基于行车轨迹数据的交通信号灯周期估算方法研究

摘要

在现代城市中，交通信号灯系统直接影响着交通流的效率和城市道路的拥堵情况。然而不少城市的交通信号灯并未与网络相连，其周期数据难以直接获取。本文主要研究不同条件下利用车辆行车轨迹数据来估计交通信号灯周期的方法，分析大量实时行车轨迹，构建数学模型来推断各个路口的红绿灯周期，并探讨如何在信号灯周期可能发生变化时快速准确地检测并适应周期变化。该方法的应用不仅能优化驾驶者的导航体验，还能为城市交通管理部门提供有力的决策支持，以实现更高效的交通流控制。

针对问题一，根据附件 1 提供的较为完整的车辆轨迹数据，抽取部分车辆绘制路口 A1 到 A5 的轨迹示意图，在此基础上建立定周期信号分析模型，用差分法计算所有车辆各时刻的速度，运用统计学方法归纳队首车辆位置关键特征，对队首车辆的启动时刻进行深入研究，确定五个路口信号总周期为 105s 或 88s，然后进一步运用交通流理论推断出每个周期中绿灯持续时间的最优估计值，将结果按照格式填入表 4。

针对问题二，分别讨论三个重要因素对定周期信号分析模型精度的影响，首先使用卡尔曼滤波法递归估计动态系统的状态，通过图像分析技术确定定位误差对模型精度的微小影响，验证了模型在一定误差范围内的健壮性；通过蒙特卡洛模拟不同车辆数据比例的多轮实验并定义精度计算方法。结果显示，车辆比例从 50% 增加到 100% 时，模型精度增加约 2% ～ 3%，表明模型精度并非严重依赖于车辆数据的完整性；采用周期性密度法设定合理阈值后发现，车流量的极端情况显著影响周期计算的精度。借助上述启示微调模型，利用附件 2 中不完整的车辆数据，计算了路口 B1 到 B5 的信号灯周期，将结果记录于表 8。

针对问题三，面对信号灯周期可能变化的情况，初步研究发现路口 C3 的总周期在第 1025-1445 秒发生了突变，然后建立带滑动窗口的贝叶斯变点检测模型，用向前填充和向后填充的平均值方法来弥补时间序列中的缺陷，运用变点前后特征点的欧几里得距离筛选出变化最显著的点填入表 9。就本模型而言，识别出周期变化需要至少一个时间窗口的长度以及稳定的数据采集频率。

针对问题四，深入挖掘单个路口在一个信号周期内各方向的交互制约关系，将附件 4 提供的数据全向分解为 D1 至 D12 共十二个方向，延续已有模型的思路构建整体性全路径周期模型，找到第 2414s 是一个周期变化的分界点，分界点前后共同遵循长度为 142 秒的总周期。应用交通流知识对周期内部的序列深入分析，通过定量方法确认左转和右转车辆可以同时通过路口，分析结果成功确定了所有方向上的信号灯周期，展示在表 12 中。

关键词：蒙特卡洛模拟　贝叶斯变点检测　卡尔曼滤波　滑动时间窗口　信号灯周期

目录

一、 问题重述

1.1 问题背景

某电子地图服务商希望获取城市路网中所有交通信号灯的红绿周期，以便为司机提供更好的导航服务，提高交通效率。而由于无法直接从交通管理部门获取所有信号灯的数据或在所有路口安排人工读取信号灯周期信息，该公司计划通过分析大量客户的行车轨迹数据估计交通信号灯的周期。同时为对问题进行一定简化，假设信号灯只有红绿两种状态。

1.2 问题要求

(1) 在信号灯周期固定不变的情况下，建立数学模型，要求其能通过分析一段时间多个不相关路口所收集的所有车辆的轨迹数据，求出路口对应方向的信号灯周期。

(2) 讨论样本车辆比例、车流量、定位误差等因素对上述模型估计精度的影响。根据已知数据求出某些路口相应方向的信号灯变化周期。

(3) 在信号灯周期发生变化的情况下，探讨如何尽快检测出这种变化，以及变化后的新周期。尝试求出周期切换的时刻，以及新旧周期参数，明确周期变化所需的时间和条件。

(4) 综合上述考虑，通过分析给定时间内某路口所有方向样本车辆的轨迹数据，识别出该路口信号灯的周期。

1.3 问题分析

(1) 问题一分析:

针对问题一，为推算出各路口对应方向上的信号灯周期，首先利用附件 1 所提供的车辆轨迹数据，抽取部分车辆绘制路口轨迹示意图，在图像基础上尝试建立定周期信号分析模型，依次进行计算所有车辆各时刻的速度，运用统计学方法归纳队首车辆位置关键特征，对队首车辆的启动时刻进行深入研究，同时建立红灯、绿灯对应的序列，进而确定周期时长。

(2) 问题二分析:

针对问题二，我们需要分别讨论三个重要因素对定周期信号分析模型精度的影响，我们可以采用卡尔曼滤波法递归估计动态系统的状态，通过蒙特卡洛模拟进行不同车辆数据比例的多轮实验并定义精度计算方法，判断车流量的极端情况对周期计算精度的影响。借助发现的结论微调模型，利用附件 2 中不完整的车辆数据，计算路口 B1 到 B5 的信号灯周期。

(3) 问题三分析:

针对问题三，首先观察在特定时间段信号灯周期是否出现显著变化，然后考虑将滑动窗口与变点检测模型结合建模，同时弥补时间序列中的缺陷，在变点检测后，为了更精确地识别和定位周期的变化点，采用距离来度量变点前后特征点的差异，筛选出变化最显著的点，进而分析得出问题三的答案。

(4) 问题四分析:

针对问题四，研究重点是探索单个路口在一个信号周期内各方向的交互制约关系。首先将附件四提供的数据全向分解为多个方向。然后继续使用已有的模型框架，构建整体性全路径模型，找到周期变化的分界点及其前后共同遵循的总周期长度。接着依据交通流的原理和知识对周期内部的序列进行深入分析，对细节的模糊处深入讨论，得到问题四的答案。

二、 模型假设

1. 题目所给车辆行驶轨迹数据的来源和质量真实可靠，且符合实际情况；

2. 假设所有司机都遵守交通规则，即车辆在红灯时会停止，在绿灯时会通过，司机的行为不会因个人习惯或意图而有显著不同；

3. 不考虑交通事故，道路维修，紧急车辆等因素的影响；

4. 司机的反应时间极短，没有由于个人因素产生不必要的停顿；

5. 假设每个路口的信号灯是独立控制的，与其他路口的信号灯无协同或联动效应。

三、 符号说明

表 **1** 符号说明表

符号	符号说明
x_e	前一时刻的状态估计
p_p	当前时刻的误差协方差预测
S	残差协方差
H	观测矩阵
I	单位矩阵
R	观测噪声协方差矩阵
m	变点的数量
β	惩罚参数

四、 问题一模型建立与求解

4.1 数据预处理和模型准备

4.1.1 缺失值检查

由于行车轨迹的缺失以及不准确会影响对红绿灯周期的判断，因此确定是否存在缺失值是分析中重要的环节。基于附件一所提供的五张表格，对其进行数据筛查后，确认本题不存在缺失值。

4.1.2 轨迹探索

行车轨迹可以大致反应红绿灯的位置信息有利于更好地分析其周期，源于此我们对每个路口的车辆都进行了轨迹探索，其中每个路口选取五辆车同时选取每辆车的部分数据绘制散点图，结果如下：

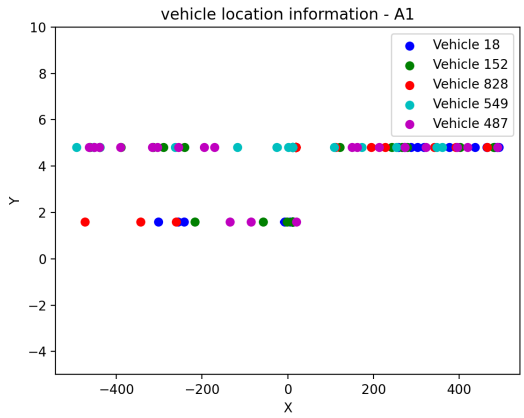

图 1　A1 轨迹图

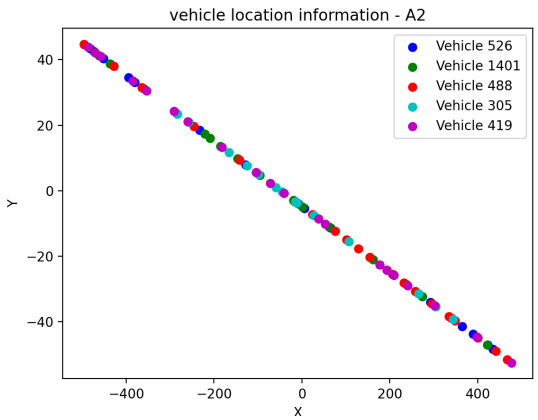

图 2　A2 轨迹图

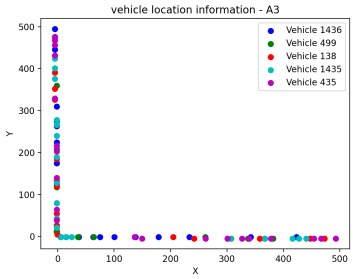

图 3　A3 轨迹图

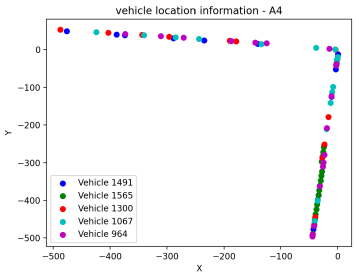

图 4　A4 轨迹图

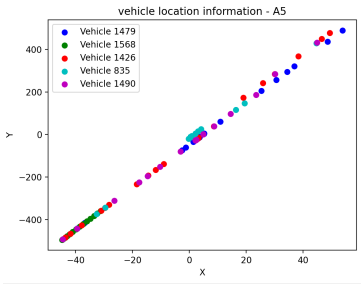

图 5　A5 轨迹图

对上图进行分析, 可以发现在 A1、A2、A5 的轨迹图中，车辆大致沿直线行驶，部分车辆在靠近路口的地方发生换道，三幅图的红绿灯分别位于 $(0,6),(-11,-3),(5,-10)$ 附近；A3、A4 轨迹图中，车辆在路口处左转，其中 A3 中红绿灯位置在 $(-10,-10)$ 附近，A4 中，红绿灯位置在 $(0,0)$ 附近。

