm{d}E_q}{\mathrm{d}t} = \rho\alpha\beta q(I+\theta E) - \delta_q E_q, \\ \dfrac{\mathrm{d}H}{\mathrm{d}t} = \delta_I I + \delta_q E_q - (d+\varsigma_H)H, \\ \dfrac{\mathrm{d}R}{\mathrm{d}t} = \varsigma_I I + \varsigma_H H, \\ \dfrac{\mathrm{d}D}{\mathrm{d}t} = D + d(H+I). \end{cases} \tag{9}$$

4.3 免疫差分进化算法

本文设计免疫差分进化算法估计微分方程组中的未知参数，定义决策向量为

$$X = [x_1, x_2, x_3, x_4], \tag{10}$$

其中 x_1、x_2、x_3 和 x_4 分别表示每个患者的日平均接触人数、接触感染概率、初始潜伏者数量以及潜伏者相对于感染者传播能力的比值。将目标函数定义为损失函数如下

$$minLoss(X) = \sum_{t=1}^{T}(|\frac{D_r(t) - D(t)}{D_r(t)}| + |\frac{R_r(t) - R(t)}{R_r(t)}| + |\frac{H_r(t) - H(t)}{H_r(t)}|), \tag{11}$$

其中 T 表示选取数据的终止节点，即表示选取用于估计参数的数据来自疫情发生的第 1 天到第 T 天。$D_r(t)$、$R_r(t)$ 与 $H_r(t)$ 分别表示疫情发生后第 t 天的死亡人数、治愈人数和医院患者人数的真实数据；$D(t)$、$R(t)$ 与 $H(t)$ 分别表示其对应的由 SEIR 模型。损失函数 $Loss$ 表示预测结果与实际结果间的距离，即 $Loss$ 值越小，预测曲线就与真实曲线越接近。

种群初始化 在解空间中随机产 p 个初始个体 $X_i(0) = [x_1, x_2, x_3, x_4], (i = 1, 2, 3, \cdots, p)$. 其中第 i 个个体的第 j 维取值方式如下

$$x_{i,j}(0) = x_{j,min} + rand(0,1)(x_{j,max} - x_{j,min}),$$
$$i = 1, 2, 3, \cdots, p, j = 1, 2, 3, 4,$$

其中 p 表示种群规模，$x_{j,max}$ 和 $x_{j,min}$ 分别表示决策变量 X 第 j 维的取值范围上界与下界。

变异 在第 g 次迭代中，生成变异个体 $H_i(g)$，从种群中随机选取三个个体 $X_{p1}(g)$,$X_{p2}(g)$ 和 $X_{p3}(g)$，且 $p_1 \neq p_2 \neq p_3 \neq i$，生成的变异向量为

$$H_i(g) = X_{p1}(g) + F(g) * (X_{p2}(g) - X_{p3}(g)),$$

$F(g) \in (0, 1)$ 是每一代中的放缩因子，其服从柯西分部如下

$$F(g) = cauchyrnd(uF, 0.1),$$

其中 uF 是 F 的期望值，本文取值为 $uF = 0.5$。

交叉 对第 g 代种群中第 i 个体进行交叉操作，生成交叉个体 $V_i(g)$，具体表达式如下：

$$v_{i,j} = \begin{cases} h_{i,j}(g), rand(0,1) \leq cr_i, \\ x_{i,j}(g), rand(0,1) > cr_i, \end{cases} \tag{12}$$

其中 $cr_i \in [0.1, 0.6]$ 是个体 i 的交叉概率，参数 cr_i 将进行自适应调整，具体表达式如下：

$$cr_i = \begin{cases} cr_l + (cr_u - cr_l)\dfrac{Loss_i - Loss_{min}}{Loss_{max} - Loss_{min}}, Loss_i > \overline{Loss}, \\ cr_l, Loss_i \leqslant \overline{Loss}. \end{cases} \tag{13}$$

