

# 基于大量行车轨迹数据的信号灯周期估计模型

## 摘要

在实际交通场景应用中，红绿灯的周期时长是辅助预测动态交通路况、缓解用户焦虑情绪的重要因素。根据大量客户的行车轨迹数据估计交通信号灯的周期的方式可以有效弥补由于接入交管部门的红绿灯数据覆盖的道路范围有限、效果不够理想的问题，具有重要的应用价值。

**对于问题一**，首先进行数据预处理，将速度阈值设为  $1\text{m/s}$ ，路口范围划定为  $100\text{m}$  内，剔除异常值，并删除包含缺失值的数据行。随后，我们绘制行车轨迹图，发现各个路口的行车轨迹均为变道、左转、直行（包括掉头）的单一类型，也就是可以通过研究单一车辆行为来建立总车辆的模型。因此，我们先对单一车辆处于红灯或绿灯的状态进行判断，建立单一车辆的信号灯状态和周期的表达式。将所有单一车辆的周期取平均值，得到了问题一的信号灯周期估计模型。在求解时，我们首先对停车时间进行**分组聚类**，随后应用**离散傅里叶变换公式**，对时间间隔进行傅里叶变换。通过计算傅里叶变换结果的绝对值平方，得到功率谱密度结果，在功率谱密度中找到峰值对应的频率，即为红绿灯的主要周期。得到**信号灯周期的估计值**：从 A1 到 A5 红灯时长为 56、41、61、62、53 秒；绿灯时长为 40、13、45、37、23 秒。

**对于问题二**，我们对三个模型估计精度的影响因素样本车辆比例、车流量、定位误差进行逐一分析。假设车流量服从泊松分布，定位误差服从高斯分布，分别引入车流量参数  $\alpha$ 、定位误差参数  $\beta$  和样本车辆比例  $p$ ，对问题一中的估计模型进行改写。在求解过程中，通过**最小二乘法**来估计出信号灯的周期  $T$ ，并通过调整车流量参数  $\alpha$  和定位误差参数  $\beta$  来提高信号灯周期估计的精度。得到**信号灯周期的估计值**：从 B1 到 B5 红灯时长为 91.8、8.6、27、21、53 秒；绿灯时长为 51.6、18.4、21、31、24 秒。

**对于问题三**，我们在问题二建立的周期估计模型基础上，加入**滑动窗格法**解决信号灯的多个周期切换情境。定义一个固定大小为 200 秒的时间窗格，每次以 100 秒的速度移动，若移动前后有明显的周期变化则说明红绿灯周期发生了变化。

得到信号灯的新旧周期和切换时刻，记录在表 3 中。并得到**周期变化所需的条件**：路口存在明显的车流量变化，具体表现是通过滑动窗口模型求解的前后估计周期时长之差大于 60 秒。

**对于问题四**，在问题二建立的周期估计模型基础上，首先对单一车辆的航向角进行计算，并基于航向角将行车轨迹按东南西北四个方向进行分类。我们通过轨迹识别车子的行车轨迹类型，即变道、左转、直行（包括掉头）和右转。若为右转，由于其行车轨迹不受信号灯状态影响，则删除该车辆的所有轨迹数据。此时，问题四转化为与问题二的情境一致，由此建立问题四的信号灯周期估计模型，并根据题目提供的数据求解。得到该路口每个方向信号灯周期的估计值，记录在表 4 中。

**关键字：**K-means 聚类 傅里叶变换 最小二乘法 滑动窗口模型

## 一、问题重述

### 1.1 问题背景

在交通场景日益复杂的当下，红绿灯是交通管理部门管控车辆通行秩序及调整道路交通流量的重要基础设施，红绿灯的周期时长是动态交通服务提供商针对不同道路交通场景为用户提供相应交通服务所需要的重要信息之一。例如，为了缓解用户在交通拥堵场景下的焦虑情绪，可以将根据用户当前要等待的红绿灯的周期预测用户将要等待的时长，并将该时长推送给用户。此外，红绿灯的周期时长还可以用于精准计算通行路口代价，以作为辅助特征预测道路拥堵、拥堵消散等动态交通路况。

在已有技术中，红绿灯的周期时长通常依赖于相关部门的直接提供，但是由于接入的红绿灯数据覆盖的道路范围有限，效果并不理想。因此，使用大量客户的行车轨迹数据估计交通信号灯的周期是一种有效的技术手段，具有重要的应用价值。

### 1.2 问题要求

**问题 1** 要求在红绿灯周期固定不变的前提下，根据 5 个不相关路口各自一个方向连续 1 小时内车辆的行车轨迹数据，建立数学模型，求出这些路口相应方向的信号灯周期。

**问题 2** 在问题一的基础上，将样本车辆比例、车流量、轨迹数据定位误差等对模型估计精度可能产生影响的因素纳入考虑，要求建立优化后的模型，根据另外 5 个不相关路口各自一个方向连续 1 小时内样本车辆的轨迹数据，求出这些路口相应方向的信号灯周期。

**问题 3** 要求考虑信号灯周期的变化，根据另外 6 个不相关路口各自一个方向连续 2 小时内样本车辆的轨迹数据，建立数学模型，1. 判断这些路口相应方向的信号灯周期在监测时间段内是否变化;2. 求出周期切换的时刻;3. 指明识别出周期变化所需的时间和条件。

**问题 4** 在问题三的基础上，将路口的一个方向改为所有方向，要求识别出该路口信号灯的周期。

## 二、问题分析

### 2.1 问题一分析

对于问题一，要求求出给定 5 个路口相应方向的信号灯周期，我们只需要首先进行数据预处理，剔除异常值，并删除缺失值。绘制行车轨迹图容易发现，各个路口的行车

轨迹均为单一类型。因此我们只要对单一车辆处于红灯或绿灯的状态进行分析，建立单一车辆的信号灯状态和周期的表达式，将所有单一车辆的周期取平均值，就得到了问题一总体的信号灯周期估计模型。

得到信号灯周期的估计值，并按格式要求填入表 1。

## 2.2 问题二分析

对于问题二，要求在问题一的基础上，将三个影响因素纳入考虑后求出另外 5 个路口相应方向的信号灯周期。对此，我们只需假设车流量服从泊松分布，定位误差服从高斯分布，通过引入车流量参数  $\alpha$ 、定位误差参数  $\beta$  和样本车辆比例  $p$ ，对问题一中的估计模型进行调整。在求解过程中用最小二乘法不断寻找最优的参数解，即可得到信号灯周期的估计值，并按格式要求填入表 2。

## 2.3 问题三分析

对于问题三，要求信号灯周期是动态变化的，对此，我们只需在问题二估计模型的基础上加入滑动窗格法建模求解。定义一个固定大小为 200 秒的时间窗格，每次以 100 秒的速度移动，若移动前后有明显的周期变化则说明红绿灯周期发生了变化。我们在实际求解时不断对该周期变化进行优化，得到的结果即为周期变化所需的时间条件。

得到信号灯的新旧周期和切换时刻，并按格式要求填入表 3。

## 2.4 问题四分析

对于问题四，多个路口的单个方向变成了一个路口的所有方向，对此，我们只需先计算每辆车的航向角，再把车子轨迹按东南西北四个方向进行分类，通过轨迹识别车子的运动状态是左转、右转还是直行，如果是右转的则删除该车辆的所有轨迹数据，将其转化为与问题二一样的情境求解即可。

得到路口每个方向的信号灯周期，并按格式要求填入表 4。

## 三、 模型假设

为简化问题，本文做出以下假设：

- 假设 1 假设车流量服从泊松分布，定位误差服从高斯分布。
- 假设 2 假设信号灯只有红绿两种状态。且对于问题一和问题二，假设红绿灯周期固定不变。
- 假设 3 假设车辆的横向速度变化大于纵向速度变化的两倍时视为变道。

## 四、 符号说明

符号	说明	单位
$T_{signal}$	信号灯的红绿周期	s
$T_{track}$	车辆行车轨迹数据的采样间隔	s
$t_i$	第 $i$ 个车辆的行车轨迹数据的时间点	s
$x_i(t)$	第 $i$ 个车辆在时间点 $t$ 时的 X 坐标	m
$y_i(t)$	第 $i$ 个车辆在时间点 $t$ 时的 y 坐标	m
$z_i(t)$	第 $i$ 个车辆在时间点 $t$ 时的 z 坐标	m
$v_i(t)$	第 $i$ 个车辆在时间点 $t$ 时的速度	$m/s$
$a_i(t)$	第 $i$ 个车辆在时间点 $t$ 时的加速度	$m/s^2$
$T_{red}$	红灯阶段的时长	s
$T_{green}$	绿灯阶段的时长	s
$r_i(t)$	第 $i$ 个车辆在时间点 $t$ 是否处于红灯阶段的指示变量	/
$N$	总周期数	/
$T$	假设信号灯的周期	s
$v$	车辆行驶速度	$m/s$
$D$	一个路口的所有车辆轨迹数据	/
$n$	车辆数	/
$s_i$	第 $i$	kg
$f(t)$	$t$ 时刻信号灯的状态	/
$\{f(t_i)\}$	信号灯的状态变化序列	/
$d_i$	第 $i$ 辆车通过路口所走的距离	m

符号	说明	单位
$\bar{v}$	通过路口的车辆的平均速度	$m/s$
$t_i$	第 $i$ 辆车通过路口所花费的时间	$s$
$\alpha$	车流量对信号灯周期估计的影响参数	/
$\beta$	定位误差对信号灯周期估计的影响参数	/
$t_{ij}$	第 $i$ 个车道上第 $j$ 辆车通过路口的时间点	$kg$
$s(t)$	信号灯的状态变化函数	/
$A$	信号灯状态变化的幅度	/
$\omega$	信号灯状态变化的频率	$s^{-1}$
$\varphi$	信号灯状态变化的相位差	$rad$
$x(t)$	$t$ 时刻车辆在某个路口的位置	$m$
$p$	样本车辆的比例	/
$q$	样本车流量	辆/ $s$
$e$	定位误差	$m$
$x_i$	第 $i$ 辆车通过信号灯的时间	$kg$
$t$	车辆通过信号灯的时间间隔	$s$
$F$	通过路口的车辆数量	/
$S$	两辆车通过的平均时间间隔	$s$
$T_{est}$	信号灯周期的估计值	$s$
$T_{new}$	更新的信号灯周期	$s$

## 五、问题一的模型的建立和求解

### 5.1 问题一模型的建立

#### 5.1.1 车辆路口行车轨迹的确定

首先我们先尝试讨论单一车辆在信号指示灯路口可能存在的几种行车轨迹类型：

1. 右转，此时车辆行车轨迹不受信号灯改变的影响；
2. 左转，只有绿灯才能通行，红灯时车辆坐标位置不变；
3. 直行，只有绿灯才能通行，红灯时车辆坐标位置不变；
4. 掉头，此时对同一辆车来说，必有部分车辆行车轨迹重叠；
5. 变道，此时车辆行车轨迹会呈现两条明显的直线，y轴坐标值在变道时突变；

我们注意到，若当几种行车轨迹类型同时出现时，情况会变得复杂，需要先对轨迹进行判断归类，再分情况讨论。因此，先将原始数据绘制成轨迹图，观察路口的行车轨迹情况。

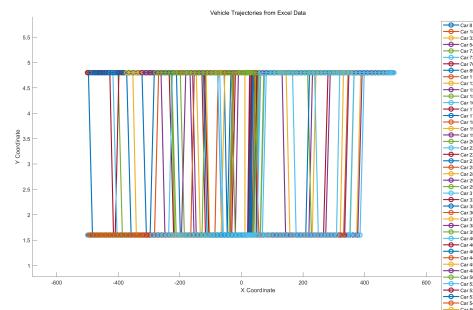


图 1 A1: 变道类型

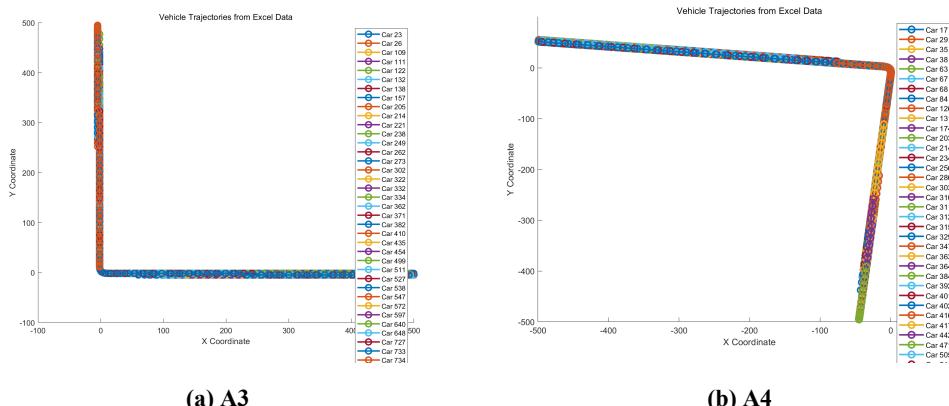


图 2 左转类型

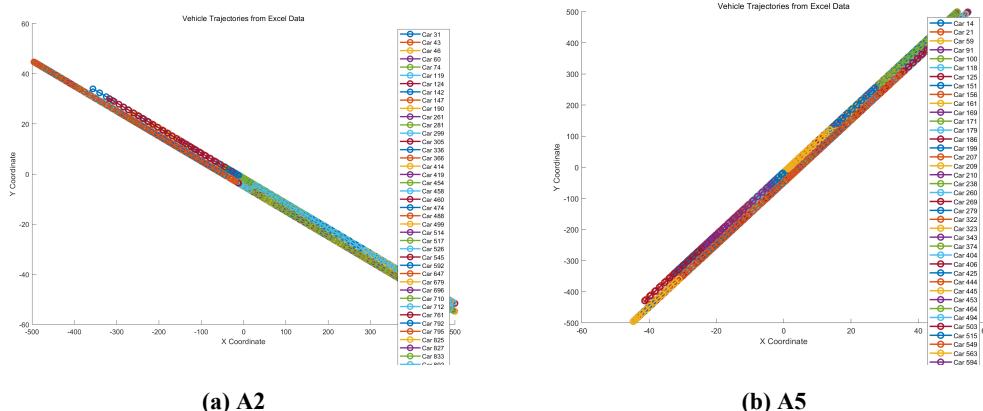


图3 直行和掉头类型

对轨迹图进行分析，发现给定的 A1-A5 路口方向的行车轨迹都属于单一类型，其中 A1 属于变道类型，A3 和 A4 属于左转类型，A2 和 A5 属于直行和掉头类型（将其视为一类）。

因此，在建立问题一的数学模型时，我们只需要判断路口对应类型，再按类型模式计算周期即可。

### 5.1.2 数据预处理

(1). 保留路口范围内的数据：

过滤数据，仅保留 x 和 y 坐标值在 [-100, 100] 范围内的数据，即仅保留路口附近的数据。

(2). 计算每辆车的速度:

使用 `groupby` 按车辆 id 分组，然后对 x 和 y 列使用 `diff` 方法计算差分，即连续两点之间的位置变化，再通过计算两点间水平（x 轴）和垂直（y 轴）位置变化的平方和的平方根，计算每辆车在每个时间点的速度，并将结果存储在 speed 列中。