4.2 定周期信号分析模型的建立

4.2.1 模型思想

(1) 速度计算：已知所给的车辆轨迹数据，根据连续时间点的车辆位置信息采用差分法计算出各时刻车辆的瞬时速度。

(2) 队首位置确定：运用统计学方法，分析出每个路口车辆分布最为集中的位置坐标，该坐标及其附近的点被认定为等待红灯的队首位置。

(3) 周期划分：队首车辆速度从 0 变为正值的瞬间对应红灯切换至绿灯的时刻，进而可以推算出每个路口红绿灯总周期，按照周期把 1 小时的观测时间分为了多个等长的时间段，一个时间段总是以绿灯开始，红灯结束。

(4) 信号灯周期推算：基于交通流理论，车辆穿过路口的时刻可以被视为绿灯的时刻。由于红灯并不阻碍离路口较远的车辆的行驶状态，即红灯时间和车辆速度不存在直接的双向对应关系，然而基本可以通过头部车辆的静止推断红灯状态。由此我们可以对每个周期内确定的绿灯时刻、红灯时刻进行标注，然后通过统计分析可以确定每个周期的第一个红灯时间点与周期开始时间的差的最小值，以及每个周期的最后一个绿灯时间点与周期开始时间的差的最大值，一个周期内绿灯的持续时间应位于两个值附近。

4.2.2 模型建立

首先用差分法计算各时间点速度的表达式，补充到数据集中：

$$v(t)=\frac{\sqrt{(x(t)-x(t-1))^2-(y(t)-y(t-1))^2}}{(t-(t-1))} \tag{1}$$

其中，$(x(t),y(t))$ 是车辆在 t 时刻的位置坐标。

根据路口 3 处 vehicle_id 为 640、648 的车辆绘制速度—时间序列图。可以观察到两车在第 1552 秒前等待红灯，第 1552 秒后通过路口：

通过分析观测数据中车辆位置的聚集趋势，分析得到车辆在红灯期间停留的最频繁区域，进而可以识别出等待红灯的队首位置。

以下是等待队首区域识别的结果:

表 **2**　等待队首区域

路口	1	2	3	4	5
x	11.4	-11.83	-1.6	0.46	3.64
y	4.8 或 1.6	-3.64	11.4	-11.51	-11.83

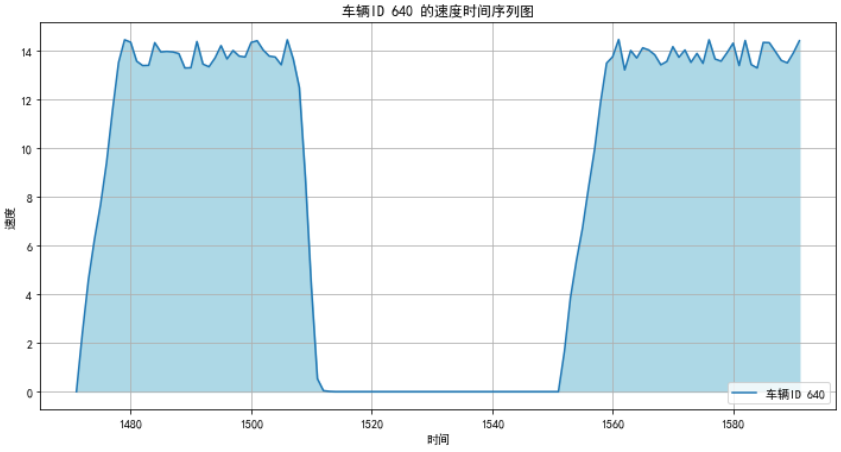

图 **6**　路口 **3** 车辆 **640** 的速度-时间序列图

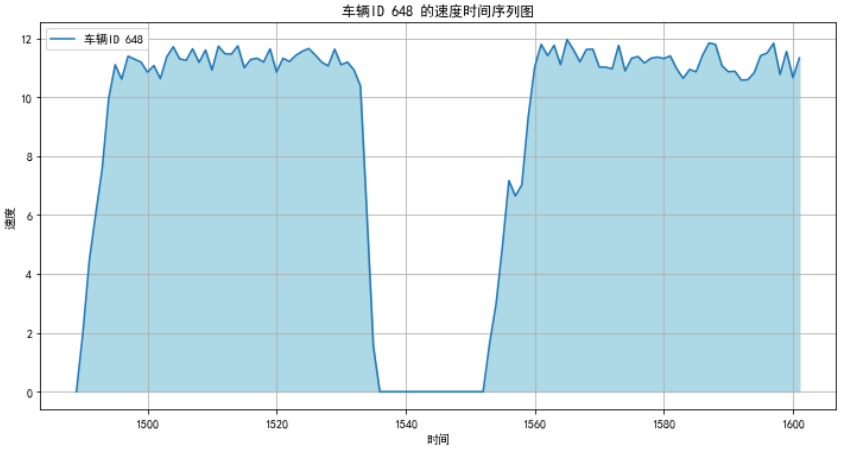

图 **7**　路口 **3** 车辆 **648** 的速度-时间序列图

按照下面的规则，对重要的过程状态进行识别与标注：

$$\begin{cases} \text{slowing-down} = v_{\text{current}} < v_{\text{previous}}, \\ \text{stopped} = \text{speed} \leq \text{threshold}, \\ \text{red} = \text{stopped} \wedge (y < y'), \\ \text{green} = (x \cdot x_{\text{prev}} < 0). \end{cases} \tag{2}$$

各行分别表示减速检测、停止检测、红灯标注以及绿灯标注。绿灯标注中，判断条件是车辆是否穿过中心线，即当前位置和前一位置的 x 坐标乘积是否小于零。

在周期识别方法中，关键时间点是通过监测车辆从停止状态到运动状态的转变来确定的，每个路口关键时间点呈现等距分布，容易识别出信号周期：

在模型中，重点关注在同余调整的条件下，交通信号周期的倍数时刻是否有车辆开始移动，这些时刻被标记为一般切换点，符合这一模式的时间点被视为周期内的正常绿灯开始时刻，那些不在预定周期倍数时刻发生的切换，被标记为异常切换点，异常切换

表 **3**　**A1-A5** 信号周期

路口	1	2	3	4	5
周期	105	88	105	88	88

点的数量相对较少，可能由数据误差或非典型交通状况引起，在本模型中可以被忽略。

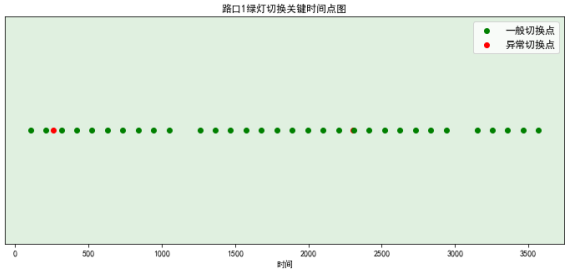

图 **8**　**A1** 绿灯切换点序列

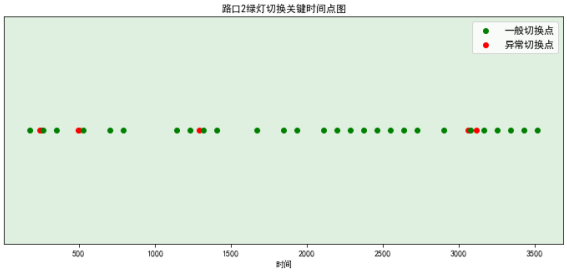

图 **9**　**A2** 绿灯切换点序列

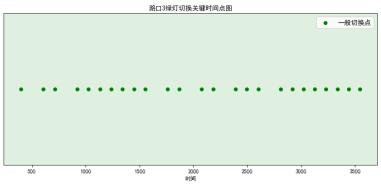

图 **10**　**A3** 绿灯切换点序列

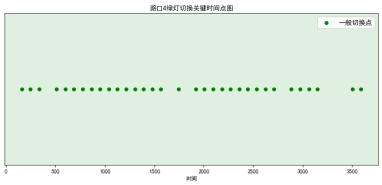

图 **11**　**A4** 绿灯切换点序列

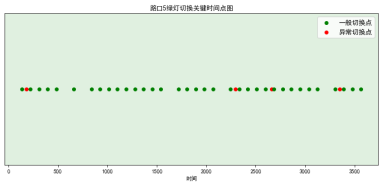

图 **12**　**A5** 绿灯切换点序列

对每个周期，检索该时间范围内的红灯和绿灯时间点，特别是周期内的第一个红灯时间点和最后一个绿灯时间点，用条形图将每个周期的红灯和绿灯时间差进行可视化，发现最后一个绿灯时间点和最后一个红灯时间点十分接近，取算术平均值作为绿灯时长。以下是绘制的条形图：

