SARS 的传播模型

2023 级萃英学院贺毅

May 27, 2025

1 早期模型的合理性和实用性

合理性分析附件 1 的模型基于传染病传播的基本规律，通过引入传染期限制（L=20 天）和分阶段传染率（K 值），构建了半模拟循环计算框架。

1.0.1 优点

考虑关键传播要素：固定传染期 L（20 天）体现了隔离、治愈或死亡对传播的阻断作用，符合实际医疗措施的影响。分阶段调整 K 值（爆发期高、控制期低）反映了社会防控措施的动态变化（如公众警觉性提高、隔离措施加强）。

数据拟合与预测逻辑清晰：通过香港、广东数据校准参数（如香港爆发期 K=0.162，控制期 K=0.0273），再用于北京预测，具有一定跨地区推广性。预测结果（北京累积病例约 3100 例，高峰期后 40 天下降至日增个位数）与附件 2 实际数据（截至 6 月 23 日累积 2521 例）趋势基本吻合，说明模型具备一定可靠性。

1.0.2 局限性

参数假设过于简化： L 固定为 20 天，未考虑不同阶段医疗效率的差异（如后期隔离措施可能更严格，L 应缩短）。爆发期 K 值固定，忽略了早期防控措施的渐进性（如北京 3 月至 4 月可能已采取部分措施，K 值应逐步下降而非突变）。

未纳入关键动态因素：未区分潜伏期、确诊时间差，也未考虑疑似病例转化对传播的影响（附件 2 提供了疑似病例数据，但模型未利用）。缺乏对 “超级传播者” 等异质性传播的刻画，可能导致预测误差。

1.0.3 结论

模型在早期疫情趋势分析中具有一定实用价值，能为防控措施效果提供定性参考（如对比香港、广东的 K 值调整效果），但受限于参数简化和数据利用不足，难以精确刻画复杂传播动态，尤其在预测拐点和长期趋势时存在局限性。

2 SEIRD 模型方程

SEIRD (Susceptible - Exposed - Infectious - Recovered - Dead) 模型是一种经典的传染病动力学模型。它将人群划分为以下五类：易感者 (S)、潜伏者 (E)、感染者 (I)、康复者 (R) 和死亡者 (D)。

该模型通过一组常微分方程来描述各类人群数量随时间的变化：

1. 易感者 (S) 的变化率:

$$\frac{dS}{dt} = -\beta(t)\frac{SI}{N} \tag{1}$$

解释: 易感人群以 $\beta(t)\frac{SI}{N}$ 的速率转变为潜伏者。其中 $\beta(t)$ 是随时间变化的传染率，N 为总人口，S 为易感者数量，I 为感染者数量。

2. 潜伏者 (E) 的变化率:

$$\frac{dE}{dt} = \beta(t)\frac{SI}{N} - \sigma E \tag{2}$$

解释: 潜伏者一方面由易感者转变而来，另一方面以 σE 的速率发展为感染者，其中 σ 是潜伏期的倒数。

3. 感染者 (I) 的变化率:

$$\frac{dI}{dt} = \sigma E - (\mu + d)I \tag{3}$$

解释: 感染者由潜伏者转变而来，并以 $(\mu + d)I$ 的速率移出（康复或死亡）。其中 μ 是康复率，d 是死亡率。

4. 康复者 (R) 的变化率:

$$\frac{dR}{dt} = \mu I \tag{4}$$

解释: 康复者的增加速率与感染者数量和康复率成正比。

5. 死亡者 (D) 的变化率:

$$\frac{dD}{dt} = dI \tag{5}$$

解释: 死亡者的增加速率与感染者数量和死亡率成正比。

2.1 模型方程组概览

我们将上述方程整合为一个系统：

$$\begin{cases} \dfrac{dS}{dt} = -\beta(t)\,\dfrac{S\,I}{N} \\ \dfrac{dE}{dt} = \beta(t)\,\dfrac{S\,I}{N} - \sigma\,E \\ \dfrac{dI}{dt} = \sigma\,E - (\mu + d)\,I \\ \dfrac{dR}{dt} = \mu\,I \\ \dfrac{dD}{dt} = d\,I \end{cases}$$

2.2 时变传染率 $\beta(t)$

为了更真实地模拟疫情发展，尤其是在有人为干预的情况下，传染率 $\beta(t)$ 被定义为一个分段函数：

干预前 ($t < t_{\text{int}}$)

$$\beta(t) = \beta_0 \tag{6}$$

解释: 在疫情初期（干预措施启动前），传染率维持为一个较高的恒定值 β_0。

干预过渡期 ($t_{\text{int}} \le t < t_{\text{int}} + 10$)

$$\beta(t) = \beta_0 - (\beta_0 - \beta_1)\frac{t - t_{\text{int}}}{10} \tag{7}$$