免疫选择 混合第 g 代的交叉个体 $V(g)$ 与原始个体 $X(g)$，得到待选组 $\{X'(g+1)\}$ 如下

$$X'_i(g+1) = \begin{cases} X_i(g), i \leqslant p, \\ V_{i-p}(g), i > p. \end{cases}$$

个体 $X'_a(g+1)$ 和 $X'_b(g+1)$ 的亲和度 $S_{a,b}$ 可表示为

$$S_{a,b} = \sqrt{\sum_{i=1}^{4}(\frac{x_{i,a} - x_{i,b}}{x_{i,max} - x_{i,min}})^2}, \tag{14}$$

$S_{a,b}$ 为 $X'_a(g+1)$ 和 $X'_b(g+1)$ 的归一化距离，表示个体 $X'_a(g+1)$ 和 $X'_b(g+1)$ 的相似性。定义个体 $X'_i(g+1)$ 的抗体浓度为 C_i, 即

$$C_i = \frac{1}{2p}\sum_{j=1}^{2p} N_{i,j},$$

$$N_{i,j} = \begin{cases} 1, S_{i,j} \geqslant \mu, \\ 0, S_{i,j} < \mu, \end{cases}$$

$\mu(\mu \in [0,1])$ 为相似度阈值，即当个体 i 和 j 的亲和度 $S_{i,j} \geqslant \mu$ 时认为个体 i 和 j 为相似个体。C_i 即为 $\{X'(g+1)\}$ 中 $X'_i(g+1)$ 的相似个体所占比例，C_i 越大即表示 $X'_i(g+1)$ 所在区域的个体密度越大。我们优先将损失函数 $Loss$ 值最优的前 σ 个解放入下一代个体 $\{X(g+1)\}$ 中以防止最优解丢失。再计算剩余个体的复合适应度函数，即个体 i 的复合适应度函数可表示为

$$minF(X'_i(g+1)) = \frac{Loss(X'_i(g+1)) - Loss_{min}}{Loss_{max} - loss_{min}} + C_i \tag{15}$$

即选取复合适应度函数 F 较优的剩余 $p - \sigma$ 个个体放入下一代个体 $\{X(g+1)\}$ 中。重

复迭代上述算法 G 次后终止算法并输出最优参数集 X_{best}。

4.4 实验结果及分析

本文先后采取湖北和美国的数据预测确诊病例和死亡病例数的变化。并设立在 **2020 年 1 月 22 日**时两地的初值统计参数如下表所示

表 **1**　模型统计前初值参数

参数名称	湖北省	美国	来源
人口总数	59170000	310000000	goolge
感染者	786	3	goolge
尚在接受医学观察的人数	2776	124	参考文献[2]
正在被隔离的潜伏者	400	10	估计值
正在住院的患者	1186	20	参考文献[2]
出院人数	31	0	google
死亡人数	3	0	百度

使用免疫差分进化算法可求得湖北省与美国的未统计参数如表 2 所示，其中初始潜伏者数量的步长为 25 进行遍历。

表 **2**　免疫差分进化算法求得的未统计参数

参数名称	湖北省	美国
日平均接触人数	4.13	14.28
接触感染概率	0.4384	0.6851
初始潜伏者数量	125	1100
潜伏者与患者感染能力比	1.00	1.00

根据该参数求得预测参数与实际统计参数可解得湖北省数据预测曲线如图 3 所示，其中取隔离解除速率 $\lambda = \frac{1}{14}$，潜伏者向感染者的转化速率 $\sigma = \frac{1}{7}$。取 1 月 22 日前，即严格的隔离措施执行前隔离比例 $q = 0.2$，取执行之后的隔离比例为 $q = 0.95$。