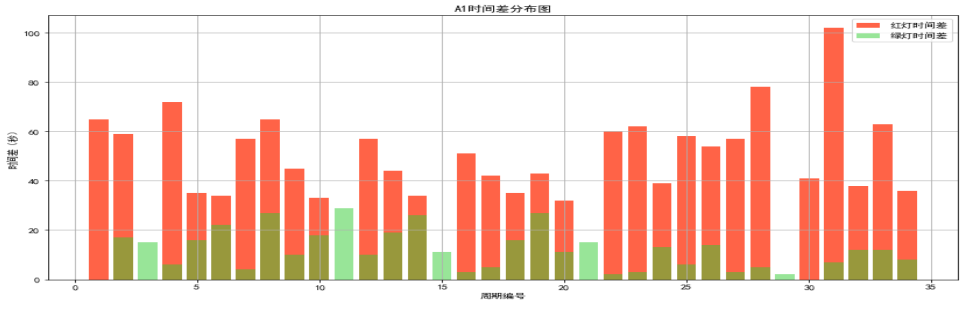

图 **13**　**A1** 时间差分布图

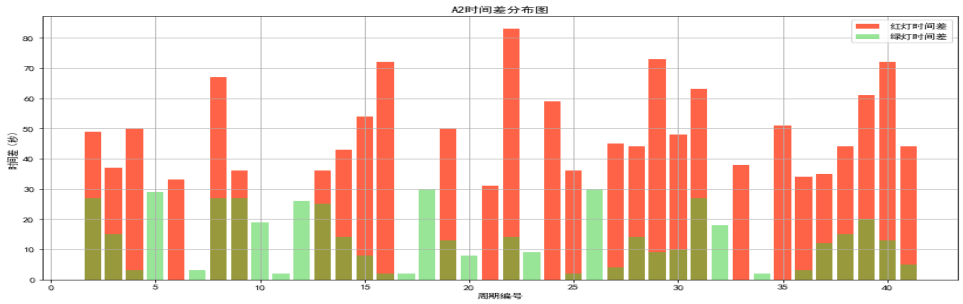

图 **14**　**A2** 时间差分布图

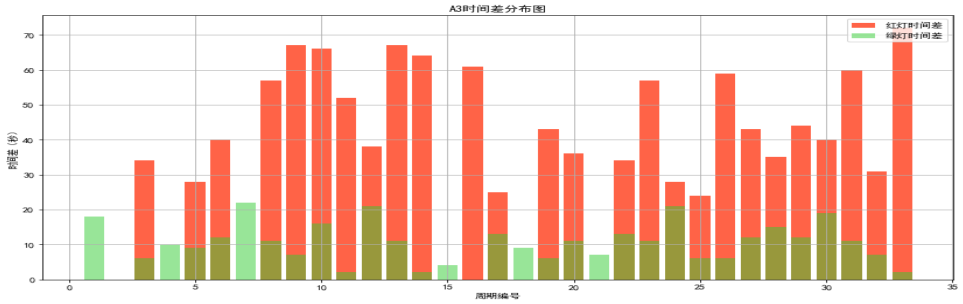

图 **15**　**A3** 时间差分布图

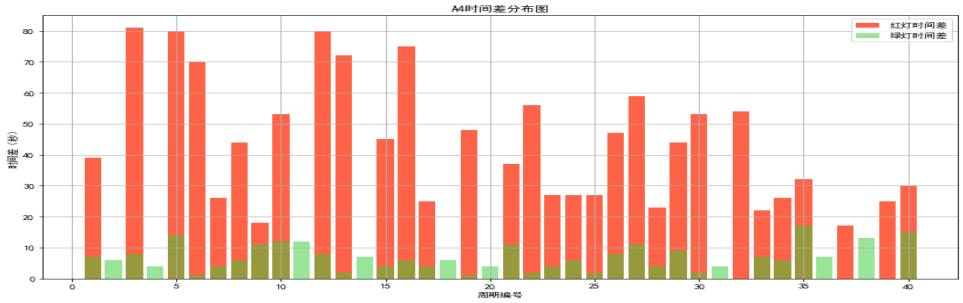

图 **16**　**A4** 时间差分布图

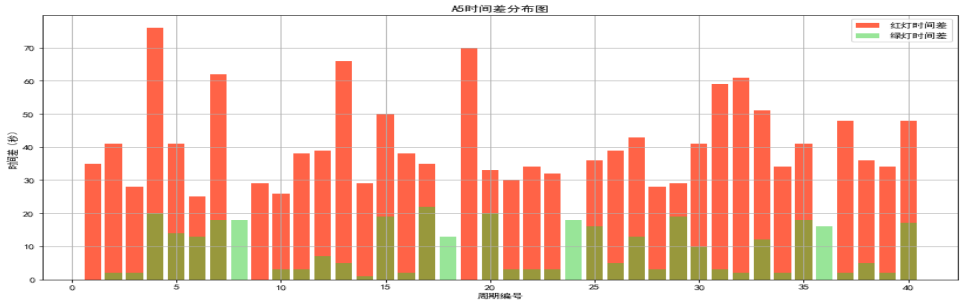

图 **17**　**A5** 时间差分布图

由上述结果得出各个路口最晚绿灯时间、最早红灯时间，由此得出结果：

表 **4**　路口 **A1-A5** 各自一个方向信号灯周期识别结果

路口	A1	A2	A3	A4	A5
红灯时长 (秒)	74.5	57.5	82	71	64.5
绿灯时长 (秒)	30.5	30.5	23	17	23.5

五、 问题二模型建立与求解

5.1 各因素对信号分析模型的影响评估

5.1.1 卡尔曼滤波法评估定位误差

在都市环境的定位应用中，普遍存在的多路径效应和信号遮挡等现象对定位系统的性能构成了挑战。为了精确评估并有效补偿由这些环境因素引起的定位误差，我们采用了卡尔曼滤波技术，将其集成到周期性信号分析的框架内。

卡尔曼滤波是一种先进的估计算法，用于在含噪测量条件下，对动态系统状态进行递归估计。它的核心在于一个两阶段的估计过程：在预测阶段，利用系统的先验知识，包括其状态转移模型和过程噪声统计特性，来产生下一时刻状态的预估值；在更新阶段，新的观测数据被用来修正预测值，通过一种加权平均的方式融合观测信息与预测信息，从而得到对当前系统状态的最优估计。

针对定位误差问题，我们采用自适应噪声协方差的卡尔曼滤波方法。这种改进版的卡尔曼滤波器能够自动调整噪声的统计特性，以适应观测数据的实际变化，从而提高滤波器对不确定环境的适应性和鲁棒性。通过实时调整内部噪声参数，该方法能够更有效地捕捉和抑制由于复杂城市环境引起的定位噪声，进而优化定位精度。具体过程如下：

（1）状态预测：使用状态转移矩阵预测下一个时刻的状态和误差协方差：

$$\begin{cases} \mathbf{x}_p = \mathbf{A}\mathbf{x}_e, \\ \mathbf{P}_p = \mathbf{A}\mathbf{P}\mathbf{A}^\top + \mathbf{Q}. \end{cases} \tag{3}$$

其中 $\mathbf{x}_p$ 是当前时刻的状态预测，P 是前一时刻的误差协方差，A 是系统动态矩阵，描述状态随时间的演变，Q 是过程噪声协方差矩阵，表示模型的不确定性。

（2）观测更新：将新的观测数据整合到状态估计中以校正预测状态:

$$
\begin{cases}
\mathbf{z} = \begin{bmatrix} x(k) \\ y(k) \end{bmatrix} \\
\mathbf{y} = \mathbf{z} - \mathbf{H}\mathbf{x}_p \\
\mathbf{S} = \mathbf{H}\mathbf{P}_p\mathbf{H}^\top + \mathbf{R} \\
\mathbf{K} = \mathbf{P}_p\mathbf{H}^\top\mathbf{S}^{-1} \\
\mathbf{x}_e = \mathbf{x}_p + \mathbf{K}\mathbf{y} \\
\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{P}_p
\end{cases} \tag{4}
$$

其中 **z** 是当前的观测变量，**y** 观测残差，**K** 是卡尔曼增益。

（3）自适应噪声协方差调整：根据观测残差的变化动态调整过程噪声协方差，以实现自适应调整:

$$
\begin{cases}
\text{var\_0} = \text{var}(y) \\
\mathbf{Q} = \mathbf{Q\_i} \cdot \text{residual\_var}
\end{cases} \tag{5}
$$

其中 residual_var 是观测残差的方差，**Q_i** 是初始过程噪声协方差矩阵，**Q** 是根据残差调整后的过程噪声协方差.