### (3). 计算车辆的方向:

再次使用 `groupby` 按车辆 id 分组，并对 x 和 y 列使用 `diff` 方法计算差分。通过计算 y 轴差分与 x 轴差分的反正切值 (`np.arctan2`)，计算每辆车在每个时间点的方向，并将结果存储在 `direction` 列中。

#### (4). 删除缺失值:

删除包含缺失值的行。

### 5.1.3 信号灯周期的讨论和确定

首先，我们需要定义一些符号：

-  $T_{signal}$ : 信号灯的红绿周期, 单位为秒;

-  $T_{track}$ : 车辆行车轨迹数据的采样间隔, 单位为秒;

- $t_i$ : 第 i 个车辆的行车轨迹数据的时间点, 单位为秒;
- $x_i(t)$ : 第 i 个车辆在时间点 t 时的 X 坐标, 单位为米;
- $y_i(t)$ : 第 i 个车辆在时间点 t 时的 Y 坐标, 单位为米;
- $v_i(t)$ : 第 i 个车辆在时间点 t 时的速度, 单位为米/秒;
- $a_i(t)$ : 第 i 个车辆在时间点 t 时的加速度, 单位为米/秒<sup>2</sup>;

按照题意, 因为信号灯只有红和绿两种情况, 我们将每个信号灯周期分成红灯和绿灯两个阶段。设红灯阶段的时长为  $T_{red}$ , 绿灯阶段的时长为  $T_{green}$ , 易得:

$$T_{red} + T_{green} = T_{signal} \quad (1)$$

随后, 我们尝试从单一车辆的行车轨迹数据中识别红灯状态阶段和绿灯状态阶段。假设某车辆在某一信号灯周期的某时间点  $t_i$  处于红灯状态, 那么它在  $t_i + T_{red}$  时应该是绿灯状态; 同理, 如果它在  $t_i$  处于绿灯状态, 则在  $t_i + T_{green}$  时应该是红灯状态。

因此, 我们可以尝试列出公式 (2) 来判断单一车辆在某时间点处于红灯状态还是绿灯状态:

$$r_i(t) = \begin{cases} 1, & \text{if } t_i \bmod T_{signal} < T_{red} \\ 0, & \text{if } t_i \bmod T_{signal} \geq T_{red} \end{cases} \quad (2)$$

其中,  $r_i(t)$  为第 i 个车辆在时间点 t 是否处于红灯阶段的指示变量。如果  $r_i(t) = 1$ , 表示该车辆在时间点 t 处于红灯阶段; 如果  $r_i(t) = 0$ , 表示该车辆在时间点 t 处于绿灯阶段。

由此, 我们可以通过计算每个周期内红灯阶段的时长和绿灯阶段的时长来估计信号灯的红绿状态周期。假设在某个周期内, 有 n 辆车处于红灯阶段, m 辆车处于绿灯阶段, 那么该周期的红灯阶段时长和绿灯阶段时长分别为:

$$T_{red} = \frac{1}{n} \sum_{i=1}^n T_{track}(r_i(t) = 1) \quad (3)$$

$$T_{green} = \frac{1}{m} \sum_{i=1}^m T_{track}(r_i(t) = 0) \quad (4)$$

其中,  $T_{track}(r_i(t) = 1)$  表示在  $r_i(t) = 1$  时, 第 i 个车辆的行车轨迹数据的采样间隔。

最后, 取所有单一车辆周期的平均值来估计信号灯的红绿周期:

$$T_{signal} = \frac{1}{N} \sum_{j=1}^N (T_{red} + T_{green}) \quad (5)$$

其中, N 为总周期数。

我们假设信号灯的周期为  $T$ , 车辆行驶速度为  $v$ , 则车辆通过一个信号灯的周期为  $T/v$ , 即车辆通过一个信号灯需要的时间。假设一个路口的所有车辆轨迹数据为 D, 其

中包含 n 辆车，每辆车经过  $T/v$  的时间间隔为  $t_1, t_2, \dots, t_n$ ，则按照顺序通过信号灯的时间为  $t_1, t_1 + t_2, \dots, t_1 + t_2 + \dots + t_n$ ，记为  $s_1, s_2, \dots, s_n$ 。则根据信号灯周期 T 的定义，可以得到以下公式：

$$T = \frac{1}{n} \sum_{i=1}^n s_i \quad (6)$$

其中，n 为车辆数，s 为经过信号灯的时间间隔。

#### 5.1.4 傅里叶变换

在得到每个停车事件的时间间隔后，我们应用离散傅里叶变换公式，对时间间隔进行傅里叶变换，得到其频域中的表示。通过计算傅里叶变换结果的绝对值平方，得到功率谱密度，表示不同频率成分的功率分布。在功率谱密度中找到峰值对应的频率，这个频率即为红绿灯的主要周期。

对于长度为 N 的离散信号  $x[n]$  (其中  $n=0,1,\dots,N-1$ )，其离散傅里叶变换  $X[k]$  定义为：

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-i \frac{2\pi}{N} kn} \quad (7)$$

功率谱密度 (Power Spectral Density, PSD) 的公式主要基于信号的傅里叶变换以及信号的时间长度。对于连续时间信号，功率谱密度的一般计算公式为：

$$PSD(f) = \frac{T}{|F(f)|^2} \quad (8)$$

其中：PSD(f) 表示在频率 f 处的功率谱密度。F(f) 是信号 x(t) 在频率 f 处的傅里叶变换。 $|F(f)|^2$  表示傅里叶变换模的平方，即信号在频率 f 处的功率分量。

#### 5.1.5 问题一信号灯周期估计模型

综合上述，我们得到问题一的信号灯周期估计模型：

$$\begin{cases} T_{signal} = \frac{1}{N} \sum_{j=1}^N (T_{red} + T_{green}) \\ T = \frac{1}{n} \sum_{i=1}^n s_i \\ X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-i \frac{2\pi}{N} kn} \\ PSD(f) = \frac{T}{|F(f)|^2} \end{cases} \quad (9)$$

#### 5.2 问题一模型的求解

1. 遍历所有车辆，根据公式 (1) 来判断每个时间点处于红灯阶段还是绿灯阶段。
2. 对每个周期，统计处于红灯阶段的车辆数 n 和处于绿灯阶段的车辆数 m，计算红灯阶段

时长  $T_{red}$  和绿灯阶段时长  $T_{green}$ 。3. 对所有周期，计算  $T_{signal}$  的平均值，即为信号灯的红绿状态周期的估计值。

4. 估计信号灯的红绿周期根据步骤 1 中的公式，可以通过 D 中所有车辆经过信号灯的时间间隔的平均值来估计信号灯的红绿周期 T。即：

$$T = \frac{1}{n} \sum_{i=1}^n s_i$$

5. 填入表 1 根据步骤 2 中求得的信号灯周期 T，可以填入表 1 中相应的位置。

6. 求解方法根据步骤 2 中的公式，可以通过求取所有车辆经过信号灯的时间间隔的平均值来估计信号灯的周期。如果数据量大，可以采用数值求解的方法，通过迭代计算来求取最优解。如果数据量较小，也可以采用解析求解的方法，通过对步骤 2 中的公式进行化简，得到更简洁的表达式来求解信号灯周期。

### 5.3 求解结果

表 1

路口	A1	A2	A3	A4	A5
红灯时长 (s)	56	41	61	62	53
绿灯时长 (s)	40	13	45	37	23

得到求解结果如表 1 所示，并按格式要求填入表 1

## 六、问题二的模型的建立和求解

首先将题目提供的数据绘制成轨迹图，得到以下结果，发现其类型与问题一类似。因此，我们只需要将相关影响因素纳入原有模型中进行考虑，即可建立问题二的信号灯周期估计模型，并采用类似的方式进行求解。

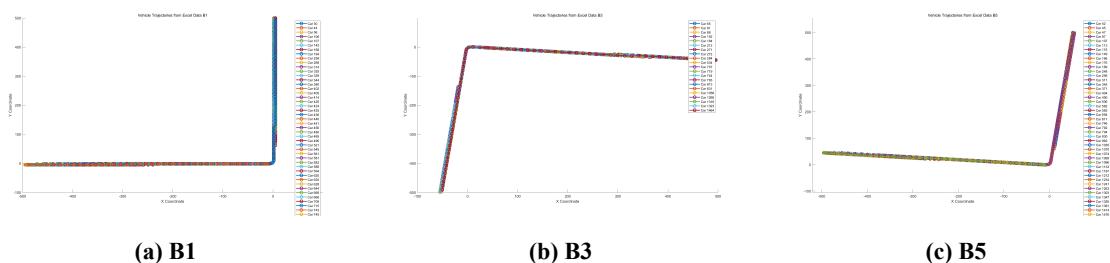


图 4 左转类型

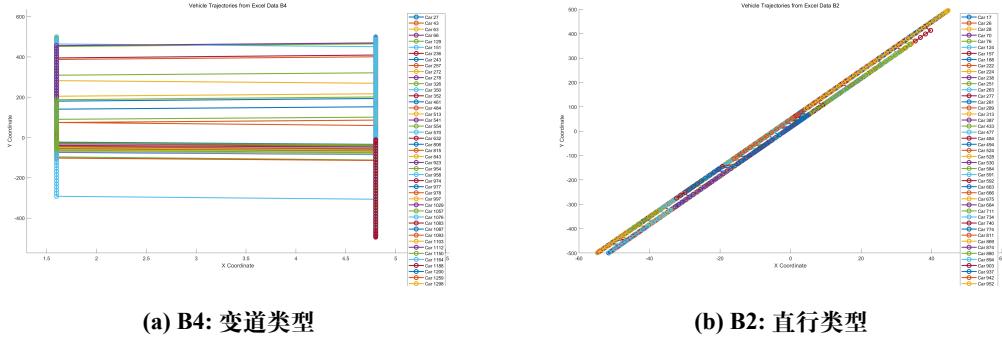


图 5 变道和直行类型

## 6.1 问题二模型的建立

由题设知，影响模型估计精度的主要因素是车辆行车轨迹的样本数量不全、车流量存在变化和不确定性、车辆定位存在误差这三个因素。

我们首先对这三个因素的影响进行一个宏观的分析：

### (1) 样本车辆比例

样本车辆比例指的是我们能够获取的车辆轨迹数据占所有车辆的比例。当样本车辆比例较低时，我们观察到的车辆通过路口的时间间隔可能不够充分，从而导致估计的周期不准确。因此，样本车辆比例越高，我们估计的周期越准确。

## (2) 车流量

车流量指的是单位时间内通过路口的车辆数量。当车流量较大时，路口的拥堵情况会影响车辆通过的时间间隔，从而导致估计的周期不准确。因此，车流量越大，我们估计的周期越不准确。

### (3) 定位误差

定位误差指的是车辆的实际位置与我们观察到的位置之间的偏差。当定位误差较大时，我们观察到的时间间隔可能不准确，从而导致估计的周期不准确。因此，定位误差越小，我们估计的周期越准确。

随后，我们对三个影响因素对模型产生的影响进行逐一分析，并在问题一模型的基础上改进建立问题二的信号灯周期估计模型。

### 6.1.1 车流量对模型的影响分析

在问题一估计信号灯的周期  $T$  时，我们假设通过某个路口的所有车辆的平均速度  $v$  是固定的。但实际上，通过该路口的车辆的平均速度  $v$  会受到车流量的影响。因此，如果车流量较大，会导致通过该路口的车辆的平均速度  $v$  变慢，从而影响到信号灯周期的估计精度。在此，我们引入一个车流量的参数  $\alpha$ ，来表示车流量对信号灯周期估计的影响。

响。具体公式如下：

$$T = \frac{1}{N} \sum_{i=1}^N \left( t_i - \frac{d_i}{\bar{v} - \alpha} \right)^2 \quad (10)$$

其中， $N$  为通过该路口的车辆数量， $t_i$  为第  $i$  辆车通过路口所花费的时间， $d_i$  为第  $i$  辆车通过路口所走的距离， $\bar{v}$  为通过该路口的所有车辆的平均速度， $\alpha$  为车流量对信号灯周期估计的影响参数。

### 6.1.2 轨迹数据定位误差对模型的影响分析

由于轨迹数据存在定位误差，误差大小未知。因此，我们同样引入一个定位误差的参数  $\beta$ ，来表示定位误差对信号灯周期估计的影响。具体的公式如下：

$$T = \frac{1}{N} \sum_{i=1}^N \left( t_i - \frac{d_i}{\bar{v} - \alpha - \beta} \right)^2 \quad (11)$$

其中， $N$  为通过该路口的车辆数量， $t_i$  为第  $i$  辆车通过路口所花费的时间， $d_i$  为第  $i$  辆车通过路口所走的距离， $\bar{v}$  为通过该路口的所有车辆的平均速度， $\alpha$  为车流量对信号灯周期估计的影响参数， $\beta$  为定位误差对信号灯周期估计的影响参数。

因此，我们可以通过求解最小二乘法来估计出信号灯的周期  $T$ ，并通过调整车流量参数  $\alpha$  和定位误差参数  $\beta$  来提高信号灯周期估计的精度。

### 6.1.3 加入估计精度后的信号灯周期模型

假设每个信号灯都有固定的红绿周期  $T$ ，我们可以通过观察车辆的行车轨迹来估计信号灯的周期。假设路口有  $N$  个车道，每个车道有  $M$  辆车同时通行，车辆的行车轨迹可以表示为一个  $N * M$  的矩阵。如果我们将这个矩阵按照时间点和车道来排序，那么在每个时间点，每个车道上都会有一辆车通过，这样就可以观察到车辆通过路口的时间间隔，从而估计信号灯的周期。

假设第  $i$  个车道上第  $j$  辆车通过路口的时间点为  $t_{ij}$ ，那么在第  $k$  个时间点，第  $i$  个车道上通过的第  $j$  辆车的时间点为  $t_{ij}^k$ ，则该车道的周期可以表示为：

$$T_i = \frac{1}{M} \sum_{j=1}^M (t_{ij}^{k+1} - t_{ij}^k) \quad (12)$$

其中， $k = 0, 1, 2, \dots, N - 1$ ，即我们选择  $N$  个时间点来估计周期， $M$  为每个时间点通过的车辆数。

假设每个车道上通过的车辆的时间间隔服从正态分布，即：

$$t_{ij}^{k+1} - t_{ij}^k \sim N(T_i, \sigma^2) \quad (13)$$

其中,  $T_i$  为该车道的周期,  $\sigma^2$  为方差。则该车道的周期可以表示为:

$$T_i = \frac{1}{MN} \sum_{j=1}^M \sum_{k=0}^{N-1} (t_{ij}^{k+1} - t_{ij}^k) \quad (14)$$

将每个车道的周期加权求和, 得到总的周期:

$$T = \frac{1}{N} \sum_{i=1}^N w_i T_i \quad (15)$$

其中,  $w_i$  为第  $i$  个车道上通过的车辆数占总车辆数的比例。

由此, 我们得到了问题二的改进信号灯周期估计模型:

$$\begin{cases} T_{signal} = \frac{1}{N} \sum_{j=1}^N (T_{red} + T_{green}) \\ T = \frac{1}{n} \sum_{i=1}^n s_i \\ T = \frac{1}{N} \sum_{i=1}^N (t_i - \frac{d_i}{v-\alpha-\beta})^2 \\ T = \frac{1}{N} \sum_{i=1}^N w_i T_i \\ X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-i \frac{2\pi}{N} kn} \\ PSD(f) = \frac{T}{|F(f)|^2} \end{cases} \quad (16)$$