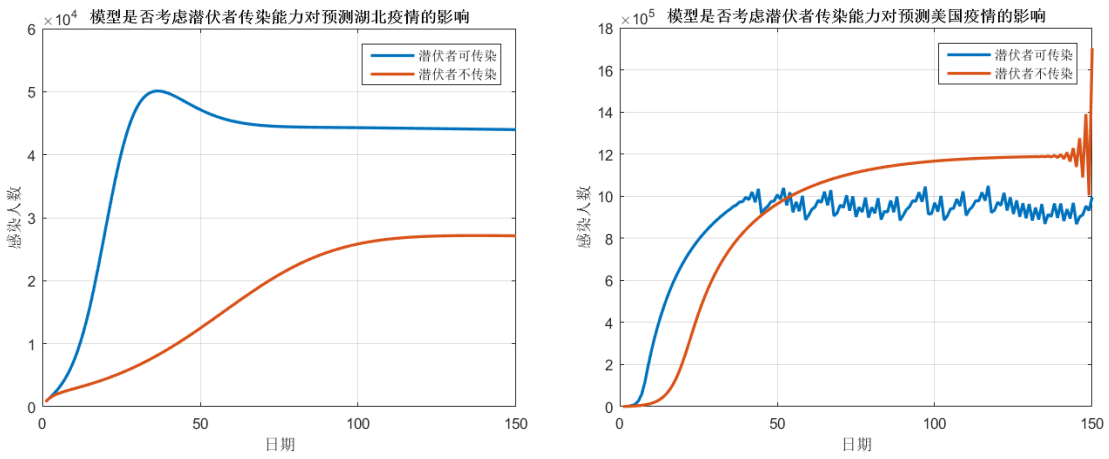
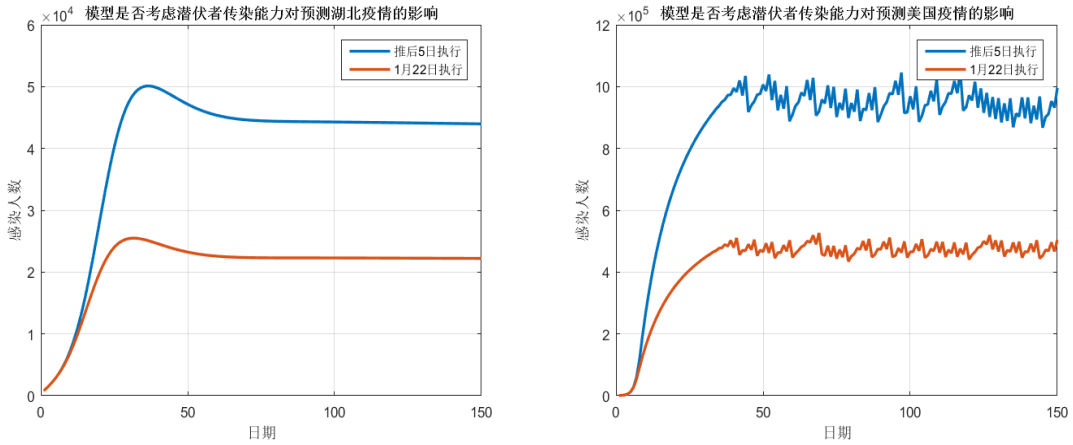