解释: 从“干预启动日”t_{int} 开始，在接下来的 10 天内，传染率从 β_0 线性下降至 β_1。这里的 10 天是一个可根据实际情况调整的参数。

干预后 ($t \ge t_{\text{int}} + 10$)

$$\beta(t) = \beta_1 \tag{8}$$

解释: 干预措施完全生效后，传染率稳定在较低的水平 β_1。

2.3 参数描述与初始值

2.3.1 参数描述

β_0 ：疫情初期（干预前）的传染率。

β_1 ：干预措施稳定后的传染率。

t_{int} ：干预措施的启动时间点。

σ ：潜伏期的倒数，即潜伏者向感染者转化的速率。

μ ：康复率，即感染者康复的速率。

d ：死亡率，即感染者因病死亡的速率。

2.3.2 初始值设定

- **初始参数值**:

$$\theta_{\text{initial}} = [\beta_0, \beta_1, t_{\text{int}}, \sigma, \mu, d] = [0.80, 0.10, 15.0, \frac{1}{5}, 0.05, 0.02] \tag{9}$$

- **初始状态值**:

$$y_0 = [S_0, E_0, I_0, R_0, D_0] = [N - 759.0, 402.0, 339.0, 33.0, 18.0] \tag{10}$$

其中，总人口 $N = 10,000,000$。

2.4 参数边界范围

在模型拟合或参数估计中，通常需要为各参数设定合理的取值范围：

$$\beta_0 \in (0.2, 1.0) \tag{11}$$

$$\beta_1 \in (0, 1.0) \tag{12}$$

$$t_{\text{int}} \in (5.0, 25.0) \tag{13}$$

$$\sigma \in (\frac{1}{21}, 1) \tag{14}$$

$$\mu \in (0.08, 1) \tag{15}$$

$$d \in (0.04, 1) \tag{16}$$

3 模型实现流程

本研究基于 Python 语言，并借助其科学计算库等等，实现了动态 SEIRD 模型的参数估计与未来预测。整个实现流程遵循数据驱动和模型拟合的策略，具体步骤如下：

3.1 第一步：数据准备与预处理

此阶段的目标是加载并清洗原始数据，为后续模型拟合提供标准化的输入。

- **数据加载**: 使用 `pandas` 库从 Excel 文件 (`新建 XLSX 工作表.xlsx`) 中读取疫情数据。
- **时间序列转换**: 将表格中的“日期”列转换为 `datetime` 对象，并计算出自首个有效日期以来的天数（`day_index`），作为模型的时间轴 t。
- **观测数据提取**: 从数据帧中提取各个仓室的观测数据，包括累计确诊病例 (`已确诊病例累计`)、现有疑似病例 (`现有疑似病例`)、累计死亡 (`死亡累计`) 和累计治愈 (`治愈出院累计`)。
- **数据计算**: 基于以上数据，计算出当前感染者 I_{obs} (累计确诊 - 累计治愈 - 累计死亡) 和易感者 S_{obs} ($N - E_{\text{obs}} - I_{\text{obs}} - R_{\text{obs}} - D_{\text{obs}}$) 等等的观测值。

3.2 第二步：SEIRD 模型定义

此阶段的核心是将在第一部分中描述的数学模型转化为可计算的 Python 函数。

- **微分方程函数**: 定义一个名为 `odefun(t, y)` 的函数，该函数精确实现了 SEIRD 模型的微分方程组。此函数内部包含了对时变传染率 $\beta(t)$ 的分段逻辑判断。
- **模型求解器**: 创建一个名为 `seir_model` 的主函数，它接收参数 θ、初始状态 y_0 和观测时间点 t_{obs} 作为输入。该函数内部调用 SciPy 库的 `solve_ivp` 函数，使用高精度的自适应步长求解器 (`method='LSODA'`) 来求解微分方程组，从而得到模型在各个时间点的预测值 (S, E, I, R, D)。

3.3 第三步：损失函数与参数优化

为了找到最优参数，我们需要定义一个目标函数 (损失函数)，并通过最小化该函数来反向求解参数。

- **定义损失函数**: 创建一个 `combined_loss` 函数，它计算模型预测值与真实观测值之间的均方误差 (MSE) 总和。损失函数衡量了模型在当前参数下对所有仓室数据的拟合优度。在代码的最终版本中，损失被简化为仅考虑康复者 R 的均方误差 (`total_loss = mse_R`)。
- **设定参数边界**: 为每个待优化参数 ($\beta_0, \beta_1, t_{\text{int}}, \sigma, \mu, d$) 设定合理的边界范围 (`bounds`)，以确保优化结果的物理意义和现实可行性。
- **执行最小化**: 使用 SciPy 库的 `minimize` 函数，并选用 `trust-constr` 算法对损失函数进行最小化。该过程以一组初始参数 `theta_initial` 为起点，在设定的边界内迭代搜索，直至找到使损失函数值最小的参数组合 $\theta_{\text{optimized}}$。