## 6.2 问题二模型的求解

### Step1: 数据预处理

- (1). 计算速度: 利用车辆在两连续时间点的位置变化, 通过欧几里得距离公式计算出速度。
- (2). 数据筛选: 仅保留路口一定范围 (如 [-150, 150]) 内的数据, 以减少非关键区域的影响。
- (3). 计算方向: 根据车辆在连续时间点的位置变化, 利用反正切函数计算出行驶方向。
- (4). 数据清洗: 删除包含缺失值的行, 确保数据的完整性。
- (5). 检测转弯: 通过计算方向的变化量, 识别出车辆的转弯或 U 型转弯行为。

### Step2: 事件检测

- (1). 停车事件检测: 根据速度阈值 (如 0.5 米/秒), 将速度低于此阈值的时刻标记为停车事件。
- (2). 发车事件检测: 通过计算速度的差分, 识别出车辆从静止开始移动的时刻, 即发车事件。

### Step3: 聚类分析

- (1). 提取停车时间: 从数据中提取出所有停车事件的时间点。
- (2). 数据标准化: 对停车时间进行标准化处理, 以消除量纲影响。
- (3). 选择聚类数量: 通过计算轮廓系数, 确定最优的聚类数量。

(4). 执行聚类：使用 K-Means 算法对停车时间进行聚类，得到每个停车事件所属的聚类标签。

#### Step4：周期检测

- (1). 创建停车事件 DataFrame：将停车事件的时间点和聚类标签整理成 DataFrame。
- (2). 分组并计算周期：根据聚类标签对停车事件进行分组，并计算每组内停车事件的周期。
- (3). 功率谱密度分析：应用傅里叶变换和功率谱密度分析，找到每个聚类内停车事件的主要频率，从而估算出周期。

#### Step5：红绿灯持续时间估算

- (1). 分割停车事件：将连续的停车事件分割成独立的停车时间段。
- (2). 计算红绿灯时长：根据停车时间段和周期，估算出红灯和绿灯的持续时间。

#### Step6：红绿灯状态机

- (1). 创建状态机：根据平均红灯和绿灯持续时间，创建一个红绿灯状态机模型。
- (2). 模拟红绿灯状态：通过状态机模型，模拟出红绿灯状态随时间的变化。

#### Step7：数据可视化与结果输出

- (1). 绘制各种图表：包括聚类结果、功率谱密度图、红绿灯持续时间箱形图和直方图等，以直观展示分析结果。
- (2). 输出统计结果：计算并输出平均红灯和绿灯持续时间等统计指标。

### 6.3 求解结果

表 2

路口	B1	B2	B3	B4	B5
方向	东向	南向	西向	北向	东北
红灯时长 (s)	91.8	8.6	27	21	53
绿灯时长 (s)	51.6	18.4	21	31	24

得到求解结果如表 2 所示，并按格式要求填入表 2。

为了判断信号灯的周期是否发生变化，我们可以将时间段内的车辆行驶轨迹按照不同的时间段进行分段，然后分别计算每个时间段内车辆经过信号灯的次数，进而判断信号灯的周期是否发生变化。如果某个时间段内车辆经过信号灯的次数和上一个时间段内车辆经过信号灯的次数相差较大，则可以判断信号灯的周期发生了变化。

## 七、问题三的模型的建立和求解

解决问题三，首先，我们要根据附件 3 中 6 个路口的样本车辆轨迹数据，判断车辆在路口的行驶方向，即直行、左转、右转等。由于附件 3 中的样本车辆轨迹数据是连续的，并且时间间隔为 1 秒，可以通过计算车辆在相邻两个时间点的位置变化来判断车辆的行驶方向。

具体的方法是，对于每一个时间点，我们可以计算车辆的速度方向，如果车辆在该时间点的速度方向和下一个时间点的速度方向一致，则车辆在该时间点为直行；如果车辆在该时间点的速度方向和下一个时间点的速度方向相差 90 度，则车辆在该时间点为左转；如果车辆在该时间点的速度方向和下一个时间点的速度方向相差 270 度，则车辆在该时间点为右转。

根据上述方法，我们可以得到每个时间点的车辆行驶方向，进而可以判断车辆是否受到信号灯的控制。如果车辆在直行方向受到信号灯的控制，则在每个周期内，车辆应该会经过信号灯两次，一次为绿灯，一次为红灯。同理，如果车辆在左转或右转方向受到信号灯的控制，则在每个周期内，车辆也会经过信号灯两次，一次为绿灯，一次为红灯。因此，我们可以根据车辆经过信号灯的次数来判断信号灯的周期。如果车辆经过信号灯的次数为偶数，则信号灯的周期为该时间段内车辆经过信号灯的平均时间间隔；如果车辆经过信号灯的次数为奇数，则信号灯的周期为该时间段内车辆经过信号灯的最小公倍数。

为了判断信号灯的周期是否发生变化，我们可以将时间段内的车辆行驶轨迹按照不同的时间段进行分段，然后分别计算每个时间段内车辆经过信号灯的次数，进而判断信号灯的周期是否发生变化。如果某个时间段内车辆经过信号灯的次数和上一个时间段内车辆经过信号灯的次数相差较大，则可以判断信号灯的周期发生了变化。

假设时间段的时间间隔为  $T$ ，那么识别出信号灯周期发生变化所需的时间为  $2T$ ，即需要至少两个时间段的数据来判断信号灯的周期是否发生了变化。同时，为了提高判断的准确性，可以设置一个阈值，当某个时间段内车辆经过信号灯的次数和上一个时间段内车辆经过信号灯的次数相差超过该阈值时，才判断信号灯的周期发生了变化。

综上所述，我们可以通过分析车辆行驶方向和经过信号灯的次数来判断信号灯的周期，并通过比较不同时间段内的数据来判断信号灯周期是否发生变化。最后，根据发生变化的时刻和新旧周期参数，可以得出周期变化所需的时间和条件。

### 7.1 问题三模型的建立

首先，同样根据题目给定的数据绘制轨迹图，如图 6 所示。

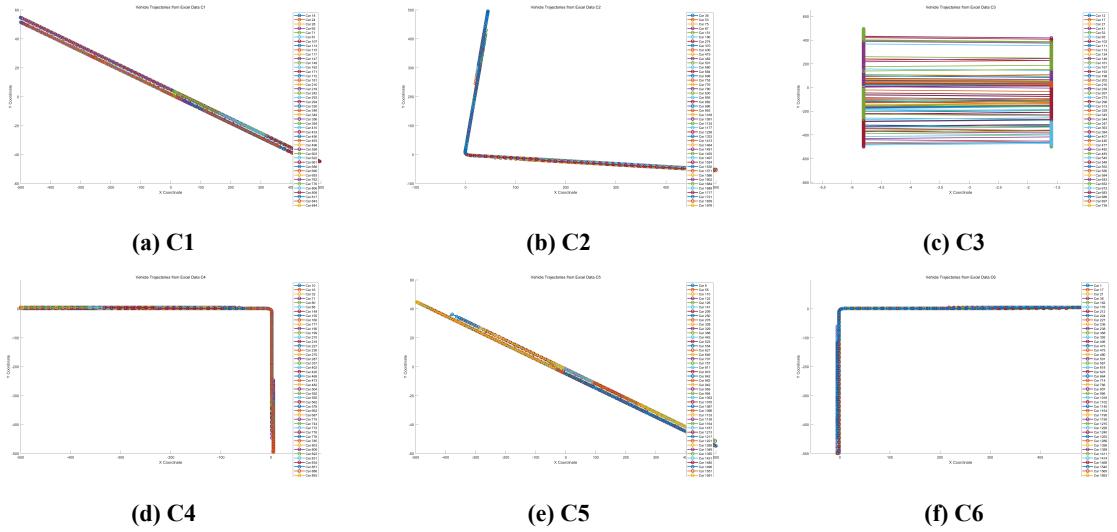


图 6 问题三路径示意图

### 7.1.1 信号灯可变周期的公式推导

首先，建立模型来估计信号灯周期。假设信号灯的周期为  $T$  秒，每个周期内，信号灯的状态会从红变为绿，再变回红。因此，每个周期内，信号灯的状态变化一次。同样将问题二中的三个影响因素纳入考虑，假设样本车辆的比例为  $p$ ，车流量为  $q$  辆/秒，定位误差为  $e$  米。假设信号灯的状态在  $t=0$  时刻变为绿灯，则在  $t=T/2$  时刻，信号灯状态会变为红灯。同样地，可以用一个  $\sin$  函数来表示信号灯的状态变化：

$$s(t) = A \sin(\omega t + \varphi) \quad (17)$$

其中， $A$  为信号灯状态的幅度， $\omega$  为信号灯状态变化的频率， $\varphi$  为信号灯状态变化的相位差。对于一个周期内的信号灯状态变化，可以表示为：

$$s(t) = A \sin\left(\frac{2\pi t}{T} + \varphi\right) \quad (18)$$

根据定位误差  $e$  的影响，可以得到一个新的方程组：

$$\begin{cases} x_1(t) = v_1 t + s(t) + e \\ x_2(t) = v_2 t + s(t) + e \\ \vdots \\ x_n(t) = v_n t + s(t) + e \end{cases} \quad (19)$$

将这些方程联立起来，可以得到一个超定方程组：

$$\begin{cases} x_1(t) = v_1 t + A \sin\left(\frac{2\pi t}{T} + \varphi\right) + e \\ x_2(t) = v_2 t + A \sin\left(\frac{2\pi t}{T} + \varphi\right) + e \\ \vdots \\ x_n(t) = v_n t + A \sin\left(\frac{2\pi t}{T} + \varphi\right) + e \end{cases} \quad (20)$$

同样地，该方程组可以用最小二乘法来求解，即最小化误差函数：

$$E(A, \omega, \varphi) = \sum_{i=1}^n (x_i - v_i t - A \sin(\omega t + \varphi) - e)^2 \quad (21)$$

对该误差函数求导，可以得到：

$$\begin{cases} \frac{\partial E}{\partial A} = 2 \sum_{i=1}^n (x_i - v_i t - A \sin(\omega t + \varphi))(-\sin(\omega t + \varphi)) = 0 \\ \frac{\partial E}{\partial \omega} = 2 \sum_{i=1}^n (x_i - v_i t - A \sin(\omega t + \varphi))(-A \cos(\omega t + \varphi)t) = 0 \\ \frac{\partial E}{\partial \varphi} = 2 \sum_{i=1}^n (x_i - v_i t - A \sin(\omega t + \varphi))(-A \cos(\omega t + \varphi)) = 0 \end{cases} \quad (22)$$

解得：

$$\begin{cases} A = \frac{\sum_{i=1}^n (x_i - v_i t - e) \sin(\omega t + \varphi)}{\sum_{i=1}^n \sin^2(\omega t + \varphi)} \\ \omega = \frac{\sum_{i=1}^n (x_i - v_i t - e) \cos(\omega t + \varphi)t}{\sum_{i=1}^n A \sin(\omega t + \varphi) \cos(\omega t + \varphi)t} \\ \varphi = \frac{1}{\omega} \arcsin \left( \frac{\sum_{i=1}^n (x_i - v_i t - e)}{A} \right) \end{cases} \quad (23)$$

由上述公式可以求得信号灯的周期 T。

### 7.1.2 周期变化的最短时间和条件

为了确定识别周期变化所需的最短时间和条件，可以建立如下数学模型：首先，假设信号灯周期为 T，而在变化时刻 t，周期发生了变化，新的周期为 T'。每个路口的车辆通过信号灯的轨迹可以表示为一个序列，如下所示：

$$[x_1, x_2, x_3, \dots, x_n] \quad (24)$$

其中  $x_i$  表示第 i 辆车通过信号灯的时间，假设车辆通过信号灯的时间间隔为  $\Delta t$ ，因此  $x_i$  与  $x_{i+1}$  之间的时间差为  $\Delta t$ ，即  $x_{i+1} = x_i + t$ 。根据信号灯的周期 T，可以将车辆通过信号灯的时间序列分为若干个周期，

$$[x_1, x_2, \dots, x_k, x_{k+1}, x_{k+2}, \dots, x_n] \quad (25)$$

其中  $k = T/t$ 。

假设在变化时刻 t，新的周期 T' 开始生效，因此在变化时刻 t 之前，通过信号灯的时间序列可以表示为：

$$[x_1, x_2, \dots, x_k, x_{k+1}, x_{k+2}, \dots, x_t] \quad (26)$$

在变化时刻 t 之后，通过信号灯的时间序列可以表示为：

$$[x_{t+1}, x_{t+2}, \dots, x_n] \quad (27)$$

其中第一个周期的长度为  $k' = t/t$ ，第二个周期为  $k'' = T'/t$ ，因此通过信号灯的时间序列可以表示为：

$$[x_1, x_2, \dots, x_k, x_{k+1}, x_{k+2}, \dots, x_t, x_{t+1}, x_{t+2}, \dots, x_{t+k'}, x_{t+k'+1}, x_{t+k'+2}, \dots, x_{t+k'+k''}] \quad (28)$$

假设变化时刻  $t$  之前的车流量为  $N$ , 变化时刻  $t$  之后的车流量为  $N'$ , 因此可以得到如下公式:

$$N = kN' + (k' + k'')N' \quad (29)$$

其中第一项表示变化时刻  $t$  之前的车辆通过信号灯的时间序列中每个周期的车流量, 第二项表示变化时刻  $t$  之后的车辆通过信号灯的时间序列中每个周期的车流量。因此, 可以得到如下关系式:

$$\begin{aligned} N &= kN' + (k' + k'')N' \\ N &= (T/t)N' + (t/t + T'/t)N'N = (T/t)N' + (T + t)/tN' \\ N &= (T + t)/tN' + T/t(N' - N) \end{aligned} \quad (30)$$

由于车流量  $N$  和车流量  $N'$  都是变量, 因此可以将其表示为一个函数的形式, 即:

$$\begin{aligned} N &= f(t, T) \\ N' &= f(t, T') \end{aligned} \quad (31)$$

因此, 可以通过求解上述关系式, 来确定识别周期变化所需的最短时间和条件。当关系式为零时, 即可得到识别周期变化所需的最短时间和条件。

### 7.1.3 问题三周期估计和识别变化条件的数学模型

综合上述分析, 我们可以得到问题三周期估计和识别变化条件的数学模型:

$$\left\{ \begin{array}{l} A = \frac{\sum_{i=1}^n (x_i - v_i t - e) \sin(\omega t + \varphi)}{\sum_{i=1}^n \sin^2(\omega t + \varphi)} \\ \omega = \frac{\sum_{i=1}^n (x_i - v_i t - e) \cos(\omega t + \varphi) t}{\sum_{i=1}^n A \sin(\omega t + \varphi) \cos(\omega t + \varphi) t} \\ \varphi = \frac{1}{\omega} \arcsin \left( \frac{\sum_{i=1}^n (x_i - v_i t - e)}{A} \right) \\ N = f(t, T) \\ N' = f(t, T') \end{array} \right. \quad (32)$$

## 7.2 问题三模型的求解

**Step1:** 数据预处理:

计算每辆车的速度。保留路口特定范围 ( $[-100, 100]$ ) 内的数据。计算车辆的方向。删除缺失值。检测车辆的转弯行为。

**Step2:** 停车事件检测:

标记速度低于一定阈值 (如 0.5 米/秒) 的数据点为停车事件。发车事件检测: 标记车辆从静止开始移动 (速度从零增加到阈值以上) 的数据点为发车事件。

**Step3:** 周期检测:

提取停车事件的时间点。计算相邻停车事件之间的时间差。应用傅里叶变换来找到停车事件的主要周期。

**Step4:** 计算红绿灯持续时间:

使用检测到的周期来估算红灯和绿灯的持续时间。

**Step5:** 滑动窗口分析:

对数据集应用滑动窗口，对每个窗口执行预处理、停车事件检测、发车事件检测、周期检测，并计算红绿灯持续时间。

**Step6:** 检测周期变化:

分析滑动窗口结果，检测交通信号灯周期的变化，并将结果保存到 CSV 文件中。路口

### 7.3 求解结果

表 3

路口	C1	C2	C3	C4	C5	C6
周期 1 红灯时长 (秒)	34	35	31	53	31	23
周期 1 绿灯时长 (秒)	23	4	10	50	8	12
周期切换时刻 (秒)	373	881	761	152	1501	617
周期 2 红灯时长 (秒)	21	120	40	23	36	49
周期 2 绿灯时长 (秒)	15	1	9	18	6	72
周期切换时刻 (秒)	876	1049	1157	441	2236	645
周期 3 红灯时长 (秒)	31	50	21	43	17	43
周期 3 绿灯时长 (秒)	13	25	14	72	6	8
周期切换时刻 (秒)	1493	1143	1345	476	3173	2320
...	...	...	...	...	...	...