图 **3**　湖北省和美国两地的感染者预测曲线

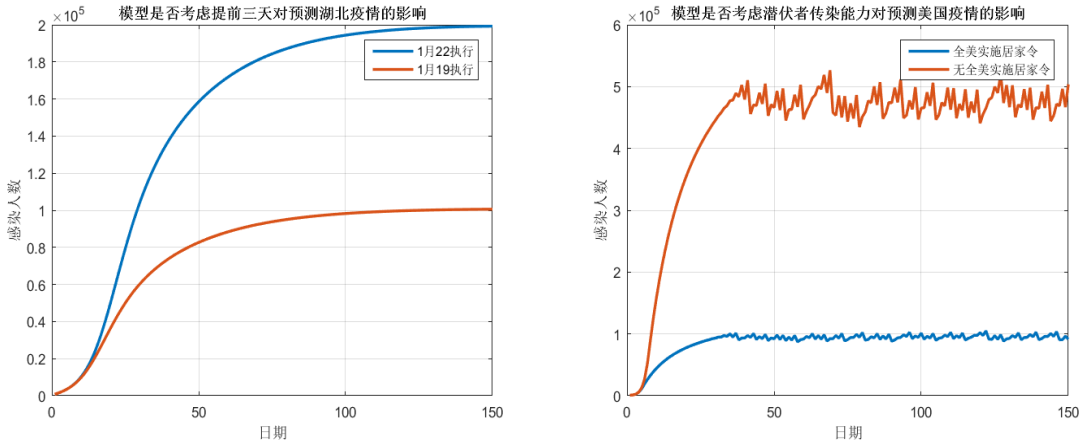
若使严格隔离措施推后 5 日执行，即使得 2 月 1 日前隔离比例为 $q = 0.2$，之后的隔离比例为 $q = 0.99$，可得预测曲线如图 4 所示



(a) 湖北推后 5 日执行　　(b) 美国推后 5 日执行

图 **4**　湖北省和美国两地懈怠措施的感染者预测曲线

由图可知，推后 5 日执行严格隔离措施将使得疫情高峰于 **50 天后**到来。最高感染人数将分别达到 **5 万人/120 万人**，是原始数据的**两倍**左右。死亡人数将达到 **3 千/14 万人**，是原始数据**的 3 倍**。若使严格隔离措施提前 3 日执行，即使得 1 月 19 日前隔离比例为 $q = 0.2$，之后的隔离比例为 $q = 0.99$，可得预测曲线如图 5 所示



(a) 湖北提前 3 日执行　　(b) 美国执行全美实施居家令

图 **5**　湖北省和美国两地及时管控措施的感染者预测曲线

若全美实施居家令和湖北提前 3 日执行，即使得日平均接触人数减半，即仅提前三天执行隔离措施将大幅度降低疫情损失，而推后 5 日执行隔离措施将使得疫情损失大幅度增长。取隔离比例 $q = 0.1$ 可得美国测曲线如图 5 所示。提前 3 日执行严格隔离措施将使得疫情高峰于 **30 天后**到来。最高感染人数将分别达到 **2 万人/60 万人**，是原始数据的 0.7 倍。死亡人数将达到 **8 百/7 万人**，是原始数据的 0.6 倍。即若及时管控严格将使得疫情损失大幅度降低。