进行上述计算后，为了直观评估卡尔曼滤波器对车辆位置估计的改进效果，我们将每辆车的实测位置数据与经过卡尔曼滤波处理后估计得到的的位置数据在同一图形界面中进行对比展示:

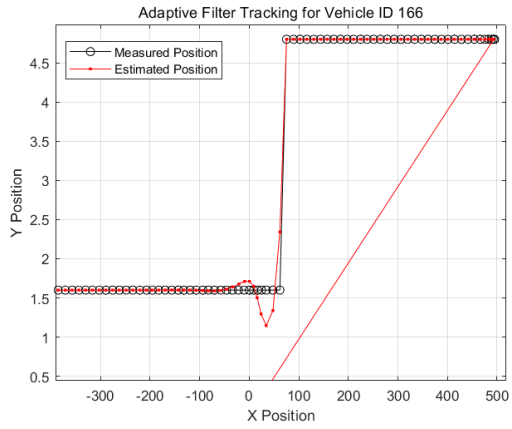

图 **18**　**A1** 车辆 **166** 轨迹对比

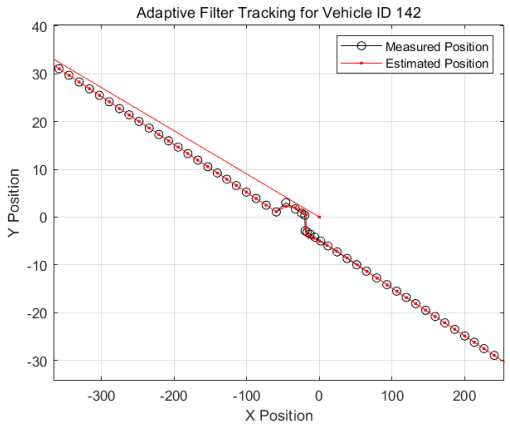

图 **19**　**A2** 车辆 **142** 轨迹对比

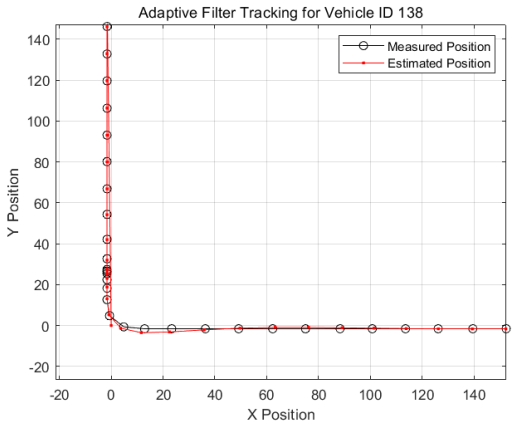

图 **20**　**A3** 车辆 **138** 轨迹对比

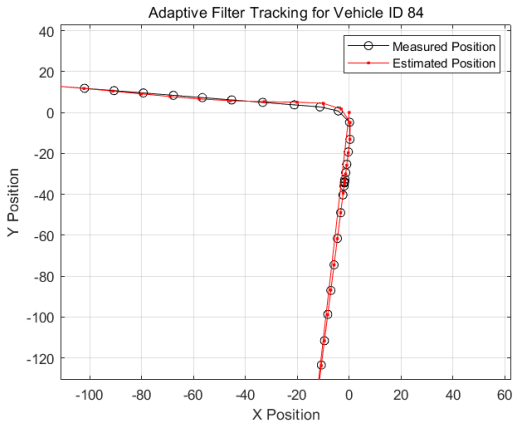

图 **21**　**A4** 车辆 **84** 轨迹对比

图形分析显示，在路口外部区域，卡尔曼滤波算法能够较好地拟合车辆的实际运动轨迹，表明滤波器在相对开阔区域的定位精度较高，而路口区域复杂的交通状况和信号遮挡引起的高定位误差使得估计位置与实际位置之间存在少量偏差。

将卡尔曼滤波算法求解结果整理到表格中，按照问题 1 建立的定周期信号分析模型求解，周期识别结果与问题 1 求解的结果差异很小，因此我们推断一定范围内的定位误差对模型精度影响较小。

5.1.2 蒙特卡洛模拟评估车辆比例对精度的影响

车辆比例指的是从整个可用数据集中抽取的车辆数据的比例。在交通信号周期分析中，使用的车辆数据比例较低可能导致数据不足，影响模型对交通流动特征的识别能力，进而影响信号周期的计算精度；较高的车辆数据比例则能够提供更全面的交通流态观测，提高周期计算的准确性。

蒙特卡洛模拟是一种基于概率和统计理论的模拟方法，通过重复随机抽样来计算或估算可能的结果分布，它可以模拟各种情况下的结果，并估计不确定性和概率分布，尤其适用于对具有随机变量的复杂系统进行分析。在评估车辆比例对信号周期分析精度的影响时，蒙特卡洛模拟法通过模拟不同车辆数据比例的抽取情况并进行多次实验，量化车辆比例变化对信号周期估计精度的具体影响。

针对本模型，我们设置车辆比例从 20% 以 10% 为增量逐步增加到 100%。对于每个车辆比例，重复进行 100 次蒙特卡洛模拟，每次模拟都随机抽取相应比例的车辆数据，并计算所对应的信号周期的最小红灯时间差和最大绿灯时间差，并计算二者的差值，记作 δ。此外，定义一个精度指标 A 以便于量化车辆比例对信号周期分析模型精度的影响：

$$A = (\frac{T-\delta}{T}) \times 100\% \tag{6}$$

其中，T 是预设的周期长度，用作差值的上限，当差值与周期相等时，精度为 0。

Algorithm 1 Monte Carlo Simulation for Vehicle Proportion Analysis

procedure MONTECARLOSIMULATION(D, N, S, E, s)
  for $p \leftarrow S$ **to** E **step** s **do**
    $R \leftarrow$ empty list
    for $i \leftarrow 1$ **to** N **do**
      $v\_ids \leftarrow$ random_choice($vehicle\_ids$, int(len($vehicle\_ids$) $\times$ p))
      $S\_D \leftarrow D$[filter by v_ids]
      $(r\_min, g\_max) \leftarrow$ analyze_signal_cycles(S_D)
      append (r_min, g_max) to R
    end for
    $(\bar{r}, \bar{g}) \leftarrow$ mean values of R
    $a \leftarrow$ calculate accuracy based on $\bar{r}, \bar{g}$
    append $(p, \bar{r}, \bar{g}, a)$ to $results$
  end for
  return $results$
end procedure

根据公式计算出每个路口不同车辆比例对应的定周期信号分析精度值。结果显示，随着车辆采样率的增加，信号周期分析模型的精度逐渐提高。我们也观察到，当车辆采样率达到某个阈值后，精度的提高幅度开始变得不明显。

表 **5** 车辆比例变化对信号周期估计精度的影响

	20%	30%	40%	50%	60%	70%	80%	90%	100%
路口 B1	87.51%	90.56%	91.93%	93.41%	93.70%	94.10%	94.49%	95.13%	95.32%
路口 B2	89.47%	92.73%	94.75%	95.60%	96.64%	96.30%	97.05%	97.53%	97.72%
路口 B3	88.09%	90.78%	93.27%	94.18%	95.03%	95.73%	96.16%	96.41%	96.76%
路口 B4	89.40%	91.93%	93.59%	95.26%	95.81%	96.35%	96.72%	97.78%	97.93%
路口 B5	88.49%	91.11%	91.95%	93.34%	94.23%	94.27%	94.65%	94.82%	94.94%

5.1.3 周期性密度法评估车流量对精度的影响

在交通流理论中，交通密度被认为是直接影响交通流稳定性的关键参数，不同交通密度条件下，交通信号周期计算的准确性可能存在显著差异，为了更精确的评估不同交通密度下交通信号周期的准确性，我们采用周期性密度法来评估车流量对模型精度的影响。

首先，我们对路口区域进行定义，即基于车辆的位置数据来界定一个固定半径的圆形区域，我们认为此圆形区域内的车辆处于路口区域内，随后我们计算每个时间点在路口区域内的车辆数量，以此来表示该时刻的车流量；接着，我们设置一个合理的阈值 q 用于区分高峰时段和非高峰时段，如果某个时间段内车辆数量不小于 q，则该时间段被

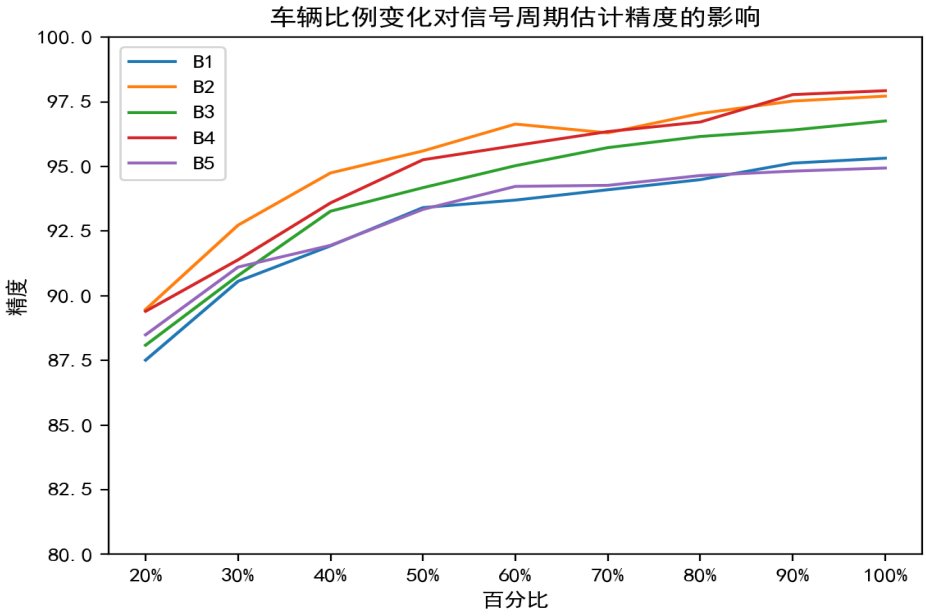

图 **22**　车辆比例变化对信号周期估计精度的影响曲线

识别为高峰时段。随后找到这些时间段所对应的信号周期。最后对高峰和非高峰时段使用 *analyze_signal_cycles* 函数进行分析，计算最小红灯时间差和最大绿灯时间差，以及对应的精度。以路口 A4 为例，一个小时内路口区域车流量变化如图所示：