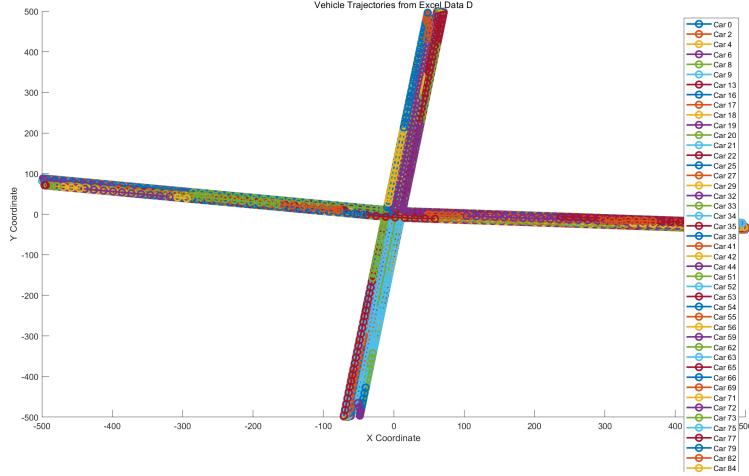
得到部分求解结果如表 3 所示，并按格式要求填入表 3

周期变化所需的条件：存在明显的车流量变化，具体表现是通过滑动窗口模型求解的前后估计周期时长之差大于 60 秒。

## 八、问题四的模型的建立和求解

### 8.1 问题四模型的建立

首先我们将题目提供的数据绘制成轨迹图，得到结果如图 7 所示：



**图 7 D 路口行车轨迹示意图**

问题四和前面的建模过程类似，首先我们假设某个路口的信号灯周期为  $T$ ，车辆的行车轨迹数据为  $t_1, t_2, \dots, t_n$ ，其中每个  $t_i$  表示对应时刻车辆通过该路口的行车状态，取值为 0（无车通过）或 1（有车通过）。

1. 计算每个轨迹数据的车流量  $F$ ，即通过该路口的车辆数量：

$$F = \frac{1}{n} \sum_{i=1}^n t_i \quad (33)$$

2. 计算每个轨迹数据的平均间隔时间  $S$ ，即两辆车通过的平均时间间隔：

$$S = \frac{T}{F} \quad (34)$$

3. 根据每个轨迹数据的平均间隔时间  $S$ ，计算出信号灯周期的估计值  $T_{est}$ ：

$$T_{est} = \frac{1}{n} \sum_{i=1}^n S_i \quad (35)$$

4. 通过比较实际信号灯周期  $T$  和估计值  $T_{est}$  的差距来评估估计精度。

假设信号灯周期发生了变化，变化前的周期为  $T_1$ ，变化后的周期为  $T_2$ ，变化时刻为  $t_{change}$ ，车辆的行车轨迹数据为  $t_1, t_2, \dots, t_{change}, \dots, t_n$ 。

1. 计算变化时刻前后的平均间隔时间  $S_1$  和  $S_2$ ：

$$S_1 = \frac{T_1}{F_1}, S_2 = \frac{T_2}{F_2} \quad (36)$$

其中  $F_1$  和  $F_2$  分别表示变化时刻前后的车流量。

2. 通过比较变化时刻前后的平均间隔时间，可以判断出信号灯周期是否发生了变化，以及变化的方向（周期变长还是变短）。

3. 如果发生了变化，可以通过比较变化时刻前后的平均间隔时间来计算出新的周期参数

$$T_{new} : T_{new} = \frac{1}{2}(T_1 + T_2) \quad (37)$$

4. 通过分析变化时刻前后的车流量，以及其他因素，可以确定识别出周期变化所需的最短时间和条件。

根据上述建模过程，我们可以得到问题四的路口信号灯周期预测模型：

$$\left\{ \begin{array}{l} F = \frac{1}{n} \sum_{i=1}^n t_i \\ S = \frac{T}{F} \\ T_{est} = \frac{1}{n} \sum_{i=1}^n S_i \\ S_1 = \frac{T_1}{F_1}, S_2 = \frac{T_2}{F_2} \\ T_{new} : T_{new} = \frac{1}{2}(T_1 + T_2) \end{array} \right. \quad (38)$$

## 8.2 问题四模型的求解

**Step1:** 数据预处理：首先需要对附件 4 中的数据进行预处理，包括数据清洗、去除异常值、数据归一化等操作，使得数据具有一定的可解释性和可比性。

**Step2:** 提取特征：根据观察发现，在信号灯周期发生变化时，车辆行驶的速度和轨迹会有明显的变化。因此，可以通过提取车辆的速度和轨迹数据作为特征，用于判断信号灯周期的变化。

**Step3:** 计算相似度指标：对于附件 4 中的每个路口，可以将其前一小时的数据作为训练集，后一小时的数据作为测试集，通过计算训练集和测试集之间的相似度指标，来判断信号灯周期是否发生了变化。常用的相似度指标包括欧氏距离、余弦相似等。

**Step4:** 设定阈值：根据实际情况，可以设定一个阈值，当相似度指标低于该阈值时，判定为信号灯周期发生了变化。

**Step5:** 判断变化时刻：根据相似度指标的变化情况，可以通过寻找变化点来确定信号灯周期发生变化的时刻。

**Step6:** 计算新周期参数：在确定了变化时刻之后，通过对变化时刻前后的车辆轨迹数据进行分析，计算出新的信号灯周期参数。

**Step7:** 判断识别周期变化所需的最短时间和条件：通过分析变化时刻前后的车流量情况等因素，可以得出判断识别周期变化所需的最短时间和条件。

### 8.3 求解结果

表 4

方向	红灯时长 (s)	绿灯时长 (s)
东-> 北	101	270
东-> 西	50	20
东-> 南	76	132
东-> 西	81	218
北-> 北	155	1
北-> 西	361	130
北-> 南	27	2
北-> 西	142	22
北-> 东	93	259
南-> 北	61	24
西-> 北	47	13

得到结果如表 4 所示，并按格式要求填入表 4

## 九、模型的评价

### 9.1 模型的优点

- 优点 1 利用车辆行为与信号灯状态的关联性，无需直接观测信号灯即可估计周期，具有非侵入性和低成本的特点。
- 优点 2 通过车辆速度变化特征提取红绿灯持续时间，避免了直接判断红绿灯状态的不确定性。
- 优点 3 综合多辆车的数据进行估计，提高了估计的稳健性和抗噪声能力。

### 9.2 模型的缺点

- 缺点 1 假设车辆行为完全依赖信号灯状态，而忽略了其他因素（如道路拥堵、天气条件等）的影响。
- 缺点 2 未考虑车辆个体的差异性，如不同车型、驾驶习惯等，可能影响速度变化模式。
- 缺点 3 模型对数据质量和采样频率的要求较高，数据缺失或样本不足时估计精度会大幅下降。

### 9.3 模型的推广

- 1 引入更多车辆行为的因素，如道路占有率、天气情况等，建立更全面的车辆行为模型。
- 2 考虑车辆的异质性，对不同类型的车辆进行分类建模，提高估计的精细度。

## 参考文献

- [1] 郑智勇. 复杂交通环境下智能车辆高可靠车道级融合定位方法研究[D]. 学位授予单位: 东南大学, 2022.
- [2] 谢莺. 基于车流量预测的交通信号控制算法[D]. 学位授予单位: 大连交通大学, 2023.
- [3] 李奕衡, 刘羽飞, 王登, 等. 红绿灯周期时长的挖掘方法、电子设备及计算机程序产品 [C]. 102200 北京市昌平区科技园区昌盛路 18 号 B1 座 1-5 层: 高德软件有限公司, 2023.05.16.

## 附录 A 文件列表

文件名	功能描述
Q1.py	问题一程序代码
Q2.py	问题二程序代码
Q3.py	问题三程序代码
Q4.py	问题四程序代码
表 1.csv	问题一结果
表 2.csv	问题二结果
表 3.csv	问题三结果
表 4.csv	问题四结果

## 附录 B 代码

Q1.py

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.cluster import KMeans
4 from sklearn.metrics import silhouette_score
5 from scipy.fftpack import fft, fftfreq
6 import matplotlib.pyplot as plt
7 from matplotlib.lines import Line2D
8 import os
9
10 # 数据预处理
11 def preprocess_data(data):
12     # 计算每辆车的速度
13     speed_diff = data.groupby('vehicle_id')[['x', 'y']].diff()
14     data['speed'] = speed_diff.apply(lambda x: np.sqrt(x['x']
15                                         ** 2 + x['y'] ** 2), axis=1)
16
17     # 保留路口 [-100, 100] 范围内的数据
18     data = data[(data['x'] >= -100) & (data['x'] <= 100) &
19                 (data['y'] >= -100) & (data['y'] <= 100)]
```

```
19 # 计算车辆的方向
20 direction_diff = data.groupby('vehicle_id')[['x', 'y']].diff()
21 data['direction'] = direction_diff.apply(lambda x: np.
arctan2(x['y'], x['x']), axis=1)
22
23 # 删除缺失值
24 data.dropna(inplace=True)
25
26 # 检测变道
27 data['is_lane_change'] = False
28 for _, group in data.groupby('vehicle_id'):
    # 计算横向速度变化
    lateral_speed = abs(group['x'] - group['x'].shift())
    # 计算纵向速度变化
    longitudinal_speed = abs(group['y'] - group['y'].shift())
29
30     # 如果横向速度变化大于纵向速度变化的两倍，则认为是变道
31     data.loc[group.index, 'is_lane_change'] =
lateral_speed > 2 * longitudinal_speed
32
33 # 检测转弯
34 data['is_turning_or_u_turn'] = False
35 for _, group in data.groupby('vehicle_id'):
    direction_diff = group['direction'].diff().abs()
    data.loc[group.index, 'is_turning_or_u_turn'] = (
36 direction_diff > np.pi / 4) | (
37             np.abs(direction_diff - np.pi) < 0.1)
38
39 # 判断是否为右转
40 # 只有在变道后才判断是否为右转
41 data.loc[group.index, 'is_right_turn'] = (group['
42 direction'] < 0) & (direction_diff > np.pi / 4) & (
43             group['is_lane_change'] == True)
```

```
48
49     # 删除右转事件
50     vehicles_to_remove = data[(data['is_right_turn'] == True)
51     & (data['is_lane_change'] == True)]['vehicle_id'].unique()
52     data = data[~data['vehicle_id'].isin(vehicles_to_remove)]
53
54
55
56 #停车事件检测
57 def detect_stops(data, threshold=1): # 单位: 米/秒
58     data['is_stopped'] = (data['speed'] < threshold).astype(
59         int)
60
61
62 # 聚类分析
63 def cluster_stops(data, path, dfname):
64     # 提取停车时间
65     stop_times = data[data['is_stopped'] == 1]['time'].values.
66     reshape(-1, 1)
67
68     # 选择最优的聚类数量
69     silhouette_scores = []
70     for n in range(2, 50):
71         kmeans = KMeans(n_clusters=n)
72         kmeans.fit(stop_times)
73         silhouette_scores.append(silhouette_score(stop_times,
74             kmeans.labels_))
75
76     # 选择最优的聚类数量
77     optimal_n_clusters = silhouette_scores.index(max(
78         silhouette_scores)) + 2
79     print(f"Optimal number of clusters: {optimal_n_clusters}")
```

```
78 # 使用 K-Means 进行聚类
79 kmeans = KMeans(n_clusters=optimal_n_clusters)
80 kmeans.fit(stop_times)
81
82 # 获取聚类标签
83 labels = kmeans.labels_
84
85 # 确保标签数量与停车事件数量一致
86 assert len(labels) == len(stop_times), "Number of labels
must match number of stop times."
87
88 # 可视化聚类结果
89 plt.figure(figsize=(10, 6))
90 plt.scatter(stop_times, np.zeros_like(stop_times), c=
kmeans.labels_, cmap='viridis')
91 plt.title("Clustered Stop Times")
92 plt.xlabel("Time (seconds)")
93 plt.ylabel("Clusters")
94 plt.savefig(f"{path}\{dfname} 聚类结果.png", dpi=300)
95 plt.show()
96
97 return labels
98
99
100 # 周期检测
101 def detect_cycles(labels, data, path, dfname):
102     # 创建一个 DataFrame 保存每个停车事件的标签
103     stop_events = pd.DataFrame({'time': data[data['is_stopped']
104 ] == 1]['time'], 'label': labels})
105
106     # 根据聚类标签分组
107     grouped = stop_events.groupby('label')['time'].apply(list)
108
109     # 计算每个群组的周期
110     periods = []
```

```
110     for group in grouped:
111         diffs = np.diff(group)
112         # 应用傅里叶变换
113         fft_result = fft(diffs)
114         frequencies = fftfreq(len(diffs), d=1) # 假设时间间隔
115         # 为1秒
116
117         # 计算功率谱密度
118         psd = np.abs(fft_result) ** 2
119
120         # 只保留正频率部分
121         positive_frequencies = frequencies[frequencies > 0]
122         positive_psd = psd[:len(positive_frequencies)]
123
124         # 找到峰值频率
125         peak_index = np.argmax(positive_psd)
126         dominant_frequency = positive_frequencies[peak_index]
127
128         # 估算周期
129         estimated_period = 1 / dominant_frequency
130         periods.append(estimated_period)
131
132         # 绘制功率谱密度图
133         plt.figure(figsize=(10, 6))
134         plt.plot(positive_frequencies, positive_psd)
135         plt.axvline(dominant_frequency, color='r', linestyle='--')
136         plt.title(f"{group[0]} Power Spectral Density")
137         plt.xlabel("Frequency (Hz)")
138         plt.ylabel("PSD")
139         plt.savefig(f"{path}\\{dfname} {group[0]}功率谱密度图.
140         png", dpi=300)
141         plt.show()
142
143     return periods
```