4.5 灵敏度分析

以湖北为例，分别使得患者的日平均接触人数、接触感染概率、初始潜伏者数量以及潜伏者相对于安全措施的比值上下波动 15%，观察最高感染人数与死亡人数值是否发生改变。

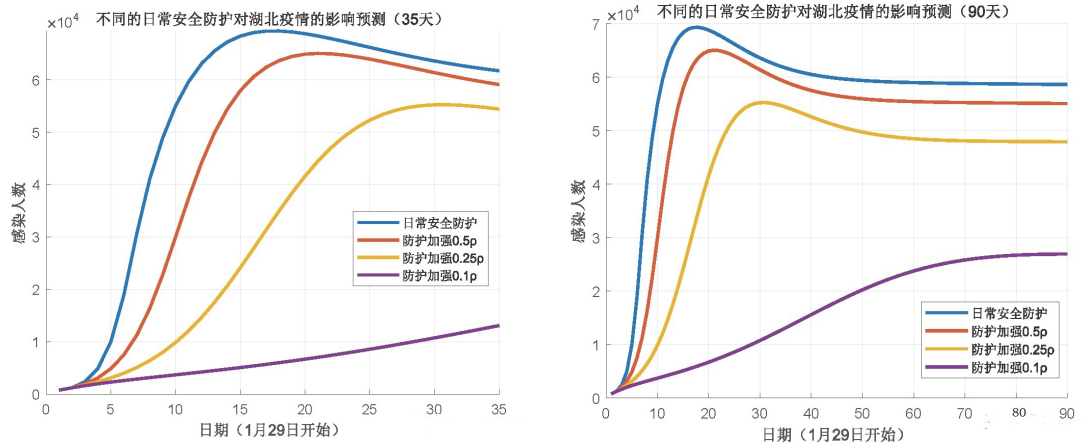


图 **6**　不同的日常安全防护对湖北的疫情影响预测

分析图 4 可知，随着人员之间日常接触率增大，个人日常安全防护将显得尤其重要。上图分析了有效接触系数降低对疫情发展的影响（假设诊断标准未发生变化），设置人员之间接触率 $c = 10.15$。有效接触系数分别取 0.5、0.25 和 0.1 时。个人日常防护措施将在保障个人安全的同时，对遏制疫情的发展起到重要作用。严格的日常安全防护有助于感染人数峰值时间的提前，并且有助于峰值人数的降低。在严格个人防护措施下 ($\rho = 0.1$)，感染人数峰值可下降接近 50%。

本文还模拟了追踪隔离措施，即追踪隔离比例下降对疫情发展的后果。如下图所示，当隔离比例下降为 0.9、0.8 和 0.6 倍时，感染人数的上升速率和峰值均会增加（假设诊断标准未发生变化，即感染人数未突跳）。尤其当取 $0.6q$ 时，感染人数峰值提高约两倍。由此可见，严格的医学追踪隔离是防止疫情发展的有效手段。

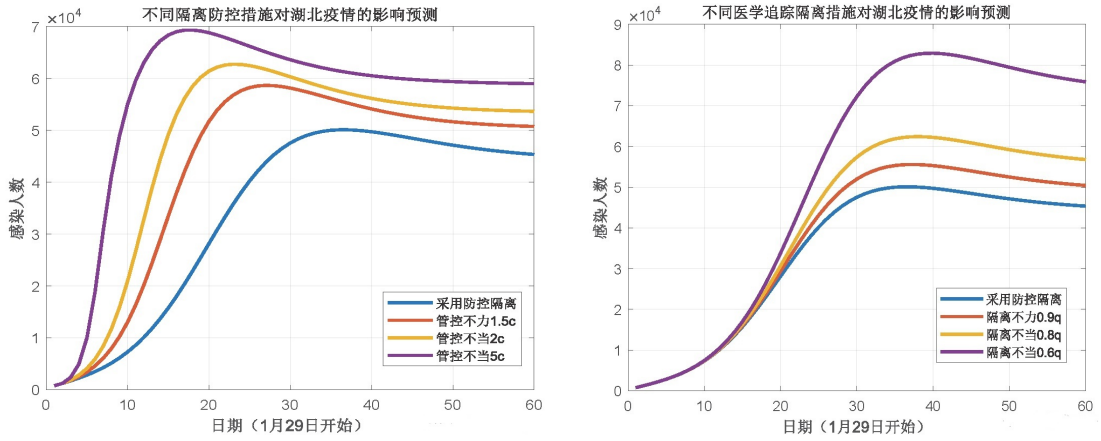


图 **7**　追踪隔离措施对湖北的疫情影响预测

5 问题二模型的建立与求解

5.1 问题描述与分析

问题二要求

其思维流程图如图 8 所示：

武漢理工大學

图 **8**　问题二思维流程图

5.2 模型的建立

5.3 模型的求解

5.4 实验结果及分析

结果如下表**??**所示：

表 **3**　**xxxxxxxxxxxxxxxxxxxxx**

xxxxxxx	xxxxxxx
xxxxxxx	909.80
xxxxxxx	852.60

由表3可知

其各个小车的运输细节图下图所示：

图 **9** **xxxxxxxxxxxxxxxxxxxxxxxxxx**

6 问题三模型的建立与求解

6.1 结果分析

7 灵敏度分析

8 模型的评价

8.1 模型的优点

(1)

(2)

8.2 模型的缺点

8.3 模型改进

参考文献

[1] Berger D W, Herkenhoff K F, Mongey S. An seir infectious disease model with testing and conditional quarantine[R]. National Bureau of Economic Research, 2020.