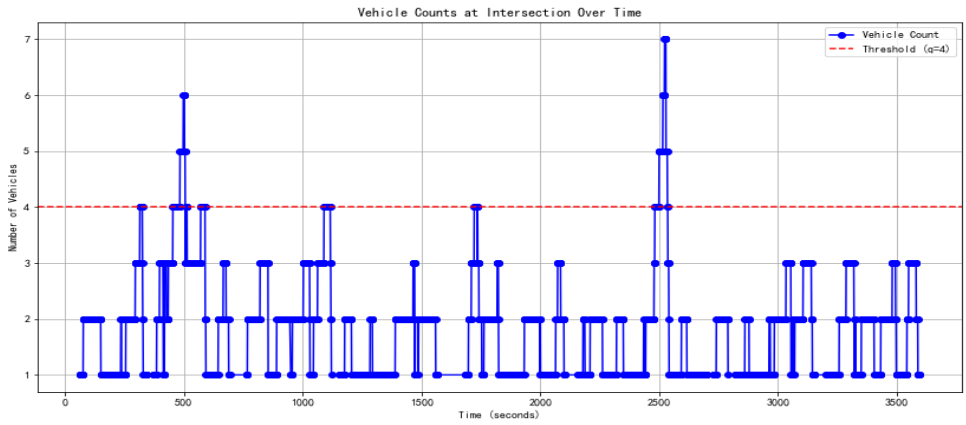

图 **23**　路口 **A4** 车流量变化

类似地，我们识别出各个路口的高峰时间段，根据上一节中对精度的定义，分别计算出两种车流量状态下信号周期模型的精度：

表 **6**　高峰时间段与精度对比

	高峰时间段（周期号）	高峰时间段精度	普通时间段精度
路口 1	5, 6, 10, 17, 18, 21	86.67%	96.19%
路口 2	5, 14, 15, 26, 32, 37, 38	84.09%	98.86%
路口 3	17, 20, 23, 24, 28, 29, 30, 31	93.33%	94.29%
路口 4	3, 5, 6, 12, 19, 28, 29	100%	90.91%
路口 5	4, 5, 6, 7, 12, 13, 20, 28, 29	94.32%	95.45%

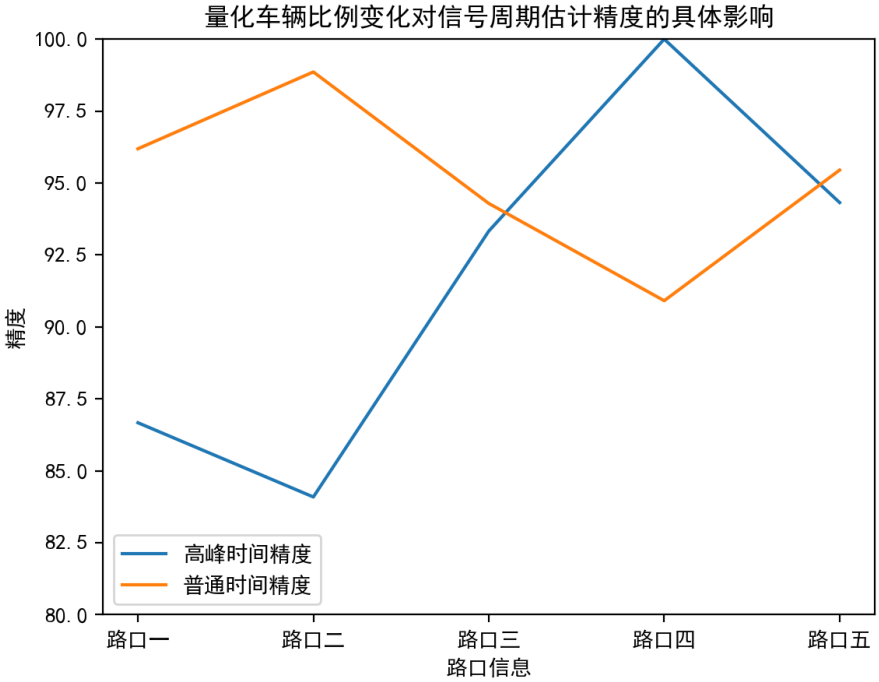

图 **24**　车流量变化精度曲线

结合图表可知，车流量的不同水平会对模型精度造成一定程度上的影响，但是仅关注高峰时间段或者普通时间段的情况都不能保证精度达到较高水平，这种情况在高车流量水平下表现尤为明显，例如交通拥堵和车辆之间的相互影响的增加，可能导致模型对车辆运动模式的预测变得更加复杂和不确定，从而降低模型的精度。

5.2 路口 B1-B5 信号灯周期求解

首先对附件二中的五个表格进行缺失项检查，确认没有缺失项。基于问题一中建立的定周期信号分析模型，对附件二中五个表格的数据进行处理，根据所给车辆位置数据采用差分法计算出各个时间点对应的车辆的速度，接着通过统计方法确认队首位置，随后进行过程状态的识别与标注，通过标注的关键时间点对信号周期进行划分，统计分析

每个周期的第一个红灯时间点与周期开始时间的差的最小值，以及每个周期的最后一个绿灯时间点与周期开始时间的差的最大值，据此确定信号灯周期。

虽然与附件 1 相比，附件 2 所提供的数据量大幅减少，但我们依然可以通过问题一中确定的方法计算出每个路口红灯 + 绿灯的总周期并绘制每个路口切换为绿灯的关键时间点图。

表 **7**　路口 **B1-B5** 的信号周期

路口	B1	B2	B3	B4	B5
信号周期	105	116	88	105	116

根据以下五张关键时间点图，可以发现关键点的分布明显比 A1-A5 稀疏：例如路口 1 的 2750-3500，路口 3 的 750-1500 时间段内出现了大段无切换点的空白区域，远大于 A1-A5 图像中的最大间隔。

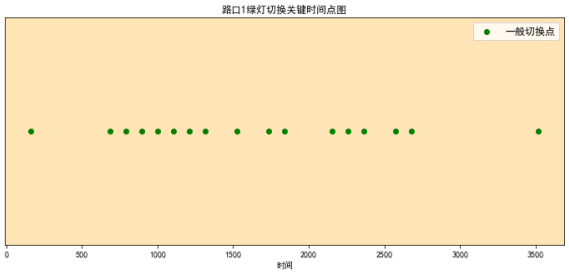

图 **25**　**B1** 绿灯切换点序列

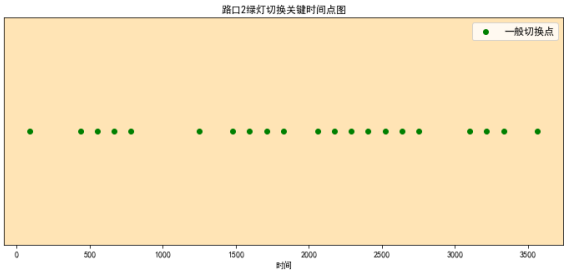

图 **26**　**B2** 绿灯切换点序列

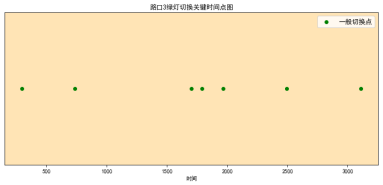

图 **27**　**B3** 绿灯切换点序列

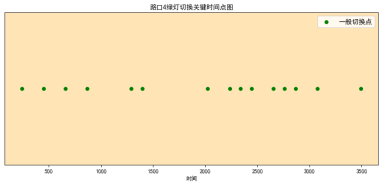

图 **28**　**B4** 绿灯切换点序列

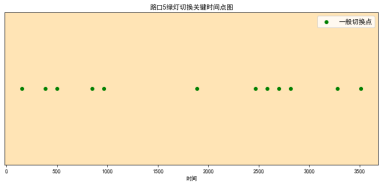

图 **29**　**B5** 绿灯切换点序列

与问题一类似，对每个周期，检索该时间范围内的红灯和绿灯时间点，特别是周期内的第一个红灯时间点和最后一个绿灯时间点，用与问题一同样的方法将每个周期的红灯和绿灯时间差进行可视化，发现最后一个绿灯时间点和最后一个红灯时间点十分接近，取算术平均值作为绿灯时长。

由上述得到问题二的结果如下：

表 **8**　路口 **B1-B5** 各自一个方向信号灯周期识别结果

路口	B1	B2	B3	B4	B5
红灯时长 (秒)	78.5	85	71	80.5	98
绿灯时长 (秒)	26.5	31	16.5	24.5	18

六、 问题三模型建立与求解

6.1 变周期信号分析模型的建立

在本问题的场景中，我们面临着信号灯周期可能发生变化的情况，但其具体变化形式尚不明确。这种变化可能体现为红绿灯整体周期长度的调整，也可能表现为在保持总周期不变的情况下，红绿灯各自持续时长的重新分配。基于问题一中关于定周期信号分析模型的部分理念，我们在此基础上对其进行拓展，对 C1-C6 六个路口的数据进行了必要的标注，其中包括速度计算、红灯点和绿灯点的标注，以及绿灯开始这一新设置关键点的标注，初步检测红绿灯总周期是否改变。

数据分析显示，在进行观测的两个小时内，只有路口 C3 的第 1025 到 1445 秒呈现出非偶然的绿灯切换关键时间点序列异常，如路口 C3 绿灯切换关键时间点图所示，这表明对于 C3 路口存在一个周期性变化，而其余五个路口均保持了红绿灯总周期的稳定性。基于以上发现，我们需要进一步研究周期内部参数是否改变，即红绿灯时长分配是否发生了调整。

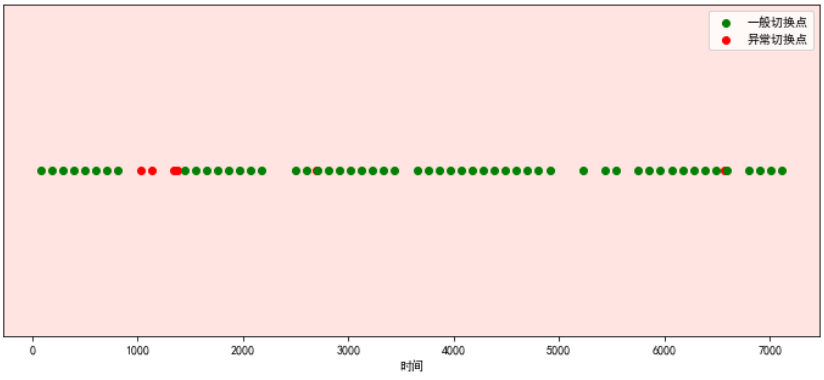

图 **30**　路口 **C3** 绿灯切换关键时间点图

6.2 带滑动窗口的贝叶斯变点检测模型

滑动窗口方法是一种在时间序列分析中常用的技术，它通过在数据上逐步滑动固定大小的“窗口”来提取局部特征或执行运算。本模型将设置了一个固定大小的窗口，专注于收集特定时间跨度内的最小红灯时间差和最大绿灯时间差，允许我们在面对数据大