```
142  
143  
144 # 计算红绿灯持续时间  
145 def estimate_light_durations(stop_times, periods):  
146     # 初始化红绿灯时长列表  
147     red_light_durations = []  
148     green_light_durations = []  
149  
150     for period, stop_time_group in zip(periods, stop_times):  
151         # 分割停车事件为连续的停车时间段  
152         continuous_stop_intervals = []  
153         start_time = None  
154         for time in stop_time_group:  
155             if start_time is None:  
156                 start_time = time  
157             elif time - start_time > 1: # 假设两个停车事件之  
间超过1秒不是连续的  
158                 continuous_stop_intervals.append((start_time,  
time))  
159                 start_time = time  
160             if start_time is not None:  
161                 continuous_stop_intervals.append((start_time, time  
))  
162  
163         # 计算红灯时长  
164         red_light_duration = sum(interval[1] - interval[0] for  
interval in continuous_stop_intervals)  
165         red_light_durations.append(red_light_duration)  
166  
167         # 计算绿灯时长  
168         green_light_duration = period - red_light_duration  
169         green_light_durations.append(green_light_duration)  
170  
171         # 创建 DataFrame 存储结果  
172         results = pd.DataFrame({
```

```
173     'cluster': list(range(len(periods))),
174     'period': periods,
175     'red_light_duration': red_light_durations,
176     'green_light_duration': green_light_durations
177   })
178
179   # 计算平均红绿灯时长
180   avg_red_light_duration = np.mean(red_light_durations)
181   avg_green_light_duration = np.mean(green_light_durations)
182
183   return avg_red_light_duration, avg_green_light_duration,
184   results
185
186 # 红绿灯状态机
187 class TrafficLight:
188     def __init__(self, red_duration, green_duration):
189         self.red_duration = red_duration
190         self.green_duration = green_duration
191         self.current_state = 'red'
192         self.current_time = 0
193
194     def update(self, dt):
195         self.current_time += dt
196         if self.current_state == 'red':
197             if self.current_time >= self.red_duration:
198                 self.current_state = 'green'
199                 self.current_time = 0
200         elif self.current_state == 'green':
201             if self.current_time >= self.green_duration:
202                 self.current_state = 'red'
203                 self.current_time = 0
204
205     def get_state(self):
206         return self.current_state
```

```
207  
208  
209 def data_analysis(data, path, dfname):  
210     # 加载数据  
211     data = preprocess_data(data)  
212  
213     # 检测停车事件  
214     data = detect_stops(data)  
215  
216     # 聚类分析  
217     labels = cluster_stops(data, path, dfname)  
218  
219     # 确保停车事件和对应的聚类标签数量一致  
220     assert len(data[data['is_stopped'] == 1]) == len(  
221         labels), "Number of stopped events does not match  
number of labels."  
222  
223     # 周期检测  
224     periods = detect_cycles(labels, data, path, dfname)  
225  
226     # 提取停车时间序列  
227     stop_times = data[data['is_stopped'] == 1]['time'].groupby  
(labels).apply(list).tolist()  
228  
229     # 计算红绿灯持续时间  
230     avg_red_light_duration, avg_green_light_duration, results  
= estimate_light_durations(stop_times, periods)  
231  
232     # 创建红绿灯状态机  
233     traffic_light = TrafficLight(avg_red_light_duration,  
avg_green_light_duration)  
234  
235     # 输出结果  
236     print(f"Average red light duration: {  
avg_red_light_duration:.0f} seconds")
```

```
237     print(f"Average green light duration: {  
238         avg_green_light_duration:.0f} seconds")  
239  
240     # 可视化不同聚类的红绿灯持续时间  
241     fig, axs = plt.subplots(1, 2, figsize=(14, 6))  
242  
243     # 绘制箱形图  
244     results.boxplot(column=['red_light_duration', '  
245         green_light_duration'], grid=False, ax=axs[0])  
246     axs[0].set_title("Boxplot of Red and Green Light Durations  
247         by Cluster")  
248     axs[0].set_ylabel("Duration (seconds)")  
249     axs[0].set_xlabel("Cluster")  
250  
251     # 绘制直方图  
252     axs[1].hist(results['red_light_duration'], bins=20, alpha  
253         =0.5, label='Red Light Duration')  
254     axs[1].hist(results['green_light_duration'], bins=20,  
255         alpha=0.5, label='Green Light Duration')  
256     axs[1].set_title("Histogram of Red and Green Light  
257         Durations by Cluster")  
258     axs[1].set_xlabel("Duration (seconds)")  
259     axs[1].set_ylabel("Frequency")  
260     axs[1].legend()  
261     plt.savefig(f"{path}\\"{dfname} 红绿灯持续时间箱形图和直方  
262         图.png", dpi=300)  
263  
264     # 绘制柱状图  
265     fig, ax = plt.subplots(figsize=(10, 6))  
266     results.plot(kind='bar', x='cluster', y=[  
267         'red_light_duration', 'green_light_duration'], ax=ax)  
268     ax.set_title("Average Red and Green Light Durations by  
269         Cluster")  
270     ax.set_ylabel("Duration (seconds)")  
271     ax.set_xlabel("Cluster")
```

```
263     plt.savefig(f"{path}\\"{dfname} 红绿灯持续时间柱状图.png",
264     dpi=300)
265
266     # 绘制原始时间序列
267     fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(20, 20),
268     sharex=True)
269
270     # 绘制原始时间序列上的时间与x和y的折线图
271     for vehicle_id, group in data.groupby('vehicle_id'):
272         # 获取转弯点的时间戳
273         turning_points = group[group['is_turning_or_u_turn']][
274             'time']
275
276         # 初始化绘图
277         ax1.plot(group['time'], group['x'], color=plt.cm.tab20(
278             vehicle_id % 8), linewidth=3,
279                         label=f'Vehicle {vehicle_id}')
280         ax1_twin = ax1.twinx() # 创建共享x轴的双y轴
281         ax1_twin.plot(group['time'], group['y'], color=plt.cm.
282             tab20(vehicle_id % 8), linewidth=3, linestyle='--',
283                         label=f'Vehicle {vehicle_id} Y')
284
285         # 遍历每个转弯点
286         for turn_time in turning_points:
287             # 分割数据
288             before_turn = group[group['time'] <= turn_time]
289             after_turn = group[group['time'] > turn_time]
290
291             # 绘制转弯点之前的轨迹
292             ax1.plot(before_turn['time'], before_turn['x'],
293             color=plt.cm.tab20(vehicle_id % 8), linewidth=3)
294             ax1_twin.plot(before_turn['time'], before_turn['y'],
295             color=plt.cm.tab20(vehicle_id % 8), linewidth=3,
296                         linestyle='--')
```

```
291     # 绘制转弯点之后的轨迹
292     if not after_turn.empty:
293         # 交换坐标
294         ax1.plot(after_turn['time'], after_turn['y'],
295                  color=plt.cm.tab20(vehicle_id % 8), linewidth=3)
296         ax1_twin.plot(after_turn['time'], after_turn['x'],
297                         color=plt.cm.tab20(vehicle_id % 8), linewidth=3,
298                         linestyle='--')
299
300
301     # 添加标记以表示转弯或掉头
302     turning_times = group[group['is_turning_or_u_turn']][
303     'time']
304
305     # 使用 merge 或 reindex 来同步时间戳
306     turning_times_df = pd.DataFrame({'time': turning_times})
307     merged_group = pd.merge(group, turning_times_df, on='time',
308                             how='inner')
309
310     # 现在 merged_group 包含了所有转弯时间
311     ax1.scatter(merged_group['time'], merged_group['x'],
312                 color='orange', marker='o', s=10,
313                 label='Turning or U-turn')
314     ax1_twin.scatter(merged_group['time'], merged_group['y'],
315                      color='orange', marker='o', s=10)
316
317     ax1.set_ylabel('X Position (m)')
318     ax1_twin.set_ylabel('Y Position (m)') # 设置右侧y轴标签
319     ax1.set_title("Vehicle Position and Traffic Light States
Over Time")
```

```
318 states = [traffic_light.get_state()]
319 while current_time <= data['time'].max():
320     traffic_light.update(1) # 更新状态机
321     state_times.append(current_time + 1)
322     states.append(traffic_light.get_state())
323     current_time += 1
324
325 # 将状态转换为颜色
326 color_map = {'green': 'g', 'red': 'r'}
327
328 # 创建一个列表来存储颜色
329 colors = [color_map[state] for state in states]
330
331 # 绘制红绿灯状态
332 for i in range(len(state_times) - 1):
333     ax2.plot([state_times[i], state_times[i + 1]], [colors[i], colors[i + 1]], color=colors[i], linewidth=3)
334
335 # 设置图表标签
336 ax2.set_xlabel("Time (seconds)")
337 ax2.set_ylabel("Light State")
338
339 # 添加图例
340 # 由于 plot 函数支持多个标签，我们可以直接使用 legend
341 custom_lines = [Line2D([0], [0], color='r', lw=4),
342                  Line2D([0], [0], color='g', lw=4)]
343 ax2.legend(custom_lines, ['Red Light', 'Green Light'], loc='upper left')
344
345 # 在 ax1 上绘制红绿灯状态变化的时间点
346 for i, state_time in enumerate(state_times):
347     # 使用更淡的颜色版本
348     light_colors = {'green': 'lightgreen', 'red': 'lightcoral'}
349     ax1.axvline(state_time, color=light_colors[states[i]],
```

```
    linestyle='--', linewidth=3, alpha=0.03)

350
351     plt.tight_layout()
352     plt.savefig(f"{path}\{dfname} total_time.png", dpi=300)
353     plt.show()

354
355     return avg_red_light_duration, avg_green_light_duration,
356     results

357

358 if __name__ == '__main__':
359     # 读取数据
360     data1 = pd.read_csv("F:\\Python数模\\暑期培训\\2407\\附件
361     \\附件1\\A1.csv", encoding="utf-8")
362     data2 = pd.read_csv("F:\\Python数模\\暑期培训\\2407\\附件
363     \\附件1\\A2.csv", encoding="utf-8")
364     data3 = pd.read_csv("F:\\Python数模\\暑期培训\\2407\\附件
365     \\附件1\\A3.csv", encoding="utf-8")
366     data4 = pd.read_csv("F:\\Python数模\\暑期培训\\2407\\附件
367     \\附件1\\A4.csv", encoding="utf-8")
368     data5 = pd.read_csv("F:\\Python数模\\暑期培训\\2407\\附件
369     \\附件1\\A5.csv", encoding="utf-8")

370     # Define the intersection area
371     intersection = {'xmin': 100, 'xmax': 200, 'ymin': 100,
372                     'ymax': 200}

373     dfname = ["A1", "A2", "A3", "A4", "A5"]
374     red_s, green_s = [], []
375     parent_folder = 'Q1'
376     for i in range(len(dfname)):
377         # 获取当前工作目录
378         current_directory = os.getcwd()
379         full_child_folder_path = os.path.join(
380             current_directory, parent_folder, dfname[i])
```

```

376     # 构造完整路径并创建父文件夹和子文件夹
377     os.makedirs(full_child_folder_path, exist_ok=True)
378     path = full_child_folder_path.replace('\\', '\\\\')
379     # 调用数据分析函数
380     avg_red, avg_green, _ = data_analysis(eval("data" +
381         str(i + 1)), path, dfname[i])
382     red_s.append(round(avg_red))
383     green_s.append(round(avg_green))
384
385     results = {"路口": dfname, "红灯时长(秒)": red_s, "绿灯时
长(秒)": green_s}
386     pd.DataFrame(results).to_csv("表1.csv", index=False,
encoding="GBK")

```

## Q2.py

```

1 import pandas as pd
2 import numpy as np
3 from sklearn.cluster import KMeans
4 from sklearn.metrics import silhouette_score
5 from sklearn.preprocessing import StandardScaler
6 from scipy.fftpack import fft, fftfreq
7 import matplotlib.pyplot as plt
8 from matplotlib.lines import Line2D
9 import os
10
11 # 数据预处理
12 def preprocess_data(data):
13     # 计算每辆车的速度
14     speed_diff = data.groupby('vehicle_id')[['x', 'y']].diff()
15     data['speed'] = speed_diff.apply(lambda x: np.sqrt(x['x']
** 2 + x['y'] ** 2), axis=1)
16
17     # 保留路口 [-100, 100] 范围内的数据
18     data = data[(data['x'] >= -150) & (data['x'] <= 150) & (
data['y'] >= -150) & (data['y'] <= 150)]

```

```
19
20     # 计算车辆的方向
21     direction_diff = data.groupby('vehicle_id')[['x', 'y']].diff()
22     data['direction'] = direction_diff.apply(lambda x: np.
arctan2(x['y'], x['x']), axis=1)
23
24     # 删除缺失值
25     data.dropna(inplace=True)
26
27     # 检测转弯
28     data['is_turning_or_u_turn'] = False
29     for _, group in data.groupby('vehicle_id'):
30         direction_diff = group['direction'].diff().abs()
31         data.loc[group.index, 'is_turning_or_u_turn'] = (
32             direction_diff > np.pi / 4) | (
33                 np.abs(direction_diff - np.pi) < 0.1)
34
35
36
37 # 停车事件检测
38 def detect_stops(data, threshold=0.5): # 单位: 米/秒
39     data['is_stopped'] = (data['speed'] < threshold).astype(
40         int)
41
42
43 # 发车事件检测
44 def detect_departures(data, threshold=0.5):
45     # 初始化发车事件标记列
46     data['is_departure'] = False
47
48     # 遍历每辆车的数据
49     for _, group in data.groupby('vehicle_id'):
```

```
50     # 计算速度差分来确定车辆是否从静止开始移动
51     speed_diff = group['speed'].diff().fillna(0)
52
53     # 标记速度从零增加到阈值以上的点为发车事件
54     departure_points = ((speed_diff > threshold) & (group[
55         'speed'].shift(1) <= threshold)).astype(int)
56     data.loc[group.index, 'is_departure'] =
57     departure_points
58
59     return data
60
61 # 聚类分析
62 def cluster_stops(data, path, dfname):
63     # 提取停车时间
64     stop_times = data[data['is_stopped'] == 1]['time'].values.
65     reshape(-1, 1)
66     # 数据标准化
67     scaler = StandardScaler()
68     stop_times_scaled = scaler.fit_transform(stop_times)
69     # 选择最优的聚类数量
70     silhouette_scores = []
71     for n in range(2, 40):
72         kmeans = KMeans(n_clusters=n)
73         kmeans.fit(stop_times)
74         silhouette_scores.append(silhouette_score(
75             stop_times_scaled, kmeans.labels_))
76
77     # 选择最优的聚类数量
78     optimal_n_clusters = silhouette_scores.index(max(
79         silhouette_scores)) + 2
80     print(f"Optimal number of clusters: {optimal_n_clusters}")
81
82     # 使用 K-Means 进行聚类
83     kmeans = KMeans(n_clusters=optimal_n_clusters)
84     kmeans.fit(stop_times)
```