[2] Pandey G, Chaudhary P, Gupta R, et al. SEIR and Regression Model based COVID-19 outbreak predictions in India[J]. arXiv preprint arXiv:2004.00958, 2020.

[3] Hou C, Chen J, Zhou Y, et al. The effectiveness of quarantine of Wuhan city against the Corona Virus Disease 2019 (COVID-19): A well-mixed SEIR model analysis[J]. Journal of medical virology, 2020.

附录 A 问题一代码

动力学模型–python 源代码

```
from scipy.optimize import dual_annealing, minimize
from sklearn.metrics import r2_score
from collections import namedtuple
from matplotlib import pyplot as plt

SEIR_PARAM = namedtuple('SEIRparm', ['alpha', 'beta', 'theta'])
q = 0.000 # 隔离比例
lamada = 1/14
sigma = 1/7
deltaI=0.13
deltaq=0.13
d = 2.7*(10**(-4)) #病死率
gammaI=0.007 #感染者的恢复率
gammaH=0.014

class SEIR(object):
def __init__(self, P=None):
self.P = P

def _forward(self, S, E, I, Sq, Eq, H, D, C, param, max_iter):
alpha, beta, theta = param
est = dp.Table(columns=['S', 'E', 'I', 'Sq', 'Eq', 'H', 'D', 'C'])
for t in range(max_iter):
rou = S/(S+E+I+C)
S_ = S - alpha*rou*(beta+q*(1-beta))*(I+theta*E) + lamada*Sq
# S_ = S - alpha*theta*E - alpha*rou*(beta+q*(1-beta))*I + lamada*Sq
E_ = E + rou*alpha*beta*(1-q)*(I+theta*E) - sigma*E
I_ = I + sigma*E - (deltaI+d+gammaI)*I
Sq_ = Sq + rou*alpha*q*(1-beta)*(I+theta*E) - lamada*Sq
Eq_ = Eq + rou*alpha*beta*q*(I+theta*E) - deltaq*Eq
H_ = H + deltaI*I + deltaq*Eq - (d+gammaH)*H
C_ = C + gammaI*I + gammaH*H
D_ = D + d*(H+I)
S, E, I, Sq, Eq, H, D, C = S_, E_, I_, Sq_, Eq_, H_, D_, C_
est.append_row([S, E, I, Sq, Eq, H, D, C])
return est

def _loss(self, obs, est):
assert len(obs) == len(est)
loss = ((np.log2(obs + 1) - np.log2(est + 1)) ** 2).sum()
# loss = (np.abs(1-est['I']/obs['I'])+np.abs(1-est['D']/obs['D'])
#              +np.abs(1-est['C']/obs['C'])).sum()
```

```
        # print(est)
        self.lossing.append(loss)
        return loss

    def _optimize(self, param, s, e, i, sq, se, h, d, c, obs):
        est = self._forward(s, e, i, sq, se, h, d, c, param, len(obs))
        return self._loss(obs, est['I', 'D', 'C'].toarray())

    def fit(self, initS, initE, initI, initSq, initEq, initH, initD, initC, Y):
        self.lossing = []
        args = (initS, initE, initI, initSq, initEq, initH, initD, initC, Y['确诊', '死亡',
            '治愈'].toarray())
        param = [(0, 1),] * 3
        # result = minimize(self._optimize, [random()] * 5, args=args, bounds=param)['x']
        result = dual_annealing(self._optimize, param, args=args, seed=30, maxiter=50)['x']
        self.P = SEIR_PARAM(*result)

    def score(self, initS, initE, initI, initSq, initEq, initH, initD, initC, Y,
            plot=False):
        est = self._forward(initS, initE, initI, initSq, initEq, initH, initD, initC, self.P,
            len(Y))['I', 'D', 'C']
        loss = self._loss(Y['确诊', '死亡', '治愈'].toarray(), est.toarray())
        est.columns = ['确诊', '死亡', '治愈']
        r1 = r2_score(Y['治愈'], est['治愈'])
        r2 = r2_score(Y['死亡'], est['死亡'])
        r3 = r2_score(Y['确诊'], est['确诊'])