量缺失的局限下，能够在局部范围内检测信号灯周期的变化，而不是囿于单一的全局视角。

贝叶斯变点检测是一种基于概率统计的方法，其核心假设是数据集中存在一种或多种潜在的分布变化。通过评估数据在分割前后的概率模型差异，贝叶斯变点检测能够确定数据中最可能的变点位置，从通过滑动窗口提取的特征中识别交通信号周期的变化。我们重点运用其中的 PELT 算法，PELT 算法的目标是最小化以下的成本函数总和：

$$\min\left(\sum_{i=1}^{m+1} C(y_{t_{i-1}:t_i}) + \beta \times m\right) \tag{7}$$

其中，$C(y_{t_{i-1}:t_i})$ 是从点 t_{i-1} 到点 t_i 的成本函数值。

PELT 算法通过动态规划的方式优化上述成本函数，能够高效的识别出潜在的信号周期变化点。

在本模型中，由于车辆数据完整度低，而相邻数据点间具有相似性，于是我们采用向前填充和向后填充平均值的方法来填补 red_time_differences 和 green_time_differences 两个序列的缺失值：

$$R_f[i] = \begin{cases} R[i] & \text{if } R[i] \text{ is not NaN} \\ R_f[i-1] & \text{if } R[i] \text{ is NaN and } i > 0 \end{cases} \tag{8}$$

$$R_b[i] = \begin{cases} R[i] & \text{if } R[i] \text{is not NaN} \\ R_b[i+1] & \text{if } R[i] \text{is NaN and } i < n-1 \end{cases} \tag{9}$$

$$R_{final}[i] = \frac{R_f[i] + R_b[i]}{2} \tag{10}$$

填补完毕后，采用定长窗口的方法来分析时间序列数据，对于每个定长窗口，计算窗口中 red_time_differences 的最小值和 green_time_differences 的最大值，这两个值形成一个二维特征空间中的点，再应用变点检测识别时间序列数据中的结构变化以找出交通信号周期发生变化的时间点，为了平衡变点检测的灵敏度和误报率，本模型设定了惩罚参数，避免过多的假阳性或假阴性出现，之后通过比较变点前后特征点的欧几里得距离筛选出变化最显著的点，最后实现从处理后的索引映射回实际时间的转换。下图以路口 C4 为例展现变点检测结果：

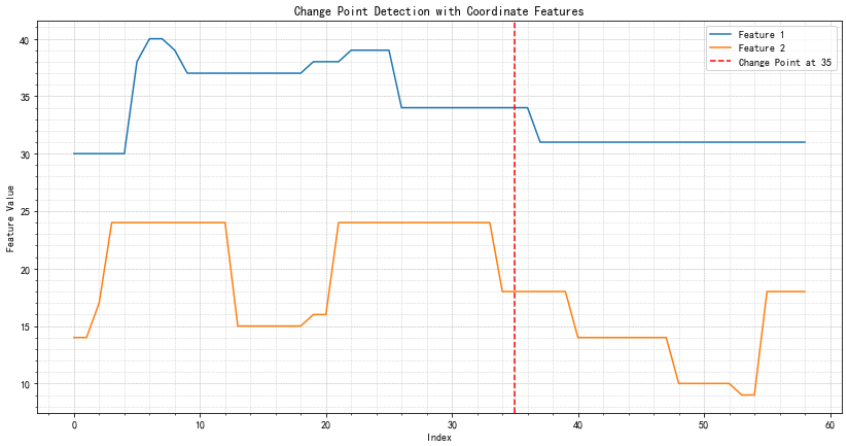

图 **31**　**C4** 贝叶斯变点检测图

完整周期识别结果如下表所示：

表 **9**　完整周期识别结果

	C1	C2	C3	C4	C5	C6
周期 1 红灯时长 (秒)	60	70	80	78	55	72
周期 1 绿灯时长 (秒)	28	18	25	27	33	33
周期切换时刻	880	4382	1027	4230	3080	无
周期 2 红灯时长 (秒)	52	72	82	80	58	
周期 2 绿灯时长 (秒)	36	16	25	25	30	
周期切换时刻			1445		6072	
周期 3 红灯时长 (秒)			80		51	
周期 3 绿灯时长 (秒)			25		37	

七、 问题四模型建立与求解

7.1 数据处理——全向路径分解

问题四的数据集给出了某路口连续 2 小时内所有方向样本车辆的轨迹数据，与之前的分析不同，本问题的关键在于，同一个路口各个方向的信号灯状态是相互制约的。此外，本题还需考察在观测窗口内信号周期的潜在变动。为了简化分析，我们首先根据车辆起始与终止位置的坐标特征，将数据集中的所有车辆按照通过路口前后所在的区域分为 12 类，这一分类不包括少量因为路径不完整而无法归类的车辆，方向的具体定义遵循如下规则：

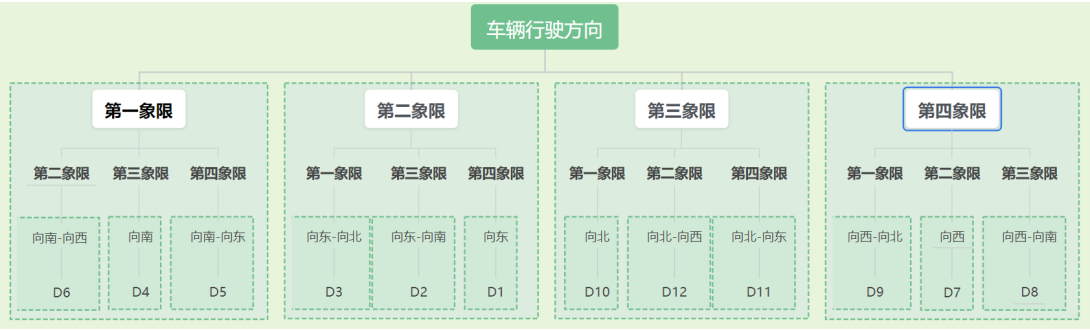

图 **32**　方向分类规则

7.2 整体性全路径周期求解

将所有车辆的路径分为 D1 到 D12 十二个方向后，我们延续先前的分析框架，探索该路口车辆共同遵循的周期。通过求解并可视化各路径的绿灯关键点序列，不难发现该路口的信号灯总周期保持 142 秒不变，但是在第 2414 秒出周期出现了统一的调整，如下图所示：

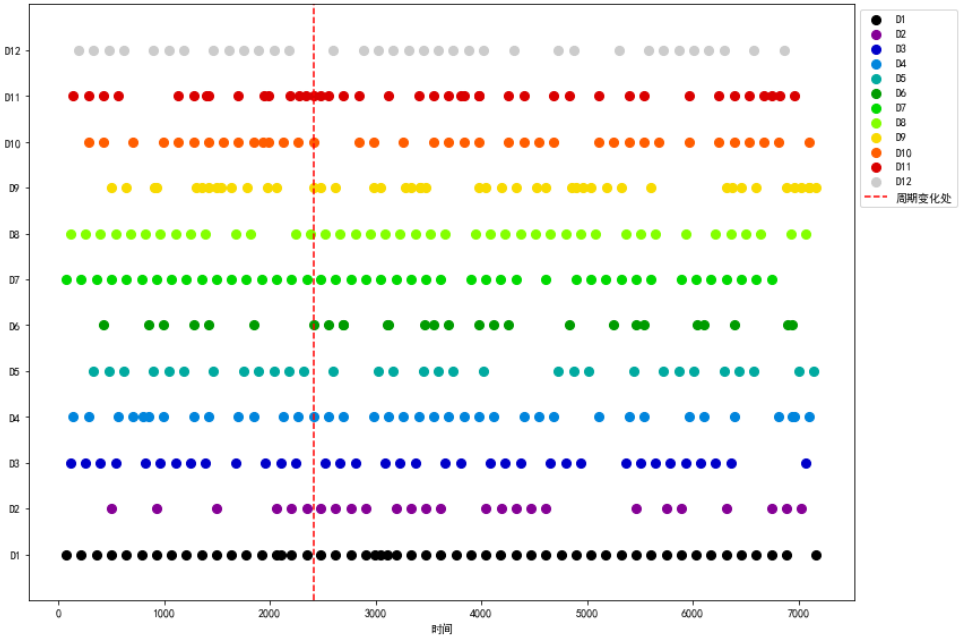

图 **33**　**12** 个方向的绿灯关键点序列

在周期变化点（第 2414 秒）前后，各方向车辆在一个周期 $T = 142s$ 内绿灯开始的时间如下表所示：

表 **10**　周期变化前各方向绿灯开始时间

方向	D4、D10	D6、D11	D5、D12	D1、D7	D2、D9	D3、D8
绿灯开始的时间	0	1	50	76	77	114

表 **11**　周期变化后前各方向绿灯开始时间

方向	D4、D10	D6、D11	D5、D12	D1、D7	D2、D9	D3、D8
绿灯开始的时间	0	1	45	68	69	110

将上表结论与现实生活对比，可以将该路口绿灯的通行顺序归纳为对立车道直行 + 右转、对立车道左转、另一对立车道直行 + 右转、另一车道左转的循环序列。为了提高结果的精度，我们以路线 D2 这一右转路线为例，按照问题二的求解方法估算右转的绿灯周期，发现绿灯周期不小于 60s，推断出同一方向的右转与左转可以并行。综上，我们得到该路口最后的信号灯周期表：