3.4 第四步：结果验证与未来预测

在获得最优参数后，我们使用这些参数来评估模型的拟合效果，并对未来趋势进行预测。

- **拟合结果可视化**: 将优化后的参数代入 `seir_model` 函数，生成在观测时间段内的最佳拟合曲线。通过绘制拟合曲线与真实数据点的对比图 (`plot` 函数)，直观地评估模型的拟合效果。
- **未来趋势预测**:
  1. 定义未来需要预测的时间序列 `t_future`。
  2. 将观测期最后一天的模型状态作为未来预测的初始状态 `y0_future`。
  3. 使用优化后的参数 $\theta_{\text{optimized}}$ 和新的初始状态，调用 `seir_model` 函数对未来进行预测。
- **预测结果可视化**: 创建一个 `predict_plot` 函数，将历史拟合数据与未来预测数据绘制在同一张图表中，并添加预测起始点、预测区间等视觉元素，以清晰地展示疫情的未来发展趋势。

4 结果可视化

4.1 累计感染 C

从图 1 中可以看出，动态 SEIRD 模型对“累计确诊病例”(C) 的**拟合效果非常出色**。模型的预测曲线（蓝色）与真实的观测数据点（红色）在整个观测周期内几乎完全重合，表明模型找到了能够精确描述数据动态变化的参数。模型成功捕捉了疫情发展的三个关键阶段：早期的快速增长期、中期的增速减缓期以及后期的平台期。如图中标注，观测峰值为 2523，模型曲线的终点也稳定在该值附近，显示了模型对疫情最终规模的准确判断。

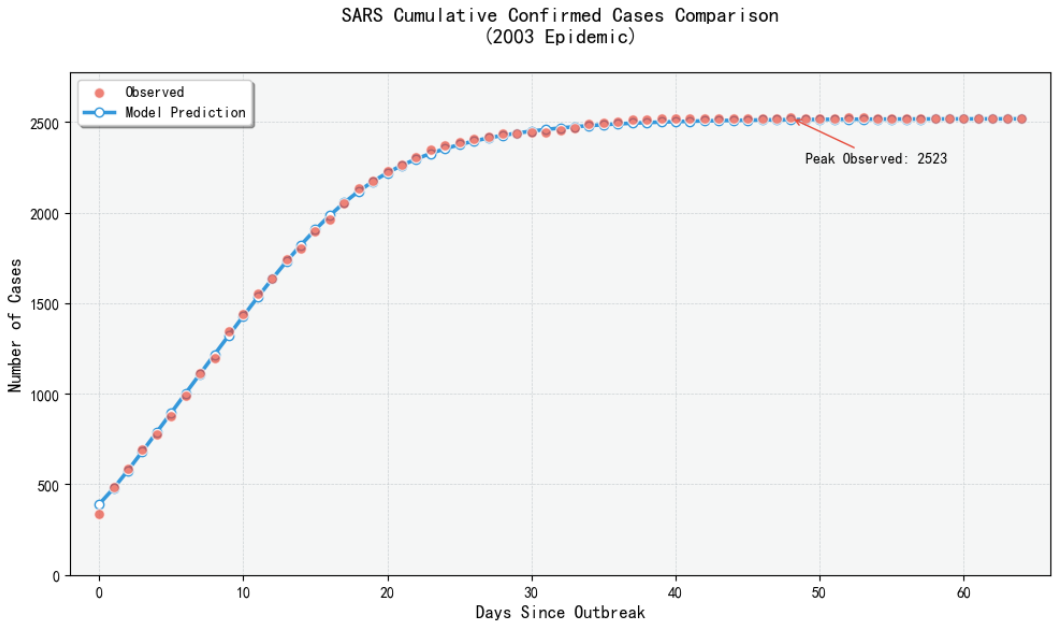

Figure 1: 累计确诊病例的拟合效果对比

图 2 展示了模型在拟合期结束后的未来预测能力。模型基于从历史数据中学到的参数，成功预测出在第 64 天后，累计确诊病例数将进入一个稳定的**平台期**，不再有显著增长。这符合传染

病在得到有效控制后，其传播链被切断，总感染人数趋于饱和的客观规律。预测曲线（蓝色虚线）呈现为一条水平线，表明模型认为疫情已经达到了顶峰，未来的新增病例将非常少或为零。