        if plot:
            self.plot_predict(Y, est)
        print(' - 平均潜伏期为: %.2f天' % (1.0 / self.P.beta))
        # print(' - 病毒再生基数: %.2f' % (self.P.alpha1 / self.P.beta + (self.P.alpha2 /
            self.P.sigma + self.P.alpha2 / self.P.gamma)/ 2))
        print(' - 确诊R2: %.4f' % r3)
        print(' - 死亡R2: %.4f' % r2)
        print(' - 治愈R2: %.4f' % r1)
        print(' - 模型R2: %.4f' % ((r1 + r2 + r3) / 3))
        print(' - 模型总误差: %.4f' % loss)
        return loss, (r1 + r2 + r3) / 3

    def plot_error(self):
        # plt.rcParams['font.sans-serif'] = ['SimHei']
        plt.plot(self.lossing, label=u'正确率')
        plt.legend()
        plt.show()

    def plot_predict(self, obs, est):
        for label, color in zip(obs.keys(), ['red', 'black', 'green']):
        # plt.rcParams['font.sans-serif'] = ['SimHei']
```

```
        plt.plot(obs[label], color=color)
        print(label)
        plt.plot(est[label], color=color, alpha=0.7)
        plt.legend()
        plt.show()

        def predict(self, initS, initE, initI, initSq, initEq, initH, initD, initC, T):

        return self._forward(initS, initE, initI, initSq, initEq, initH, initD, initC, self.P,
            T)


        train = data['患病','死亡','治愈']
        train.columns = ['确诊', '死亡', '治愈']
        train.accumulate(inplace=True)

        S = 900000000
        I = 1
        Sq = 200000
        Eq = 420000
        H = 20
        D = 0
        C = 0

        def searchBestParam(seir, S, I, Sq, Eq, H, D, C):
        min_loss, max_r2, best_param, likeli_potential = float('inf'),0.0, None, 0
        for potential in range(0, 500, 25):
        seir.fit(S, potential, I, Sq, Eq, H, D, C, train)
        loss, r2 = seir.score(S, potential, I, Sq, Eq, H, D, C, Y=train)
        print(loss,r2)
        if loss < min_loss: # and r2 > max_r2:
        print('潜在患者: %.4f | R2: %.4f | 误差:  %.6f' % (potential, r2, loss))
        min_loss, max_r2, best_param, likeli_potential = loss, r2, seir.P, potential
        seir.P = best_param
        print(seir.P, likeli_potential)
        seir.score(S, likeli_potential, I, Sq, Eq, H, D, C, Y=train, plot=True)
        return seir, likeli_potential

        seir, potentials = searchBestParam(SEIR(), S, I, Sq, Eq, H, D, C)


        def forcast(seir, T):
        predict = seir.predict(S, potentials, I, Sq, Eq, H, D, C, T)
        plt.plot(train['确诊'], label='确诊(真实)', color='red')
        plt.plot(train['死亡'], label='死亡(真实)', color='black')
        plt.plot(train['治愈'], label='治愈(真实)', color='green')
        # plt.plot(predict['S'], label='易感(预计)', color='blue', alpha=0.5)
```

```
        plt.plot(predict['E'], label='潜伏(预计)', color='orange', alpha=0.5)
        plt.plot(predict['I'], label='确诊(预计)', color='red', alpha=0.5)
        plt.plot(predict['D'], label='死亡(预计)', color='black', alpha=0.5)
        plt.plot(predict['C'], label='治愈(预计)', color='green', alpha=0.5)
        plt.legend()
        plt.show()
        forcast(seir, 30)
```