表 **12**　路口各方向红绿灯周期

方向	D1、D7	D2、D9	D3、D8	D4、D1	D5、D12	D6、D11
周期 1 红灯时长（秒）	104	76	116	92	116	66
周期 1 绿灯时长（秒）	38	66	28	50	26	76
周期切换时刻	2414	2414	2414	2414	2414	2414
周期 2 红灯时长（秒）	100	68	112	97	119	74
周期 2 绿灯时长（秒）	42	74	32	45	23	68

八、 模型的评价

8.1 模型的优点

在本研究中，我们建立了定周期信号分析、带滑动窗口的贝叶斯变点检测等模型，通过综合运用 Python、MATLAB 等工具，使用高级统计学方法和交通流理论，成功地提高了对城市交通信号灯周期的精准预测和实时调整能力。这种方法不仅在车辆轨迹数据的分析中表现出色，还能在没有直接连接到交通信号系统的数据支持下较为准确地预测和适应信号灯周期变化。具体而言，我们模型的优势在于其能够详细分析和计算车辆在各个时间点的速度变化，综合队首车辆的行为特征，以及通过交通流理论推导出每个信号周期中绿灯的最佳持续时间。

在模型的具体应用中，例如问题二所采用的卡尔曼滤波法和蒙特卡洛模拟，我们展示了模型的广泛性。即使在存在较大定位误差和车辆数据不完整的挑战下，依然能够保持健壮和准确。此外，我们的模型在识别和适应信号灯周期可能的快速变化中也显示了优异的性能，如通过带滑动窗口的贝叶斯变点检测模型，有效地从时间序列数据中筛选出周期变化的显著点，进一步增强了模型在复杂和不断变化的交通环境中的实际应用价值。

8.2 模型的局限性

本研究中使用的卡尔曼滤波和蒙特卡洛模拟等算法虽然功能强大，但计算复杂度较高，可能在实时或大规模应用场景中遇到性能瓶颈。卡尔曼滤波涉及多个矩阵计算步骤，如预测和更新阶段的矩阵乘法和逆运算。这些运算在状态空间或观测矩阵较大时特别耗费资源，因此，在处理大量数据或需要极快处理速度的系统中，计算负担可能成为限制因素；蒙特卡洛模拟要大量的随机抽样来获得精确结果，在大规模数据集上运行蒙特卡洛模拟可能需要显著的计算时间和资源。

参考文献

[1] 曾雅琼。基于模糊控制的红绿灯配时优化控制及应用 [D]. 长沙理工大学，2020. DOI: 10.26985/d.cnki.gcsjc.2020.000466.

[2] Killick, R., Fearnhead, P. and Eckley, I.A. Optimal detection of changepoints with a linear computational cost October 10, 2012

[3] 宋迎春。动态定位中的卡尔曼滤波研究 [D]. 中南大学，2006.

[4] 邵伟。蒙特卡洛方法及在一些统计模型中的应用 [D]. 山东大学，2012.

[5] 王腾。基于视频检测技术的交通车流量研究 [D]. 宁夏大学，2014.

[6] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. SIAM J. Comput., 31(6):1794–1813, 2002

[7] 李翊。基于贝叶斯方法的均匀分布变点的估计 [D]. 北京交通大学，2014.

[8] 李拂晓。几类时间序列模型变点监测与检验 [D]. 西北工业大学，2015.

[9] 王彬。基于时间分布的交通控制系统 [D]. 山东科技大学，2004.

九、 附录

9.1 编写的源程序

1. 判断附件一是否有缺失值
2. 绘制附件一散点图
3. 问题一 python 代码
4. 问题二的 matlab 代码 1
5. 问题二的 matlab 代码 2
6. 问题二的 python 代码 1
7. 问题二的 python 代码 2
8. 问题三的 python 代码
9. 问题四的 python 代码 1
10. 问题四的 python 代码 2
11. 问题四的 python 代码 3

1. 判断附件一是否有缺失值

```
import pandas as pd
import os

def check_missing_values(df, filename):
missing_values = df.isnull()
if missing_values.sum().sum() == 0:
print(f"{filename}没有空项")
else:
for i in range(len(df)):
for j in range(len(df.columns)):
if missing_values.iloc[i,j]:
print(f"{filename}第{i+1}行, 第{j+1}列存在空项")

folder_path = './附件4'
for filename in os.listdir(folder_path):
if filename.endswith('.csv'):
df = pd.read_csv(os.path.join(folder_path, filename))
check_missing_values(df, filename)
```

2. 绘制附件一散点图

```
import pandas as pd
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import os
import random

# 从文件夹中读取五个CSV文件
folder_path = 'C:\\B\\B\\f\\f1' # 替换为你的文件夹路径
csv_files = sorted([file for file in os.listdir(folder_path) if
    file.startswith('A')])[:5]

# 设置颜色
colors = ['b', 'g', 'r', 'c', 'm']

# 遍历每个按顺序选择的CSV文件并绘制对应的散点图
for i, csv_file in enumerate(csv_files):
# 读取CSV文件
df = pd.read_csv(os.path.join(folder_path, csv_file))

# 创建一个新的图形
plt.figure()

# 获取该CSV文件中所有不同的vehicle_id
unique_vehicle_ids = df['vehicle_id'].unique()

# 随机选择5个车辆
random_vehicle_ids = random.sample(list(unique_vehicle_ids), 5)

# 遍历每个随机选择的车辆并绘制对应的散点图
for j, vehicle_id in enumerate(random_vehicle_ids):
# 选取当前vehicle_id的数据
vehicle_data = df[df['vehicle_id'] == vehicle_id]

```

```
    # 若数据点不足20个，则提取全部数据
    if len(vehicle_data) < 20:
    sampled_vehicle_data = vehicle_data
    else:
    sampled_vehicle_data = vehicle_data.sample(n=20)

    # 提取x和y值
    x_values = sampled_vehicle_data['x'].tolist()
    y_values = sampled_vehicle_data['y'].tolist()

    # 绘制散点图，并指定颜色
    plt.scatter(x_values, y_values, color=colors[j], label=f'Vehicle
        {vehicle_id}')

    # 添加标题
    plt.title(f'vehicle location information - A{i+1}')

    # 添加标签
    plt.xlabel('X')
    plt.ylabel('Y')

    # 添加图例
    plt.legend()

    # 如果当前文件是 "A1"，则设置纵坐标范围为 -5 到 10
    if csv_file == "A1.csv":
    plt.ylim(-5, 10)

    # 显示图形
    plt.show()
```