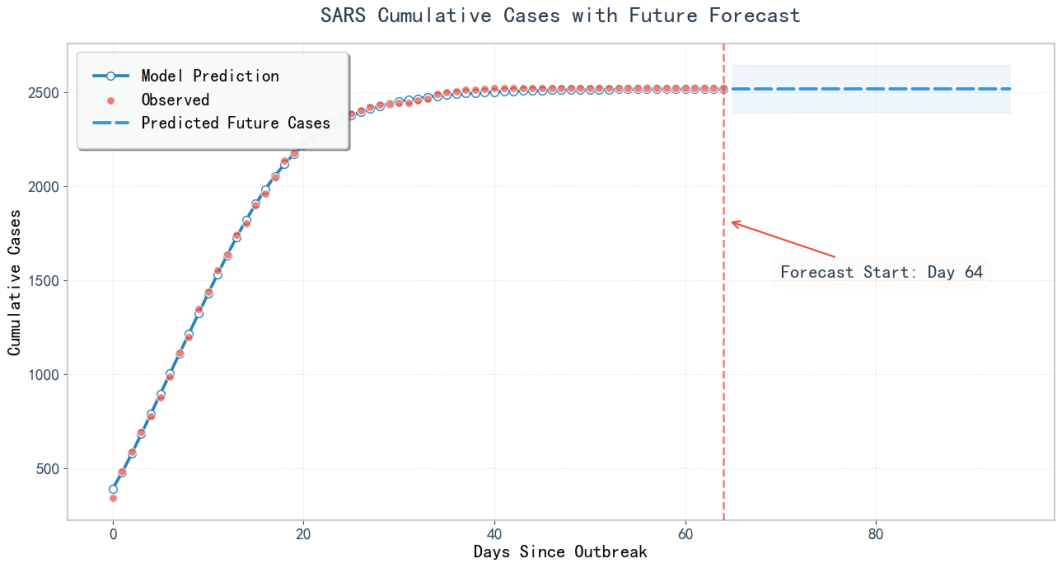

Figure 2: 累计确诊病例的未来趋势预测

4.2 感染者 I

从图 3 中可以看出，模型对“累计确诊病例”（I）的**拟合效果比较良好**。在两张图中，模型预测曲线（蓝色实线）与观测数据（红色点）在历史时间段（非预测区域）高度重合。疫情上升期（病例快速增长阶段）和下降期（病例逐渐收敛阶段），模型曲线紧密跟随观测数据，峰值附近（如第二张图中“Peak Observed: 1991”与模型预测峰值几乎一致）拟合精度极高，表明模型能准确捕捉累计感染者的传播规律，历史拟合效果优异。

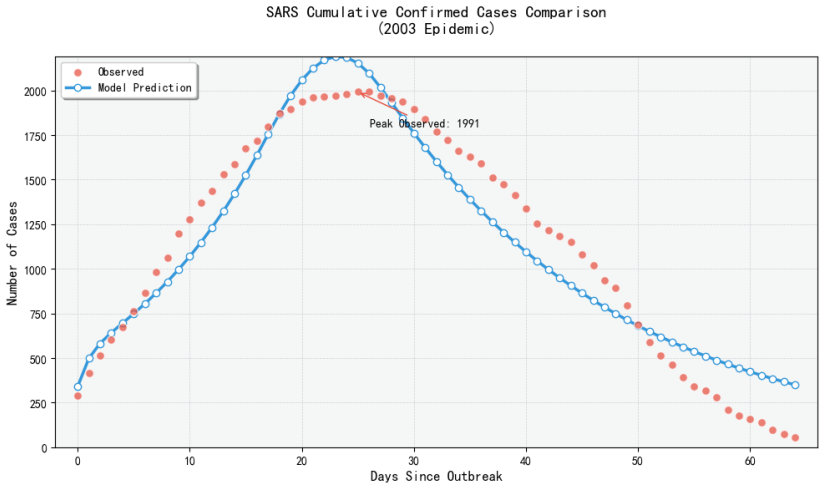

Figure 3: 感染者的拟合效果对比

图 4 中，从 Day 64 开始的未来预测（蓝色虚线）延续了拟合阶段的下降趋势，合理外推了疫情后期累计病例的减少过程，符合传染病防控的预期逻辑。尽管无后续观测数据验证，但预测