免疫遗传求解–python 源代码

```
# -*- coding: cp936 -*-
import numpy as np
import matplotlib.pyplot as plt
import math
import random

# Rastrigr 函数
def object_function(x):
f = 0
for i in range(0,len(x)):
f = f + (x[i] ** 2 - (10 * math.cos(2 * np.pi * x[i])) + 10)
return f
# 参数
def initpara():
NP = 100 # 种群数量
F = 0.6 # 缩放因子
CR = 0.7 # 交叉概率
generation = 2000 # 遗传代数
len_x = 10
value_up_range = 5.12
value_down_range = -5.12
return NP, F, CR, generation, len_x, value_up_range, value_down_range
# 种群初始化
def initialtion(NP):
np_list = [] # 种群，染色体
for i in range(0,NP):
x_list = [] # 个体，基因
for j in range(0,len_x):
x_list.append(value_down_range + random.random() * (value_up_range - value_down_range))
np_list.append(x_list)
return np_list
# 列表相减
def substract(a_list,b_list):
a = len(a_list)
new_list = []
```

```
for i in range(0,a):
new_list.append(a_list[i]-b_list[i])
return new_list
# 列表相加
def add(a_list,b_list):
a = len(a_list)
new_list = []
for i in range(0,a):
new_list.append(a_list[i]+b_list[i])
return new_list
# 列表的数乘
def multiply(a,b_list):
b = len(b_list)
new_list = []
for i in range(0,b):
new_list.append(a * b_list[i])
return new_list
# 变异
def mutation(np_list):
v_list = []
for i in range(0,NP):
r1 = random.randint(0,NP-1)
while r1 == i:
r1 = random.randint(0,NP-1)
r2 = random.randint(0,NP-1)
while r2 == r1 | r2 == i:
r2 = random.randint(0,NP-1)
r3 = random.randint(0,NP-1)
while r3 == r2 | r3 == r1 | r3 == i:
r3 = random.randint(0,NP-1)

v_list.append(add(np_list[r1], multiply(F, substract(np_list[r2],np_list[r3]))))
return v_list
# 交叉
def crossover(np_list,v_list):
u_list = []
for i in range(0,NP):
vv_list = []
for j in range(0,len_x):
if (random.random() <= CR) | (j == random.randint(0,len_x - 1)):
vv_list.append(v_list[i][j])
else:
vv_list.append(np_list[i][j])
u_list.append(vv_list)
return u_list
# 选择
def selection(u_list,np_list):
```

```
for i in range(0,NP):
if object_function(u_list[i]) <= object_function(np_list[i]):
np_list[i] = u_list[i]
else:
np_list[i] = np_list[i]
return np_list
# 主函数
NP, F, CR, generation, len_x, value_up_range, value_down_range = initpara()
np_list = initialtion(NP)
min_x = []
min_f = []
xx = []
for i in range(0,NP):
xx.append(object_function(np_list[i]))
min_f.append(min(xx))
min_x.append(np_list[xx.index(min(xx))])
for i in range(0,generation):
v_list = mutation(np_list)
u_list = crossover(np_list,v_list)
np_list = selection(u_list,np_list)
for i in range(0,NP):
xx = []
xx.append(object_function(np_list[i]))
min_f.append(min(xx))
min_x.append(np_list[xx.index(min(xx))])
# 输出
min_ff = min(min_f)
min_xx = min_x[min_f.index(min_ff)]
print('the minimum point is x ')
print(min_xx)
print('the minimum value is y ')
print(min_ff)
# 画图
x_label = np.arange(0,generation+1,1)
plt.plot(x_label, min_f)
plt.xlabel('iteration')
plt.ylabel('fx')
plt.savefig('./iteration-f.png')
plt.show()
```