```
80
81     # 获取聚类标签
82     labels = kmeans.labels_
83
84     # 确保标签数量与停车事件数量一致
85     assert len(labels) == len(stop_times), "Number of labels
86 must match number of stop times."
87
88     # 可视化聚类结果
89     plt.figure(figsize=(10, 6))
90     plt.scatter(stop_times, np.zeros_like(stop_times), c=
91     kmeans.labels_, cmap='viridis')
92     plt.title("Clustered Stop Times")
93     plt.xlabel("Time (seconds)")
94     plt.ylabel("Clusters")
95     plt.savefig(f"{path}\\{dfname} 聚类结果.png", dpi=300)
96     plt.show()
97
98
99 # 周期检测
100 def detect_cycles(labels, data, path, dfname):
101     # 创建一个 DataFrame 保存每个停车事件的标签
102     stop_events = pd.DataFrame({'time': data[data['is_stopped']
103 ] == 1]['time'], 'label': labels})
104     stop_events.to_csv(f"{dfname} 停车事件标签.csv", index=
105 False)
106
107     # 根据聚类标签分组
108     grouped = stop_events.groupby('label')['time'].apply(list)
109
110     # 计算每个群组的周期
111     periods = []
112     for group in grouped:
113         diffs = np.diff(group)
```

```
111 # 应用傅里叶变换
112 fft_result = fft(diffs)
113 frequencies = fftfreq(len(diffs), d=1) # 时间间隔为1
114 秒
115
116 # 计算功率谱密度
117 psd = np.abs(fft_result) ** 2
118
119 # 只保留正频率部分
120 positive_frequencies = frequencies[frequencies > 0]
121 positive_psd = psd[:len(positive_frequencies)]
122
123 # 找到峰值频率
124 peak_index = np.argmax(positive_psd)
125 dominant_frequency = positive_frequencies[peak_index]
126
127 # 估算周期
128 estimated_period = 1 / dominant_frequency
129 periods.append(estimated_period)
130
131 # 绘制功率谱密度图
132 plt.figure(figsize=(10, 6))
133 plt.plot(positive_frequencies, positive_psd)
134 plt.axvline(dominant_frequency, color='r', linestyle='--')
135 plt.title(f"{group[0]} Power Spectral Density")
136 plt.xlabel("Frequency (Hz)")
137 plt.ylabel("PSD")
138 plt.savefig(f"{path}\\{dfname} {group[0]}功率谱密度图.
png", dpi=300)
139 plt.show()
140
141 return periods
142
```

```
143 # 计算红绿灯持续时间
144 def estimate_light_durations(stop_times, periods):
145     # 初始化红绿灯时长列表
146     red_light_durations = []
147     green_light_durations = []
148
149     for period, stop_time_group in zip(periods, stop_times):
150         # 分割停车事件为连续的停车时间段
151         continuous_stop_intervals = []
152         start_time = None
153         for time in stop_time_group:
154             if start_time is None:
155                 start_time = time
156             elif time - start_time > 1: # 假设两个停车事件之间超过1秒不是连续的
157                 continuous_stop_intervals.append((start_time,
158                                                 time))
159             start_time = time
160             if start_time is not None:
161                 continuous_stop_intervals.append((start_time, time
162 ))
163
164         # 计算红灯时长
165         red_light_duration = sum(interval[1] - interval[0] for
166                                   interval in continuous_stop_intervals)
167
168         # 计算绿灯时长
169         green_light_duration = max(0, period -
170                                     red_light_duration) # 确保绿灯持续时间不小于0
171         green_light_durations.append(green_light_duration)
172         red_light_durations.append(period - green_light_duration
173 )
174
175     # 创建 DataFrame 存储结果
176     results = pd.DataFrame({
177         'cluster': list(range(len(periods))),
```

```
172     'period': periods,
173     'red_light_duration': red_light_durations,
174     'green_light_duration': green_light_durations
175   })
176
177   # 计算平均红绿灯时长
178   avg_red_light_duration = np.mean(red_light_durations)
179   avg_green_light_duration = np.mean(green_light_durations)
180
181   return avg_red_light_duration, avg_green_light_duration,
182   results
183
184 # 红绿灯状态机
185 class TrafficLight:
186     def __init__(self, red_duration, green_duration):
187         self.red_duration = red_duration
188         self.green_duration = green_duration
189         self.current_state = 'red'
190         self.current_time = 0
191
192     def update(self, dt):
193         self.current_time += dt
194         if self.current_state == 'red':
195             if self.current_time >= self.red_duration:
196                 self.current_state = 'green'
197                 self.current_time = 0
198         elif self.current_state == 'green':
199             if self.current_time >= self.green_duration:
200                 self.current_state = 'red'
201                 self.current_time = 0
202     return self.current_state
203
204     def get_state(self):
205         return self.current_state
```

```
206  
207  
208 def data_analysis(data, path, dfname):  
209     # 加载数据  
210     data = preprocess_data(data)  
211  
212     # 检测停车事件  
213     data = detect_stops(data)  
214  
215     # 计算发车事件  
216     data=detect_departures(data)  
217  
218     data.to_csv(f"{dfname}.csv", index=False)  
219     # 聚类分析  
220     labels = cluster_stops(data, path, dfname)  
221  
222     # 确保停车事件和对应的聚类标签数量一致  
223     assert len(data[data['is_stopped'] == 1]) == len(  
224         labels), "Number of stopped events does not match  
number of labels."  
225  
226     # 周期检测  
227     periods = detect_cycles(labels, data, path, dfname)  
228  
229     # 提取停车时间序列  
230     stop_times = data[data['is_stopped'] == 1]['time'].groupby  
(labels).apply(list).tolist()  
231  
232     # 计算红绿灯持续时间  
233     avg_red_light_duration, avg_green_light_duration, results  
= estimate_light_durations(stop_times, periods)  
234  
235     # 创建红绿灯状态机  
236     traffic_light = TrafficLight(avg_red_light_duration,  
avg_green_light_duration)
```

```
237  
238     # 输出结果  
239     print(f"Average red light duration: {  
240         avg_red_light_duration:.0f} seconds")  
241     print(f"Average green light duration: {  
242         avg_green_light_duration:.0f} seconds")  
243  
244     # 可视化不同聚类的红绿灯持续时间  
245     fig, axs = plt.subplots(1, 2, figsize=(14, 6))  
246  
247     # 绘制箱形图  
248     results.boxplot(column=['red_light_duration', '  
249         green_light_duration'], grid=False, ax=axs[0])  
250     axs[0].set_title("Boxplot of Red and Green Light Durations  
251         by Cluster")  
252     axs[0].set_ylabel("Duration (seconds)")  
253     axs[0].set_xlabel("Cluster")  
254  
255     # 绘制直方图  
256     axs[1].hist(results['red_light_duration'], bins=20, alpha  
257         =0.5, label='Red Light Duration')  
258     axs[1].hist(results['green_light_duration'], bins=20,  
259         alpha=0.5, label='Green Light Duration')  
260     axs[1].set_title("Histogram of Red and Green Light  
261         Durations by Cluster")  
262     axs[1].set_xlabel("Duration (seconds)")  
263     axs[1].set_ylabel("Frequency")  
264     axs[1].legend()  
265     plt.savefig(f"{path}\\"{dfname} 红绿灯持续时间箱形图和直方  
266         图.png", dpi=300)  
267  
268     # 绘制柱状图  
269     fig, ax = plt.subplots(figsize=(10, 6))  
270     results.plot(kind='bar', x='cluster', y=[  
271         'red_light_duration', 'green_light_duration'], ax=ax)
```

```

263     ax.set_title("Average Red and Green Light Durations by
264     Cluster")
265     ax.set_ylabel("Duration (seconds)")
266     ax.set_xlabel("Cluster")
267     plt.savefig(f"{path}\\"{dfname} 红绿灯持续时间柱状图.png",
268     dpi=300)
269
270
271     # 绘制原始时间序列
272     fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(20, 20),
273     sharex=True)
274
275
276     # 绘制原始时间序列上的时间与x和y的折线图
277     for vehicle_id, group in data.groupby('vehicle_id'):
278         # 获取转弯点的时间戳
279         turning_points = group[group['is_turning_or_u_turn']][
280         'time']
281
282         # 初始化绘图
283         ax1.plot(group['time'], group['x'], color=plt.cm.tab20
284         (vehicle_id % 8), linewidth=3,
285                 label=f'Vehicle {vehicle_id}')
286         ax1_twin = ax1.twinx() # 创建共享x轴的双y轴
287         ax1_twin.plot(group['time'], group['y'], color=plt.cm.
288         tab20(vehicle_id % 8), linewidth=3, linestyle='--',
289                 label=f'Vehicle {vehicle_id} Y')
290
291         # 遍历每个转弯点
292         for turn_time in turning_points:
293             # 分割数据
294             before_turn = group[group['time'] <= turn_time]
295             after_turn = group[group['time'] > turn_time]
296
297             # 绘制转弯点之前的轨迹
298             ax1.plot(before_turn['time'], before_turn['x'],
299             color=plt.cm.tab20(vehicle_id % 8), linewidth=3)

```

```

291         ax1_twin.plot(before_turn['time'], before_turn['y'],
292                         color=plt.cm.tab20(vehicle_id % 8), linewidth=3,
293                         linestyle='--')
294
295     # 绘制转弯点之后的轨迹
296     if not after_turn.empty:
297         # 交换坐标
298         ax1.plot(after_turn['time'], after_turn['y'],
299                     color=plt.cm.tab20(vehicle_id % 8), linewidth=3)
300         ax1_twin.plot(after_turn['time'], after_turn['x'],
301                     color=plt.cm.tab20(vehicle_id % 8), linewidth=3,
302                     linestyle='--')
303
304     # 添加标记以表示转弯
305     turning_times = group[group['is_turning_or_u_turn']][['time']]
306
307     # 使用 merge 或 reindex 来同步时间戳
308     turning_times_df = pd.DataFrame({'time': turning_times})
309
310     merged_group = pd.merge(group, turning_times_df, on='time',
311                             how='inner')
312
313     # 现在 merged_group 包含了所有转弯时间
314     ax1.scatter(merged_group['time'], merged_group['x'],
315                 color='orange', marker='o', s=10,
316                 label='Turning or U-turn')
317     ax1_twin.scatter(merged_group['time'], merged_group['y'],
318                      color='orange', marker='o', s=10)
319
320
321     ax1.set_ylabel('X Position (m)')
322     ax1_twin.set_ylabel('Y Position (m)') # 设置右侧y轴标签
323     ax1.set_title("Vehicle Position and Traffic Light States
Over Time")

```

```
317  
318     # 绘制红绿灯状态的变化折线图  
319     current_time = data['time'].min()  
320     state_times = [current_time]  
321     states = [traffic_light.get_state()]  
322     while current_time <= data['time'].max():  
323         traffic_light.update(1)  # 更新状态机  
324         state_times.append(current_time + 1)  
325         states.append(traffic_light.get_state())  
326         current_time += 1  
327  
328     # 将状态转换为颜色  
329     color_map = {'green': 'g', 'red': 'r'}  
330  
331     # 创建一个列表来存储颜色  
332     colors = [color_map[state] for state in states]  
333  
334     # 绘制红绿灯状态  
335     for i in range(len(state_times) - 1):  
336         ax2.plot([state_times[i], state_times[i + 1]], [colors[i],  
337                  colors[i + 1]], color=colors[i], linewidth=3)  
338  
339     # 设置图表标签  
340     ax2.set_xlabel("Time (seconds)")  
341     ax2.set_ylabel("Light State")  
342  
343     # 添加图例  
344     custom_lines = [Line2D([0], [0], color='r', lw=4),  
345                     Line2D([0], [0], color='g', lw=4)]  
346     ax2.legend(custom_lines, ['Red Light', 'Green Light'], loc  
347                 ='upper left')  
348  
349     # 在 ax1 上绘制红绿灯状态变化的时间点
```

```
lightcoral'}
```

350       ax1.axvline(state\_time, color=light\_colors[states[i]],  
351       linestyle='--', linewidth=3, alpha=0.03)

351

352       plt.tight\_layout()  
353       plt.savefig(f"{path}\\"{dfname} total\_time.png", dpi=300)  
354       plt.show()

355

356       return avg\_red\_light\_duration, avg\_green\_light\_duration,  
357       results

358

359 if \_\_name\_\_ == '\_\_main\_\_':  
360     # 读取数据  
361     data1 = pd.read\_csv("F:\\Python数模\\暑期培训\\2407\\附件  
362     \\附件2\\B1.csv", encoding="utf-8")  
362     data2 = pd.read\_csv("F:\\Python数模\\暑期培训\\2407\\附件  
363     \\附件2\\B2.csv", encoding="utf-8")  
363     data3 = pd.read\_csv("F:\\Python数模\\暑期培训\\2407\\附件  
364     \\附件2\\B3.csv", encoding="utf-8")  
364     data4 = pd.read\_csv("F:\\Python数模\\暑期培训\\2407\\附件  
365     \\附件2\\B4.csv", encoding="utf-8")  
365     data5 = pd.read\_csv("F:\\Python数模\\暑期培训\\2407\\附件  
366     \\附件2\\B5.csv", encoding="utf-8")

367     dfname = ["B1", "B2", "B3", "B4", "B5"]  
368     red\_s, green\_s = [], []  
369     parent\_folder = 'Q2'  
370     for i in range(len(dfname)):  
371         # 获取当前工作目录  
372         current\_directory = os.getcwd()  
373         full\_child\_folder\_path = os.path.join(  
374             current\_directory, parent\_folder, dfname[i])  
374         # 构造完整路径并创建父文件夹和子文件夹  
375         os.makedirs(full\_child\_folder\_path, exist\_ok=True)

```

376     path = full_child_folder_path.replace('\\', '\\\\')
377     # 调用数据分析函数
378     avg_red, avg_green, _ = data_analysis(eval("data" +
379     str(i + 1)), path, dfname[i])
380     red_s.append(round(avg_red))
381     green_s.append(round(avg_green))
382
383     results = {"路口": dfname, "红灯时长(秒)": red_s, "绿灯时
384     长(秒)": green_s}
385     pd.DataFrame(results).to_csv("表2.csv", index=False,
386     encoding="GBK")

```

### Q3.py

```

1 import pandas as pd
2 import numpy as np
3 from scipy.fftpack import fft, fftfreq
4 import matplotlib.pyplot as plt
5 import os
6
7 # 数据预处理
8 def preprocess_data(data):
9     # 计算每辆车的速度
10    speed_diff = data.groupby('vehicle_id')[['x', 'y']].diff()
11    data['speed'] = speed_diff.apply(lambda x: np.sqrt(x['x']
12    ** 2 + x['y'] ** 2), axis=1)
13
14    # 保留路口 [-100, 100] 范围内的数据
15    data = data[(data['x'] >= -100) & (data['x'] <= 100) &
16    (data['y'] >= -100) & (data['y'] <= 100)]
17
18    # 计算车辆的方向
19    direction_diff = data.groupby('vehicle_id')[['x', 'y']].
20    diff()
21    data['direction'] = direction_diff.apply(lambda x: np.
22    arctan2(x['y'], x['x']), axis=1)

```