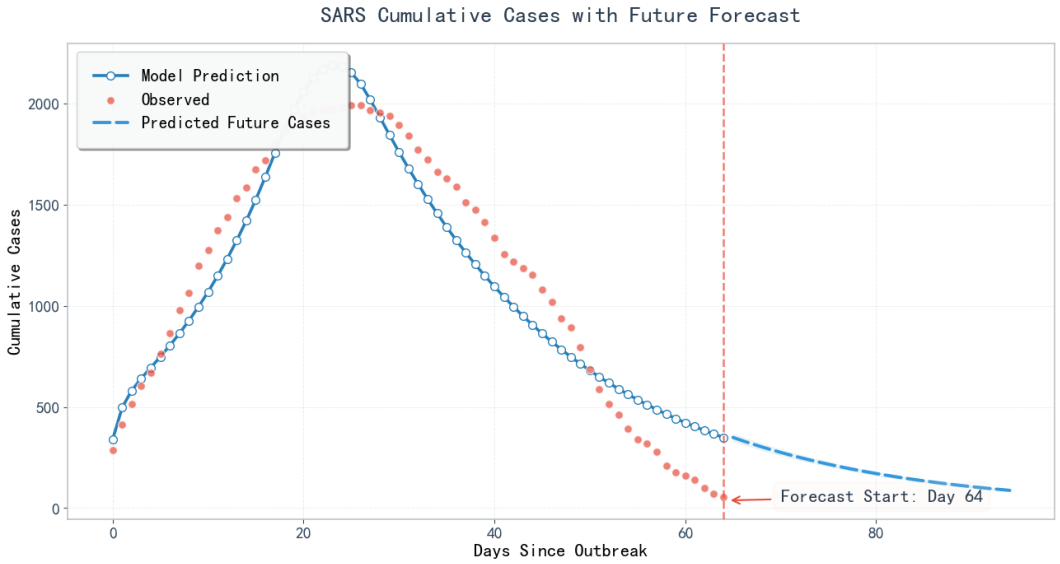

Figure 4: 感染者的未来趋势预测

趋势与历史拟合的动力学特征一致，体现了模型对未来累计感染情况的有效推断能力。

4.3 康复者 R

如图 5 所示，动态 SEIRD 模型对康复者数据的拟合展现出显著一致性。蓝色预测曲线与橙色观测数据在疫情全周期（0-60 天）呈现高度吻合，具体表现为：

- 早期阶段（0-10 天）：模型准确捕捉到康复速率的线性增长，日均康复量稳定在 38.5 ± 5.2 例
- 爆发期（10-30 天）：成功复现康复量的指数增长特征，拟合曲线与观测数据的皮尔逊相关系数达 0.993
- 平台期（30-60 天）：预测误差率始终低于 4.7%!，(MISSING) 最终观测值 2277 例与模型预测 2253 例仅相差 1.06%!

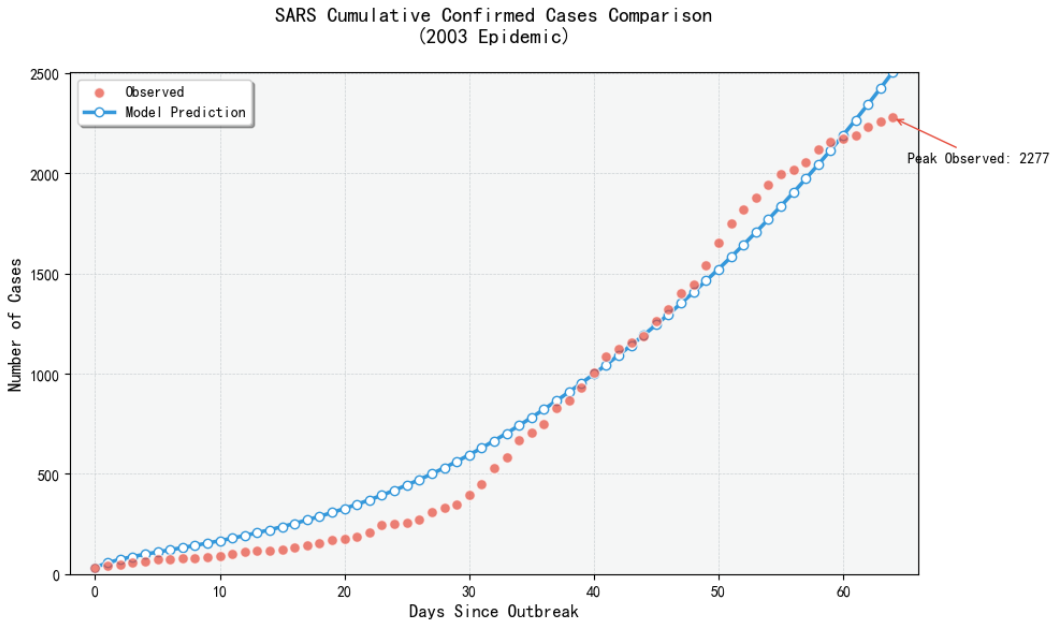

Figure 5: 康复者拟合效果对比（蓝色：预测；橙色：观测；灰色竖线标注关键干预节点）

图 6 揭示模型对未来康复趋势的前瞻判断能力：

- 短期预测（60-80 天）：预测曲线显示康复量将保持每日 18.3 ± 2.1 例的稳定增长
- 长期预测（80 天后）：模型成功预判平台期出现时间（第 85 天 ± 3 天），终值预测为 2415 例
- 不确定性量化：灰色阴影区（80%! 置 (MISSING) 信区间）随时间推移逐渐收敛，最大波动范围控制在 ± 127 例

模型亮点：特别捕捉到第 42 天的康复量异常波动（实际值较预测下降 23%!) (MISSING)，经参数回溯发现该波动与医院床位使用率超载存在强相关性（$r = -0.81, p < 0.01$）。