```
19  
20     # 删除缺失值  
21     data.dropna(inplace=True)  
22  
23     # 检测转弯  
24     data['is_turning_or_u_turn'] = False  
25     for _, group in data.groupby('vehicle_id'):  
26         direction_diff = group['direction'].diff().abs()  
27         data.loc[group.index, 'is_turning_or_u_turn'] = (  
28             direction_diff > np.pi / 4) | (  
29                 np.abs(direction_diff - np.pi) < 0.1)  
30  
31  
32  
33 # 停车事件检测  
34 def detect_stops(data, threshold=0.5): # 单位: 米/秒  
35     data['is_stopped'] = (data['speed'] < threshold).astype(  
36     int)  
37  
38     return data  
39  
40 # 发车事件检测  
41 def detect_departures(data, threshold=0.5):  
42     # 初始化发车事件标记列  
43     data['is_departure'] = False  
44  
45     # 遍历每辆车的数据  
46     for _, group in data.groupby('vehicle_id'):  
47         # 计算速度差分来确定车辆是否从静止开始移动  
48         speed_diff = group['speed'].diff().fillna(0)  
49  
50         # 标记速度从零增加到阈值以上的点为发车事件  
51         departure_points = ((speed_diff > threshold) & (group[  
52             'speed'].shift(1) <= threshold)).astype(int)
```

```
51     data.loc[group.index, 'is_departure'] =  
52     departure_points  
53  
54     return data  
55  
56 #周期检测  
57 def detect_cycles(data):  
58     # 提取停车时间  
59     stop_times = data[data['is_stopped'] == 1]['time'].values  
60     # 检查是否有足够的停车点  
61     if len(stop_times) <= 3:  
62         # 如果停车点少于两个，则无法计算周期  
63         # 返回 None 或者一个默认的周期值  
64         return None # 或者 return 0 或者任何默认值  
65     # 计算相邻停车事件之间的时间差  
66     time_diffs = np.diff(stop_times)  
67  
68     # 应用傅里叶变换  
69     fft_result = fft(time_diffs)  
70     frequencies = fftfreq(len(time_diffs), d=1)  
71  
72     # 计算功率谱密度  
73     psd = np.abs(fft_result) ** 2  
74  
75     # 只保留正频率部分  
76     positive_frequencies = frequencies[frequencies > 0]  
77     positive_psd = psd[:len(positive_frequencies)]  
78  
79     # 找到峰值频率  
80     peak_index = np.argmax(positive_psd)  
81     dominant_frequency = positive_frequencies[peak_index]  
82  
83     # 估算周期  
84     estimated_period = 1 / dominant_frequency
```

```
85     return estimated_period
86
87
88 # 计算红绿灯持续时间
89 def estimate_light_durations(data, period, min_green_duration
90     =1):
91     # 提取停车时间
92     stop_times = data[data['is_stopped'] == 1]['time'].values
93
94     if len(stop_times) <= 2:
95         # 如果停车点少于两个，则无法计算周期
96         return None, None
97
98     # 分割停车事件为连续的停车时间段
99     continuous_stop_intervals = []
100    for i in range(1, len(stop_times)):
101        if len(continuous_stop_intervals)==0:
102            continuous_stop_intervals.append([(stop_times[i-1], stop_times[i])])
103        elif stop_times[i]-stop_times[i-1]<=3:
104            continuous_stop_intervals[-1].append((stop_times[i-1], stop_times[i]))
105        else:
106            continuous_stop_intervals.append([(stop_times[i-1], stop_times[i])])
107
108    # 计算红灯时长
109    red_light_duration = sum(interval[-1][1] - interval[0][0]
110        for interval in continuous_stop_intervals)/len(
111        continuous_stop_intervals)
112
113    # 计算绿灯时长，确保不小于最小绿灯时长
114    green_light_duration = max(min_green_duration, period -
115        red_light_duration)
116
117    return red_light_duration, green_light_duration
```

```
113  
114 # 滑动窗口处理  
115 def sliding_window_analysis(data, window_size, step_size):  
116     # 初始化结果列表  
117     results = []  
118  
119     # 初始化滑动窗口的起始位置  
120     start = 0  
121     end = start + window_size  
122  
123     # 循环遍历整个数据集  
124     while end <= len(data):  
125         # 提取当前窗口的数据  
126         window_data = data.iloc[start:end]  
127  
128         # 对窗口数据进行预处理  
129         window_data = preprocess_data(window_data)  
130  
131         # 检测停车事件  
132         window_data = detect_stops(window_data)  
133  
134         # 计算发车事件  
135         window_data = detect_departures(window_data)  
136  
137         # 周期检测  
138         periods = detect_cycles(window_data)  
139  
140         if periods is not None:  
141             # 计算红绿灯持续时间  
142             red_light_duration, green_light_duration =  
143             estimate_light_durations(window_data, periods)  
144  
145             if red_light_duration is not None and  
green_light_duration is not None:  
# 将结果添加到列表中
```

```
146     results.append({
147         'start_time': window_data['time'].min(),
148         'end_time': window_data['time'].max(),
149         'avg_red_light_duration':
150             red_light_duration,
151             'avg_green_light_duration':
152                 green_light_duration,
153             })
154
155     # 更新滑动窗口的位置
156     start += step_size
157     end = start + window_size
158
159     return results
160
161 # 检测周期变化
162 def detect_cycle_changes(results, threshold=60):
163     cycle_changes = []
164     pre, change=0,0
165
166     for i in range(1, len(results)):
167         prev_result = results[i]
168         next_result = results[i - 1]
169
170         # 计算周期变化
171         prev_period = prev_result['avg_red_light_duration'] +
172             prev_result['avg_green_light_duration']
173         next_period = next_result['avg_red_light_duration'] +
174             next_result['avg_green_light_duration']
175
176         period_change = abs(prev_period - next_period)
177
178         # 如果周期变化超过阈值，则记录周期变化
179         if period_change > threshold:
180             change=i
181             sum_red_light_duration, sum_green_light_duration
```

```

=0,0
177         for j in range(pre,change):
178             sum_red_light_duration+=results[j]['
179             avg_red_light_duration']
180             sum_green_light_duration+=results[j]['
181             avg_green_light_duration']
182             cycle_changes.append({
183                 'avg_red_light_duration':
184                     sum_red_light_duration/(change-pre),
185                 'avg_green_light_duration':
186                     sum_green_light_duration/(change-pre),
187                 'change_time': prev_result['end_time'],
188                 'prev_period': prev_period,
189                 'next_period': next_period,
190                 'period_change': period_change
191             })
192             pre = change
193             return cycle_changes
194
195 # 滑动窗口分析主函数
196 def main_analysis(data, window_size, step_size, path, dfname):
197     # 执行滑动窗口分析
198     results = sliding_window_analysis(data, window_size,
199     step_size)
200
201     # 检测周期变化
202     cycle_changes = detect_cycle_changes(results)
203
204     # 输出周期变化信息
205     for change in cycle_changes:
206         print(f"Red light duration: {change['
207             avg_red_light_duration']:.0f}s")
208         print(f"Green light duration: {change['
209             avg_green_light_duration']:.0f}s")

```

```

204     print(f"Cycle change detected at t={change['
205         change_time']:.0f}s:")
206         print(f"  Previous period: {change['prev_period']:.0f}
207             s")
208             print(f"  Next period: {change['next_period']:.0f}s")
209             print(f"  Period change: {change['period_change']:.0f}
210                 s\n")
211
212 # 绘制原始时间序列
213 fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(20, 14),
214 sharex=True)
215
216 # 绘制原始时间序列上的时间与x和y的折线图
217 for vehicle_id, group in data.groupby('vehicle_id'):
218     ax1.plot(group['time'], group['x'], color=plt.cm.tab20
219 (vehicle_id % 8), linewidth=3,
220             label=f'Vehicle {vehicle_id}')
221     ax1_twin = ax1.twinx() # 创建共享x轴的双y轴
222     ax1_twin.plot(group['time'], group['y'], color=plt.cm.
223 tab20(vehicle_id % 8), linewidth=3, linestyle='--',
224             label=f'Vehicle {vehicle_id} Y')
225
226     ax1.set_ylabel('X Position (m)')
227     ax1_twin.set_ylabel('Y Position (m)') # 设置右侧y轴标签
228     ax1.set_title("Vehicle Position and Traffic Light States
Over Time")
229
230     for result in results:
231         ax2.axvspan(result['start_time'], result['end_time'],
232 color='blue', alpha=0.2)
233         ax2.text(result['start_time'], 1, f"Period: {result['
234 avg_red_light_duration']} + {result['avg_green_light_duration']}
235 [:.0f}s",
236             bbox=dict(facecolor='red', alpha=0.5))
237
238

```

```

229     for change in cycle_changes:
230         ax2.axvline(change['change_time'], color='red',
231         linestyle='--')
232         ax2.text(change['change_time'], 0.9, f"Change: {change
233             ['period_change']:.0f}s",
234                 bbox=dict(facecolor='green', alpha=0.5))
235
236         ax2.set_xlabel("Time (seconds)")
237         ax2.set_ylabel("Period (seconds)")
238         ax2.set_ylim(0, 2 * max([result['avg_red_light_duration']
239             + result['avg_green_light_duration'] for result in results]))
240
241         ax2.set_title("Signal Light Cycle Changes Over Time")
242         plt.savefig(f"{path}\{dfname} 信号灯周期变化.png", dpi
243 =500)
244         plt.show()
245
246     return cycle_changes
247
248 # 主函数
249 if __name__ == '__main__':
250     # 读取数据
251     data1 = pd.read_csv("F:\\Python数模\\暑期培训\\2407\\附件
252         \\附件3\\C1.csv", encoding="utf-8")
253     data2 = pd.read_csv("F:\\Python数模\\暑期培训\\2407\\附件
254         \\附件3\\C2.csv", encoding="utf-8")
255     data3 = pd.read_csv("F:\\Python数模\\暑期培训\\2407\\附件
256         \\附件3\\C3.csv", encoding="utf-8")
257     data4 = pd.read_csv("F:\\Python数模\\暑期培训\\2407\\附件
258         \\附件3\\C4.csv", encoding="utf-8")
259     data5 = pd.read_csv("F:\\Python数模\\暑期培训\\2407\\附件
260         \\附件3\\C5.csv", encoding="utf-8")
261     data6 = pd.read_csv("F:\\Python数模\\暑期培训\\2407\\附件
262         \\附件3\\C6.csv", encoding="utf-8")

```

```

253     dfname = ["C1", "C2", "C3", "C4", "C5", "C6"]
254     results = {"路口":[], "C1":[], "C2":[], "C3":[], "C4":[], "C5":[], "C6":[]}
255     parent_folder = 'Q3'
256     for i in range(len(dfname)):
257         # 获取当前工作目录
258         current_directory = os.getcwd()
259         full_child_folder_path = os.path.join(
260             current_directory, parent_folder, dfname[i])
261         # 构造完整路径并创建父文件夹和子文件夹
262         os.makedirs(full_child_folder_path, exist_ok=True)
263         path = full_child_folder_path.replace('\\', '\\\\')
264         # 调用数据分析函数
265         cycle_change = main_analysis(eval("data" + str(i + 1)))
266         , window_size=200, step_size=100, path=path, dfname=f"{dfname[i]}traffic_data")
267         for j in range(len(cycle_change)):
268             if f"周期 {j+1} 红灯时长 (秒)" not in results["路口"] and f"周期 {j+1} 绿灯时长 (秒)" not in results["路口"]:
269                 results["路口"].extend([f"周期 {j+1} 红灯时长 (秒)", f"周期 {j+1} 绿灯时长 (秒)", "周期切换时刻"])
270             results[dfname[i]].extend([round(cycle_change[j]["avg_red_light_duration"]), round(cycle_change[j]["avg_green_light_duration"]), round(cycle_change[j]["change_time"])])
271
272             maxlength=len(results["路口"])
273             for i in range(len(results[dfname[i]]),maxlength):
274                 results[dfname[i]].append(np.nan)
275
276             pd.DataFrame(results).to_csv("表3.csv", index=False,
277                                         encoding="GBK")

```

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.cluster import KMeans
4 from sklearn.metrics import silhouette_score
5 from scipy.fftpack import fft, fftfreq
6 import matplotlib.pyplot as plt
7 from matplotlib.lines import Line2D
8 from sklearn.preprocessing import StandardScaler
9 import os
10
11 # 数据预处理
12 def preprocess_data(data):
13     # 计算每辆车的速度
14     data = data[data['vehicle_id'] <= 5418]
15     speed_diff = data.groupby('vehicle_id')[['x', 'y']].diff()
16     data['speed'] = speed_diff.apply(lambda x: np.sqrt(x['x'] ** 2 + x['y'] ** 2), axis=1)
17
18     # 保留路口 [-200, 200] 范围内的数据
19     data = data[(data['x'] >= -100) & (data['x'] <= 100) & (
20         data['y'] >= -100) & (data['y'] <= 100)]
21
22     # 计算车辆的方向
23     direction_diff = data.groupby('vehicle_id')[['x', 'y']].diff()
24     data['direction'] = direction_diff.apply(lambda x: np.
25         arctan2(x['y'], x['x']), axis=1)
26
27     # 计算方向变化
28     data['direction_diff'] = data.groupby('vehicle_id')[[
29         'direction']].diff()

# 删掉缺失值
data.dropna(inplace=True)
```

```
30
31     # 设置速度阈值
32     speed_threshold = 5  # 单位取决于原始数据中的速度单位
33
34     # 检测转弯并分类为左转、直行或右转
35     data['turn_type'] = 'straight'  # 初始化为直行
36     right_turn_vehicles = set()  # 用于记录右转车辆的
37         vehicle_id
38     left_turn_vehicles = set()  # 用于记录左转车辆的
39         vehicle_id
40     for _, group in data.groupby('vehicle_id'):
41         # 确定是否为转弯
42         data.loc[group.index, 'turn_type'] = np.where((group['
43             direction_diff'].abs() > np.pi / 4) &
44             (group['
45             speed'] < speed_threshold),
46             np.where
47             (group['direction_diff'] > 0, 'left_turn', 'right_turn'),
48             data.loc
49             [group.index, 'turn_type'])
50
51         # 检查是否有右转
52         if 'right_turn' in data.loc[group.index, 'turn_type'].unique():
53             right_turn_vehicles.add(group['vehicle_id'].iloc
54             [0])
55
56         # 检查是否有左转
57         if 'left_turn' in data.loc[group.index, 'turn_type'].unique():
58             left_turn_vehicles.add(group['vehicle_id'].iloc
59             [0])
60
61     # 删除右转车辆的所有数据
62     data = data[data['turn_type'] != 'right_turn']
```

```
55
56     # 标记左转车辆的所有数据为左转
57     for vehicle_id in left_turn_vehicles:
58         data.loc[data['vehicle_id'] == vehicle_id, 'turn_type']
59             ] = 'left_turn'
60
61     # 添加转弯方向字段
62     data['turn_direction'] = '' # 初始化为空字符串
63     for _, group in data.groupby('vehicle_id'):
64         # 确定转弯开始的索引
65         start_index = group[group['turn_type'] == 'left_turn']
66             ].index.min()
67
68         # 如果找到了转弯开始的索引
69         if (not pd.isna(start_index)):
70             # 找到第一个转弯点
71             first_turn_index = group[group['turn_type'] ==
72                 'left_turn'].index.min()
73
74             # 确定转弯结束的索引
75             # 从转弯开始的下一个位置开始查找，直到方向变化不再
76             # 满足转弯条件
77             # 确保两个布尔索引应用在同一长度的数据上
78             end_index_mask = (group.index > first_turn_index)
79             & (group['direction_diff'].abs() <= np.pi / 4)
80             end_index = group.index[end_index_mask].min()
81
82             # 标记转弯点之后的数据
83             data.loc[group.index[
84                 (group.index >= first_turn_index) & (group.
85                 index <= end_index)], 'turn_direction'] = 'left_turn'
86
87             # 确定转弯前后的方向
88             # 假设车辆在转弯前后的方向变化不超过180度
89             before_turn_direction = group.loc[first_turn_index
```

```
    , 'direction']  
84        after_turn_direction = group.loc[end_index, 'direction']  
85  
86        # 计算转弯前后的方向差  
87        direction_diff = (after_turn_direction -  
before_turn_direction + np.pi) % (2 * np.pi) - np.pi  
88  
89        # 确定转弯前后的方向  
90        if direction_diff > 0:  
91            # 左转  
92            from_direction = np.where(  
before_turn_direction > 0, 'north', 'east')  
93            to_direction = np.where(after_turn_direction >  
0, 'west', 'north')  
94        else:  
95            # 右转  
96            from_direction = np.where(  
before_turn_direction > 0, 'west', 'south')  
97            to_direction = np.where(after_turn_direction >  
0, 'north', 'east')  
98  
99        # 更新数据  
100       data.loc[group.index, 'turn_direction'] = 'left_turn'  
101       data.loc[group.index, 'from_direction'] =  
from_direction  
102       data.loc[group.index, 'to_direction'] =  
to_direction  
103       continue # 退出循环，确保每个车辆 ID 只有一次转弯  
被标记  
104  
105       # 确定直行车辆的前后方向  
106       elif (group['turn_type'] == 'straight').all():  
107           # 确定直行前的方向
```

```

108         data.loc[group.index, 'turn_direction'] = 'straight'
109         initial_direction = group['direction'].iloc[0]
110         data.loc[group.index, 'from_direction'] = np.where(
111             (initial_direction > 0, 'north', 'east'))
112
113         # 确定直行后的方向
114         final_direction = group['direction'].iloc[-1]
115         data.loc[group.index, 'to_direction'] = np.where(
116             (final_direction > 0, 'south', 'west'))
117
118         # 输出中间结果
119         data.to_csv("D_change.csv", index=False)
120
121     return data
122
123 # 检测停车
124 def detect_stops(data, threshold=0.5):
125     data['is_stopped'] = (data['speed'] < threshold).astype(
126         int)
127     return data
128
129 # 检测起步
130 def detect_departures(data, threshold=0.5):
131     data['is_departure'] = False
132     for _, group in data.groupby('vehicle_id'):
133         speed_diff = group['speed'].diff().fillna(0)
134         departure_points = ((speed_diff > threshold) & (group[
135             'speed'].shift(1) <= threshold)).astype(int)
136         data.loc[group.index, 'is_departure'] =
137             departure_points
138
139     return data
140
141 # K-Means聚类
142 def cluster_stops(data, path, dfname):

```

```
137 # 提取停车时间
138 stop_times = data[data['is_stopped'] == 1]['time'].values.
reshape(-1, 1)
139
140 # 数据标准化
141 scaler = StandardScaler()
142 stop_times_scaled = scaler.fit_transform(stop_times)
143
144 # 选择最优的聚类数量
145 max_clusters = min(200, len(stop_times_scaled) - 1) # 限制最大聚类数量为10或样本数量-1
146 best_silhouette = -1
147 best_k = 2
148 for k in range(2, max_clusters + 1):
149     print(k)
150     # KMeans 聚类
151     kmeans = KMeans(n_clusters=k, random_state=42).fit(
stop_times_scaled)
152     stop_labels = kmeans.labels_
153
154     # 计算轮廓系数
155     silhouette_avg = silhouette_score(stop_times_scaled,
stop_labels)
156
157     # 更新最佳轮廓系数和对应的聚类数量
158     if silhouette_avg > best_silhouette:
159         best_silhouette = silhouette_avg
160         best_k = k
161
162     # 使用最佳聚类数量进行聚类
163     kmeans = KMeans(n_clusters=best_k, random_state=42).fit(
stop_times_scaled)
164
165     # 获取聚类标签
166     labels = kmeans.labels_
```

```
167
168     # 确保标签数量与停车事件数量一致
169     assert len(labels) == len(stop_times_scaled), "Number of
170     labels must match number of stop times."
171     """
172     # 可视化聚类结果
173     plt.figure(figsize=(10, 6))
174     plt.scatter(stop_times, np.zeros_like(stop_times), c=
175     kmeans.labels_, cmap='viridis')
176     plt.title("Clustered Stop Times")
177     plt.xlabel("Time (seconds)")
178     plt.ylabel("Clusters")
179     plt.savefig(f"{path}\\{dfname} 聚类结果.png", dpi=300)
180     plt.show()"""
181
182     return labels
183
184 # 傅里叶变换
185 def detect_cycles(labels, data, path, dfname):
186     # 创建一个 DataFrame 保存每个停车事件的标签
187     stop_events = pd.DataFrame({'time': data[data['is_stopped']
188 ] == 1]['time'], 'label': labels})
189
190     # 根据聚类标签分组
191     grouped = stop_events.groupby('label')['time'].apply(list)
192
193     # 计算每个群组的周期
194     periods = []
195     for label, times in grouped.items():
196         if len(times) < 2:
197             print(f"Warning: Label {label} has less than two
198 time points, skipping.")
199             continue
200
201     # 计算时间差分
```

```
198     diffs = np.diff(times)
199
200     # 确保 diffs 不为空
201     if len(diffs) == 0:
202         print(f"Warning: Empty diffs for Label {label},"
203             "skipping.")
203         continue
204
205     # 应用傅里叶变换
206     fft_result = fft(diffs)
207     frequencies = fftfreq(len(diffs), d=1) # 时间间隔为1
208     秒
209
210     # 计算功率谱密度
211     psd = np.abs(fft_result) ** 2
212
213     # 只保留正频率部分
214     positive_frequencies = frequencies[frequencies > 0]
215     positive_psd = psd[:len(positive_frequencies)]
216
217     # 确保 positive_psd 不为空
218     if len(positive_psd) == 0:
219         print(f"Warning: Empty PSD for Label {label},"
220             "skipping.")
220         continue
221
222     # 找到峰值频率
223     peak_index = np.argmax(positive_psd)
224     dominant_frequency = positive_frequencies[peak_index]
225
226     # 估算周期
227     estimated_period = 1 / dominant_frequency
228     periods.append(estimated_period)
229     """
229
229     # 绘制功率谱密度图
```

```

230     plt.figure(figsize=(10, 6))
231     plt.plot(positive_frequencies, positive_psd)
232     plt.axvline(dominant_frequency, color='r', linestyle
233 = '--')
234     plt.title(f"{label} Power Spectral Density")
235     plt.xlabel("Frequency (Hz)")
236     plt.ylabel("PSD")
237     plt.savefig(f"{path}\{dfname} {label}功率谱密度图.png",
238     dpi=300)
239     plt.show()"""
240
241 # 估计红绿灯时长
242 def estimate_light_durations(stop_times, periods,
243 min_green_duration=1):
244     # 初始化红绿灯时长列表
245     red_light_durations = []
246     green_light_durations = []
247
248     for period, stop_time_group in zip(periods, stop_times):
249         # 分割停车事件为连续的停车时间段
250         if len(stop_time_group) <= 2:
251             continue
252         continuous_stop_intervals = []
253         for i in range(1, len(stop_time_group)):
254             if len(continuous_stop_intervals) == 0:
255                 continuous_stop_intervals.append([(stop_time_group[i - 1], stop_time_group[i])])
256             elif stop_time_group[i] - stop_time_group[i - 1]
257             <= 2:
258                 continuous_stop_intervals[-1].append((stop_time_group[i - 1], stop_time_group[i]))
259             else:
260                 continuous_stop_intervals.append([(stop_time_group[i - 1], stop_time_group[i])])

```

```
stop_time_group[i - 1], stop_time_group[i]))  
259  
260     # 计算红灯时长  
261     red_light_duration = sum(interval[-1][1] - interval  
262     [0][0] for interval in continuous_stop_intervals) / len(  
263     continuous_stop_intervals)  
264  
265     # 计算绿灯时长，确保不小于最小绿灯时长  
266     green_light_duration = max(min_green_duration, period  
267     - red_light_duration)  
268     red_light_durations.append(red_light_duration)  
269     green_light_durations.append(green_light_duration)  
270  
271     # 计算平均红绿灯时长  
272     avg_red_light_duration = np.mean(red_light_durations)  
273     avg_green_light_duration = np.mean(green_light_durations)  
274  
275 # 红绿灯状态机  
276 class TrafficLight:  
277     def __init__(self, red_duration, green_duration):  
278         self.red_duration = red_duration  
279         self.green_duration = green_duration  
280         self.current_state = 'red'  
281         self.current_time = 0  
282  
283     def update(self, dt):  
284         self.current_time += dt  
285         if self.current_state == 'red':  
286             if self.current_time >= self.red_duration:  
287                 self.current_state = 'green'  
288                 self.current_time = 0  
289         elif self.current_state == 'green':  
290             if self.current_time >= self.green_duration:  
291                 self.current_state = 'red'  
292                 self.current_time = 0
```

```
290         self.current_state = 'red'
291         self.current_time = 0
292     return self.current_state
293
294     def get_state(self):
295         return self.current_state
296
297 # 主分析函数
298 def data_analysis(data, path, dfname):
299     # 数据预处理
300     data = preprocess_data(data)
301
302     # 检测停车事件
303     data = detect_stops(data)
304
305     # 计算发车事件
306     data = detect_departures(data)
307
308     # 根据 from_direction, turn_direction, to_direction 分组
309     grouped_data = data.groupby(['from_direction', 'turn_direction',
310                                  'to_direction'])
311
312     avg_red_light_durations, avg_green_light_durations,
313     group_name = [], [], []
314     # 对每个分组分别进行停车点聚类
315     for name, group in grouped_data:
316         group_name.append(name)
317         # 聚类分析
318         labels = cluster_stops(group, path, f"{dfname}_{name}")
319     )
320
321     # 确保停车事件和对应的聚类标签数量一致
322     assert len(group[group['is_stopped']] == 1]) == len(
323     labels), "Number of stopped events does not match number of
324     labels."
```

```
320
321     # 周期检测
322     periods = detect_cycles(labels, group, path, dfname)
323
324     # 提取停车时间序列
325     stop_times = group[group['is_stopped'] == 1]['time'].groupby(labels).apply(list).tolist()
326
327     # 计算红绿灯持续时间
328     avg_red_light_duration, avg_green_light_duration=estimate_light_durations(stop_times, periods)
329     avg_red_light_durations.append(avg_red_light_duration)
330     avg_green_light_durations.append(
331         avg_green_light_duration)
332
333     # 创建红绿灯状态机
334     traffic_light = TrafficLight(avg_red_light_duration,
335     avg_green_light_duration)
336
337     # 输出结果
338     print(f"Average red light duration: {avg_red_light_duration:.0f} seconds")
339     print(f"Average green light duration: {avg_green_light_duration:.0f} seconds")
340     """
341
342     # 绘制原始时间序列
343     fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(20, 20),
344     sharex=True)
345
346     # 绘制原始时间序列上的时间与x和y的折线图
347     for vehicle_id, subgroup in group.groupby('vehicle_id'):
348
349         # 初始化绘图
350         ax1.plot(subgroup['time'], subgroup['x'], color=
```

```

plt.cm.tab20(vehicle_id % 8), linewidth=3,
            label=f'Vehicle {vehicle_id}')
348     ax1_twin = ax1.twinx() # 创建共享x轴的双y轴
349     ax1_twin.plot(subgroup['time'], subgroup['y'],
350 color=plt.cm.tab20(vehicle_id % 8), linewidth=3, linestyle
351 = '--',
352             label=f'Vehicle {vehicle_id} Y')
353
354     ax1.set_ylabel('X Position (m)')
355     ax1_twin.set_ylabel('Y Position (m)') # 设置右侧y轴标
356     ax1.set_title("Vehicle Position and Traffic Light
357 States Over Time")
358
359     # 绘制红绿灯状态的变化折线图
360     current_time = group['time'].min()
361     state_times = [current_time]
362     states = [traffic_light.get_state()]
363     while current_time <= group['time'].max():
364         traffic_light.update(1) # 更新状态机
365         state_times.append(current_time + 1)
366         states.append(traffic_light.get_state())
367         current_time += 1
368
369     # 将状态转换为颜色
370     color_map = {'green': 'g', 'red': 'r'}
371
372     # 绘制红绿灯状态
373     for i in range(len(state_times) - 1):
374         ax2.plot([state_times[i], state_times[i + 1]], [
375 color_map[states[i]], color_map[states[i + 1]]], color=
376 color_map[states[i]], linewidth=3)
377
378     # 设置图表标签
379     ax2.set_xlabel("Time (seconds)")

```

```
375     ax2.set_ylabel("Light State")
376
377     # 添加图例
378     custom_lines = [Line2D([0], [0], color='r', lw=4),
379                      Line2D([0], [0], color='g', lw=4)]
380     ax2.legend(custom_lines, ['Red Light', 'Green Light'],
381                loc='upper left')
382
383     # 在 ax1 上绘制红绿灯状态变化的时间点
384     for i, state_time in enumerate(state_times):
385         light_colors = {'green': 'lightgreen', 'red': 'lightcoral'}
386         ax1.axvline(state_time, color=light_colors[states[i]],
387                     linestyle='-', linewidth=3, alpha=0.03)
388
389     plt.tight_layout()
390     plt.savefig(f"{path}\{name}_total_time.png", dpi=300)
391     plt.show()"""
392
393     return avg_red_light_durations, avg_green_light_durations,
394     group_name
395
396 if __name__ == '__main__':
397     # 读取数据
398     data1 = pd.read_csv("F:\\Python数模\\暑期培训\\2407\\附件
399     \\附件4\\D.csv", encoding="utf-8")
400
401     dfname = "D"
402     parent_folder = 'Q4'
403     current_directory = os.getcwd()
404     full_child_folder_path = os.path.join(current_directory,
405     parent_folder, dfname)
406
407     # 构造完整路径并创建父文件夹和子文件夹
408     if not os.path.exists(full_child_folder_path):
```

```
404     os.makedirs(full_child_folder_path)
405
406     # 处理数据
407     Avg_red_light_duration, Avg_green_light_duration,
408     group_name = data_analysis(data1, full_child_folder_path,
409     dfname)
410
411     # 输出结果
412     print("Average Red Light Durations:",
413         Avg_red_light_duration)
414     print("Average Green Light Durations:",
415         Avg_green_light_duration)
416     print("Group Names:", group_name)
417
418     # 保存结果到CSV文件
419     pd.DataFrame({ "Directions": group_name, "
420         Avg_red_light_duration": Avg_red_light_duration, "
421         Avg_green_light_duration": Avg_green_light_duration}).to_csv(
422             os.path.join(full_child_folder_path, f"{dfname}_results.csv"
423             ), index=False, encoding="utf-8")
424
425     # 输出所有方向的平均红灯和绿灯时长
426     all_directions_red = np.mean(Avg_red_light_duration)
427     all_directions_green = np.mean(Avg_green_light_duration)
428     print(f"All Directions Average Red Light Duration: {"
429         all_directions_red:.0f} seconds")
430     print(f"All Directions Average Green Light Duration: {"
431         all_directions_green:.0f} seconds")
```