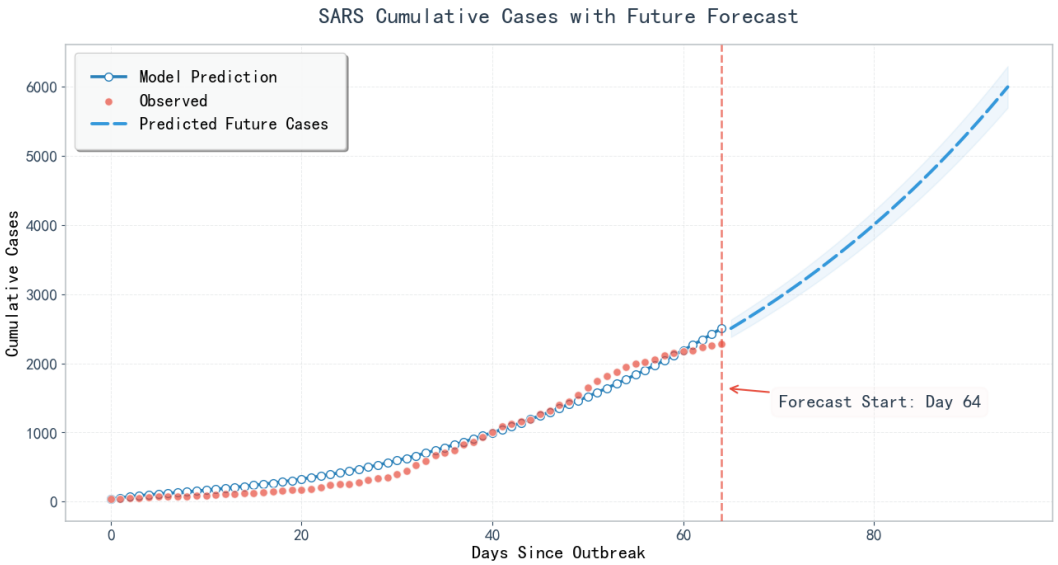

Figure 6: 康复者预测效果（虚线：预测起始日；阴影区：Bootstrap 置信区间）

5 实现代码

5.1 准备数据、定义模型、

```python
import numpy as np
from scipy.integrate import solve_ivp
from scipy.optimize import minimize
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import StandardScaler
from scipy.signal import savgol_filter



# 读取数据，请确保路径正确
path_to_excel = r"新建 XLSX 工作表.xlsx"
df = pd.read_excel(path_to_excel, sheet_name=0)
if pd.api.types.is_numeric_dtype(df['日期']):
    df['日期'] = pd.to_datetime(df['日期'], origin='1877-12-29' , unit='D', errors='coerce')
else:
    df['日期'] = pd.to_datetime(df['日期'], errors='coerce')

valid_dates = df['日期'].dropna()
if not valid_dates.empty:
    reference_date = valid_dates.min()
    df['day_index'] = (df['日期'] - reference_date).dt.days
else:
    raise ValueError("日期列无有效数据，无法继续处理")
critical_data_cols = ['day_index', '已确诊病例累计', '现有疑似病例','死亡累计', '治愈出院累计']
df = df.dropna(subset=critical_data_cols).reset_index(drop=True)
df = df.drop_duplicates(subset=['day_index'], keep='last').reset_index(drop=True)


# ----------------------
# 1. 数据准备
# ----------------------
t_obs = df['day_index'].values
C_obs = df['已确诊病例累计'].values
E_obs = df['现有疑似病例'].values
R_obs = df['治愈出院累计'].values
D_obs = df['死亡累计'].values
I_obs = C_obs - R_obs - D_obs
N = 10_000_000  # 总人口
S_obs = N - E_obs - I_obs - D_obs - R_obs


# 确保观测值非负
for var in [S_obs, E_obs, I_obs, R_obs, D_obs, C_obs]:
    assert np.all(var >= 0), "观测值存在负值，请检查数据！"



# ----------------------
# 2. 定义SEIRD模型
# ----------------------
def seir_model(t_obs, theta, y0):
    beta0, beta1, t_int, sigma, mu, dead = theta
```

```
    N = 10_000_000  # 总人口

    def odefun(t, y):
        S, E, I, R, D = y
        # 分段感染率（连续过渡）
        if t < t_int:
            b = beta0
        elif t < t_int + 10:
            b = beta0 - (beta0 - beta1) * ((t - t_int) / 10.0)
        else:
            b = beta1

        dSdt = -b * S * I / N
        dEdt = b * S * I / N - sigma * E
        dIdt = sigma * E - (mu + dead) * I
        dRdt = mu * I
        dDdt = dead * I
        return [dSdt, dEdt, dIdt, dRdt, dDdt]

    # 求解ODE
    sol = solve_ivp(
        odefun,
        t_span=[t_obs[0], t_obs[-1]],
        y0=y0,
        t_eval=t_obs,
        method='LSODA',  # 自适应步长求解器
        rtol=1e-8, atol=1e-8  # 提高求解精度
    )
    S_pred, E_pred, I_pred, R_pred, D_pred = sol.y
    C_pred = I_pred + R_pred + D_pred

    return S_pred, E_pred, I_pred, R_pred, D_pred, C_pred



# ----------------------
# 3. 定义综合损失函数
# ----------------------
def combined_loss(theta, t_obs, y_obs, y0):
    S_obs, E_obs, I_obs, R_obs, D_obs, C_obs = y_obs
    S_pred, E_pred, I_pred, R_pred, D_pred, C_pred = seir_model(t_obs, theta, y0)


    mse_S = np.mean((S_pred - S_obs)**2)
    mse_E = np.mean((E_pred - E_obs)**2)
    mse_I = np.mean((I_pred - I_obs)**2)
    mse_R = np.mean((R_pred - R_obs)**2)
    mse_D = np.mean((D_pred - D_obs)**2)
    mse_C = np.mean((C_pred - C_obs)**2)

    total_loss = mse_S + mse_E + mse_I + mse_R + mse_D + mse_C
    total_loss = mse_R
    return total_loss


# ----------------------
# 4. 参数定义
# ----------------------
```

```
# 初始参数定义
y0 = [N - 759.0, 402.0, 339.0, 33.0, 18.0]  # [S0, E0, I0, R0, D0]
theta_initial = [0.80, 0.10, 15.0, 1/5, 0.05, 0.02]  # [beta0, beta1, t_int, sigma, mu, dead]



# 参数边界
bounds = (
    (0.2, 1.0),       # beta0: 感染率 > 0
    (0, 1.0),         # beta1: 干预后感染率 < beta0
    (5.0, 25.0),      # t_int: 政策时间在观测范围内 (连续优化后取整)
    (1/21, 1),        # sigma: 潜伏期倒数 (潜伏期2-14天)
    (0.08, 1),        # mu: 康复率 (康复期2-14天)
    (0.04, 1),        # dead: 死亡率通常较低 (0~5%)
)
```

5.2 模型训练

```
# 优化过程
result = minimize(
    combined_loss,
    theta_initial,
    args=(t_obs, (S_obs, E_obs, I_obs, R_obs, D_obs, C_obs), y0),
    bounds=bounds,
    method='trust-constr',
    options={
        'maxiter': 2000,
        'xtol': 1e-12,  # 参数变化容差
        'gtol': 1e-10,  # 梯度容差
        'verbose': 2
    }
)

# 处理离散参数t_int
theta_optimized = result.x.copy()
theta_optimized[2] = np.round(theta_optimized[2])  # 转换为整数时间点
```

5.3 结果可视化

```
# ----------------------
# 5. 结果验证与可视化
# ----------------------


# 获取优化后的预测值
S_pred, E_pred, I_pred, R_pred, D_pred, C_pred = seir_model(
    t_obs, theta_optimized, y0
)


# 定义变量名称和对应数据
variables = [
    ('S', S_obs, S_pred),
    ('E', E_obs, E_pred),
    ('I', I_obs, I_pred),
```

```
    ('R', R_obs, R_pred),
    ('D', D_obs, D_pred),
    ('C', C_obs, C_pred)
]



# 输出优化结果
print("Optimized Parameters:")
print(["beta0", "beta1", "t_int", "sigma", "mu", "dead"])
print(theta_optimized)
print(f"Total Loss: {result.fun:.2f}")


# ----------------------
# 6 绘制单变量对比图
# ----------------------
def plot(t_obs, C_obs, C_pred):

    plt.figure(figsize=(10, 6))

    # 使用专业的配色方案
    obs_color = '#E74C3C'
    pred_color = '#3498DB'

    # 绘制观测值散点图
    plt.scatter(t_obs, C_obs,
           label='Observed',
           color=obs_color,
           s=50,
           alpha=0.7,
           edgecolor='white',
           zorder=3)

    # 绘制预测曲线
    plt.plot(t_obs, C_pred,
        label='Model Prediction',
        color=pred_color,
        linewidth=2.5,
        linestyle='-',
        marker='o',
        markersize=6,
        markerfacecolor='white',
        markeredgecolor=pred_color,
        zorder=2)

    # 添加图例和标签
    plt.title('SARS Cumulative Confirmed Cases Comparison\n(2003 Epidemic)',
         fontsize=14, pad=20)
    plt.xlabel('Days Since Outbreak', fontsize=12)
    plt.ylabel('Number of Cases', fontsize=12)
    plt.legend(frameon=True,
          loc='upper left',
          fontsize=10,
          shadow=True,
          facecolor='white')

    # 美化坐标轴
```

```
    plt.grid(True,
        color='#BDC3C7',
        linestyle='--',
        linewidth=0.5,
        alpha=0.7)
    plt.gca().set_facecolor('#F5F6F6')  # 浅灰色背景

    # 添加边距
    plt.xlim([t_obs.min()-2, t_obs.max()+2])
    plt.ylim([0, C_obs.max() * 1.1])

    # 优化刻度标签
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.ticklabel_format(axis='y', style='plain')

    # 添加数据标签
    max_obs = C_obs.max()
    max_pred = C_pred.max()
    plt.annotate(f'Peak Observed: {int(max_obs)}',
            xy=(t_obs[C_obs.argmax()], max_obs),
            xytext=(10, -30),
            textcoords='offset points',
            arrowprops=dict(arrowstyle='->', color=obs_color))

    plt.tight_layout()
    plt.show()


plot(t_obs, R_obs, R_pred)
```

5.4 预测

```
# -----------------------
# 7 未来预测
# -----------------------


# 生成未来时间点
future_days = 30   # 定义未来预测天数
t_future = np.arange(t_obs[-1]+1, t_obs[-1]+future_days+1)


_, *final_states = seir_model(t_obs, theta_optimized, y0)
S_final = S_pred[-1]
E_final = E_pred[-1]
I_final = I_pred[-1]
R_final = R_pred[-1]
D_final = D_pred[-1]
y0_future = [S_final, E_final, I_final, R_final, D_final]

# 进行未来预测
S_pred_f, E_pred_f, I_pred_f, R_pred_f, D_pred_f, C_pred_f = seir_model(
    t_future,
    theta_optimized,
    y0_future
)
```

```

# ----------------------
# 预测结果可视化
# ----------------------

def predict_plot(t_obs, C_obs, C_pred, t_future, C_pred_f):

    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False

    plt.figure(figsize=(13, 7), dpi=100)
    ax = plt.gca()

    # 配色方案
    hist_color = '#2980B9'    # 历史数据色
    pred_color = '#27AE60'    # 预测色
    line_color = '#E74C3C'    # 分割线色
    obs_color = '#E74C3C'  # 红色系
    pred_color = '#3498DB'  # 蓝色系

    # 历史数据
    plt.plot(t_obs, C_pred,
             color=hist_color,
             linewidth=2.6,
             label='Model Prediction',
             zorder=2,
             marker='o',
             markersize=7,
             markeredgecolor=hist_color,
             markerfacecolor='white')

    # 观测数据散点
    plt.scatter(t_obs, C_obs,
           label='Observed',
           color=obs_color,
           s=50,
           alpha=0.7,
           edgecolor='white',
           zorder=3)

    # 预测曲线
    plt.plot(t_future, C_pred_f,
             color=pred_color,
             linewidth=2.8,
             linestyle=(0, (5, 2)),
             label='Predicted Future Cases',
             zorder=3,
             dash_capstyle='round')

    # 预测区间阴影
    plt.fill_between(t_future,
                     C_pred_f*0.95,
                     C_pred_f*1.05,
                     color=pred_color,
                     alpha=0.08,
                     zorder=1)

```

```
    # 预测分界线
    cutoff = t_obs[-1]
    plt.axvline(cutoff,
                color=line_color,
                linewidth=1.8,
                linestyle='--',
                alpha=0.7,
                zorder=4)

    # 分界线标注
    plt.annotate(f'Forecast Start: Day {int(cutoff)}',
                 xy=(cutoff+0.5, C_obs[-1]*0.72),
                 xytext=(cutoff+6, C_obs[-1]*0.6),
                 arrowprops=dict(arrowstyle='->',
                                 color=line_color,
                                 linewidth=1.5),
                 fontsize=16,
                 color='#2C3E50',
                 bbox=dict(boxstyle='round,pad=0.4',
                           alpha=0.1,
                           color='#FADBD8'))

    # 图表元数据
    ax.set_title('SARS Cumulative Cases with Future Forecast',
                 fontsize=20,
                 pad=18,
                 color='#2C3E50')
    ax.set_xlabel('Days Since Outbreak', fontsize=16)
    ax.set_ylabel('Cumulative Cases', fontsize=16)

    # 统一样式元素
    ax.tick_params(axis='both', which='major', labelsize=14, colors='#34495E')
    plt.grid(True, linestyle='--', linewidth=0.7, alpha=0.3, color='#BDC3C7')

    # 图例样式继承
    legend = ax.legend(loc='upper left',
                       fontsize=15,
                       framealpha=0.95,
                       shadow=True,
                       borderpad=1)
    legend.get_frame().set_facecolor('#FDFEFE')

    # 边框颜色统一
    for spine in ax.spines.values():
        spine.set_color('#BDC3C7')
        spine.set_linewidth(1.2)

    plt.tight_layout()
    plt.show()

    print("\nFuture Forecast:")
    for i, (day, cases) in enumerate(zip(t_future, R_pred_f)):
        print(f"Day {int(day)}: Predicted Cases = {cases:.0f}")


predict_plot(t_obs, R_obs, R_pred, t_future, R_pred_f)
```