3. 问题一 python 代码

```
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
```

```
4
5   data = pd.read_csv('A3.csv')
6   data.sort_values(by=['vehicle_id', 'time'], inplace=True)
7
8   # 计算每个车辆每个时间点的速度
9   data['x_diff'] = data.groupby('vehicle_id')['x'].diff().fillna(0)
10  data['y_diff'] = data.groupby('vehicle_id')['y'].diff().fillna(0)
11  data['time_diff'] =
        data.groupby('vehicle_id')['time'].diff().fillna(1)
12
13  # 计算速度
14  data['speed'] = np.sqrt(data['x_diff']**2 + data['y_diff']**2) /
        data['time_diff']
15
16  # 标注减速数据：如果当前速度小于前一时间点的速度，则认为是在减速
17  data['slowing_down'] = data.groupby('vehicle_id')['speed'].diff() < 0
18
19  # 标注停车数据：速度接近零
20  data['stopped'] = data['speed'] <= 1
21
22  # 标注红灯数据点
23  data['red'] = (
24  (data['stopped'] & ((data['y'] < 11.8) ))).astype(int)
25
26  # 标注绿灯通过路口的时间点
27  data['prev_x'] = data.groupby('vehicle_id')['x'].shift(1)
28  data['green'] = (data['x'] * data['prev_x'] < 0).astype(int) #
        当前x与前一个x乘积小于0
29
30  # 导出结果
31  data.to_csv('processed2_A3.csv', index=False)
32
33  unique_vehicle_ids = data['vehicle_id'].nunique()
34  print("不同的车辆ID数量：", unique_vehicle_ids)
35
```

```
    # 检测从停止到运动的转变，且前两个时间点y坐标都是11.4
    data['prev_y_1'] = data.groupby('vehicle_id')['y'].shift(1)
    data['prev_y_2'] = data.groupby('vehicle_id')['y'].shift(2)
    data['prev_stopped_1'] =
        data.groupby('vehicle_id')['stopped'].shift(1)

    data['green_light'] = ((data['speed'] > 0) & (data['prev_stopped_1']
        == True) &
    (data['prev_y_1'] == 11.4) & (data['prev_y_2'] == 11.4))

    # 寻找并打印关键时间点
    green_light_times = data[data['green_light']]['time'].unique()
    print("绿灯切换的关键时间点: ", green_light_times)

    # 绘制绿灯切换关键时间点图
    plt.figure(figsize=(12, 5))

    # 获取绿灯切换关键时间点
    green_light_times = data[data['green_light']]['time'].unique()

    general_added = False
    exception_added = False

    # 绘制每个时间点的散点图
    for time in green_light_times:
    # 判断时间点是否Mod105为82
    if time % 105 == 82:
    color = 'green'
    label = '一般切换点'
    if not general_added:
    plt.scatter(time, 0, color=color, marker='o', label=label)
    general_added = True
    else:
    plt.scatter(time, 0, color=color, marker='o')
    else:
```

```
    color = 'red'
    label = '异常切换点'
    if not exception_added:
    plt.scatter(time, 0, color=color, marker='o', label=label)
    exception_added = True
    else:
    plt.scatter(time, 0, color=color, marker='o')

    plt.rcParams['font.sans-serif'] = ['SimHei'] #解决中文显示
    plt.rcParams['axes.unicode_minus'] = False #解决符号无法显示
    plt.title('路口3绿灯切换关键时间点图')
    plt.xlabel('时间')
    plt.ylabel(' ')
    plt.grid(False)
    plt.yticks([])
    plt.legend(prop={'size': 12})

    plt.gca().set_facecolor('#e0f0e0')
    plt.show()

    # 绘制前50辆车的速度时间序列图
    # 获取前50辆车的车辆ID
    top_50_vehicle_ids = data['vehicle_id'].unique()[:50]

    # 遍历前50辆车，绘制每个车辆的速度时间序列图
    for vehicle_id in top_50_vehicle_ids:
    # 创建一个新的画布
    plt.figure(figsize=(12, 6))
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    # 获取当前车辆的数据
    vehicle_data = data[data['vehicle_id'] == vehicle_id]

    # 绘制速度时间序列图
    plt.plot(vehicle_data['time'], vehicle_data['speed'], label=f'车辆ID
```

```
    {vehicle_id}')

# 添加浅蓝色阴影
plt.fill_between(vehicle_data['time'], vehicle_data['speed'],
    color='lightblue')

plt.title(f'车辆ID {vehicle_id} 的速度时间序列图')
plt.xlabel('时间')
plt.ylabel('速度')
plt.grid(True)
plt.legend()
plt.show()


# 载入数据
data = pd.read_csv('processed2_A3.csv')

# 设置周期长度
T = 105

# 选取时间段
k = 17
start = T * k - 23
end = T * k + 81
period_data = data[(data['time'] >= start) & (data['time'] <= end)]

# 绘制图形
plt.figure(figsize=(10, 5))

# 检查并绘制红灯时间点的红点
if 'red' in period_data.columns and not
    period_data[period_data['red'] == 1].empty:
plt.scatter(period_data[period_data['red'] == 1]['time'],
[1] * len(period_data[period_data['red'] == 1]),
color='red', label='红灯')
```

```
    else:
    print(f"在第{k}周期内没有红灯标记。")

    # 检查并绘制绿灯时间点的绿点
    if 'green' in period_data.columns and not
       period_data[period_data['green'] == 1].empty:
    plt.scatter(period_data[period_data['green'] == 1]['time'],
    [1] * len(period_data[period_data['green'] == 1]),
    color='green', label='绿灯')
    else:
    print(f"在第{k}周期内没有绿灯标记。")

    # 添加标题和标签
    plt.title(f'第{k}周期红绿灯标记图')
    plt.xlabel('时间')
    plt.ylabel('红/绿灯标记')
    plt.yticks([1], ['信号灯'])
    plt.legend()
    plt.xlim(start, end)
    plt.show()


    # 创建列表来存储每个周期的红灯和绿灯时间差
    red_time_differences = []
    green_time_differences = []

    # 设置周期长度
    T = 105

    for k in range(1, 34):
    start = T * k - 23
    end = T * k + 81
    period_data = data[(data['time'] >= start) & (data['time'] <= end)]

    # 寻找第一个红灯时间点
```

```
    if not period_data[period_data['red'] == 1].empty:
    first_red_time = period_data[period_data['red'] == 1]['time'].iloc[0]
    red_time_difference = first_red_time - start
    else:
    red_time_difference = np.nan # 如果该周期内没有红灯记录，则设置为NaN

    # 寻找最后一个绿灯时间点
    if not period_data[period_data['green'] == 1].empty:
    last_green_time = period_data[period_data['green'] ==
        1]['time'].iloc[-1]
    green_time_difference = last_green_time - start
    else:
    green_time_difference = np.nan # 如果该周期内没有绿灯记录，则设置为NaN

    # 存储时间差
    red_time_differences.append(red_time_difference)
    green_time_differences.append(green_time_difference)

    print("每个周期的第一个红灯时间点与周期开始时间的差：")
    print(red_time_differences)
    print("每个周期的最后一个绿灯时间点与周期开始时间的差：")
    print(green_time_differences)

    # 绘制红灯和绿灯时间差的分布图
    plt.figure(figsize=(14, 7))
    plt.bar(range(1, 34), red_time_differences, color='#FF6347',
        label='红灯时间差')
    plt.bar(range(1, 34), green_time_differences, color='#32CD32',
        label='绿灯时间差', alpha=0.5)
    plt.title('A3时间差分布图')
    plt.xlabel('周期编号')
    plt.ylabel('时间差（秒）')
    plt.legend()
    plt.grid(True)
    plt.show()
```

4. 问题二的 matlab 代码 1

```
data = readtable('A4.csv');

% 预处理：按车辆ID和时间排序
data = sortrows(data, {'vehicle_id', 'time'});

% 数据采样间隔
dt = 1;

% 系统动态矩阵和观测矩阵
A = [1 dt 0.3*dt^2 0 0 0;
0  1 dt      0 0 0;
0  0 1       0 0 0;
0  0 0       1 dt 0.3*dt^2;
0  0 0       0 1  dt;
0  0 0       0 0  1];
H = [1 0 0 0 0 0; 0 0 0 1 0 0];

% 初始过程噪声和观测噪声方差
Q_init = blkdiag(0.1*eye(3), 0.1*eye(3)); % 过程噪声
R = eye(2) * 1;                           % 观测噪声

% 获取唯一的车辆ID列表，只取前15辆车
vehicle_ids = unique(data.vehicle_id);
vehicle_ids = vehicle_ids(1:15);

% 为每辆车应用自适应滤波
for vid = vehicle_ids'
% 选取当前车辆的数据
vehicle_data = data(data.vehicle_id == vid, :);
n = height(vehicle_data);

% 初始化状态向量
x_est = [vehicle_data.x(1); 0; 0; vehicle_data.y(1); 0; 0];
P = eye(6); % 初始误差协方差
```

```
35
36  % 为当前车辆存储估计的位置
37  estimated_positions = zeros(n, 2);
38
39  % 自适应滤波主循环
40  for k = 2:n
41  % 预测
42  x_pred = A * x_est;
43  P_pred = A * P * A' + Q_init; % 使用初始过程噪声方差
44
45  % 更新
46  z = [vehicle_data.x(k); vehicle_data.y(k)];
47  y = z - H * x_pred;
48  S = H * P_pred * H' + R;
49  K = P_pred * H' / S;
50  x_est = x_pred + K * y;
51  P = (eye(6) - K * H) * P_pred;
52
53  % 观测残差的方差
54  residual_var = var(y);
55
56  % 调整过程噪声方差
57  Q = Q_init * residual_var;
58
59  % 存储估计位置
60  estimated_positions(k, :) = x_est([1, 4])';
61  end
62
63  % 绘制当前车辆的结果
64  figure;
65  plot(vehicle_data.x, vehicle_data.y, 'ko-', 'DisplayName', 'Measured
        Position');
66  hold on;
67  plot(estimated_positions(:, 1), estimated_positions(:, 2), 'r.-',
        'DisplayName', 'Estimated Position');
```

```
    legend;
    xlabel('X Position');
    ylabel('Y Position');
    title(sprintf('Adaptive Filter Tracking for Vehicle ID %d', vid));
    grid on;
    end
```

5. 问题二的 matlab 代码 2

```
data = readtable('A4.csv');

% 预处理：按车辆ID和时间排序
data = sortrows(data, {'vehicle_id', 'time'});

% 假设数据采样间隔为1秒
dt = 1;

% 系统动态矩阵和观测矩阵
A = [1 dt 0.3*dt^2 0 0 0;
0  1 dt      0 0 0;
0  0 1       0 0 0;
0  0 0       1 dt 0.3*dt^2;
0  0 0       0 1  dt;
0  0 0       0 0  1];
H = [1 0 0 0 0 0; 0 0 0 1 0 0];

% 过程噪声和观测噪声协方差
Q = blkdiag(0.01*eye(3), 0.01*eye(3)); % 过程噪声
R = eye(2) * 0.1;                      % 观测噪声

% 获取唯一的车辆ID列表
unique_vehicles = unique(data.vehicle_id);

% 创建新表格来存储滤波后的结果
results = data;
results.x_filtered = zeros(height(data), 1); %
```

```
    添加新列存储滤波后的x位置
results.y_filtered = zeros(height(data), 1); %
    添加新列存储滤波后的y位置

% 卡尔曼滤波处理
for vid = unique_vehicles'
vehicle_data = data(data.vehicle_id == vid, :);
n = height(vehicle_data);

% 初始化状态向量
x_est = [vehicle_data.x(1); 0; 0; vehicle_data.y(1); 0; 0];
P = eye(6); % 初始误差协方差

% 卡尔曼滤波主循环
for k = 1:n
global_index = find(data.vehicle_id == vid & data.time ==
    vehicle_data.time(k));
% 预测
x_pred = A * x_est;
P_pred = A * P * A' + Q;

% 更新
z = [vehicle_data.x(k); vehicle_data.y(k)];
y = z - H * x_pred;
S = H * P_pred * H' + R;
K = P_pred * H' / S;
x_est = x_pred + K * y;
P = (eye(6) - K * H) * P_pred;

% 存储估计的位置
results.x_filtered(global_index) = x_est(1);
results.y_filtered(global_index) = x_est(4);
end
end

```

```
% 保存结果
writetable(results, 'A4_Kalman.csv');
```

6. 问题二的 python 代码 1

```
import pandas as pd
import numpy as np

T = 105

def analyze_signal_cycles(vehicle_data):
vehicle_data = vehicle_data.copy() # 创建副本
vehicle_data.sort_values(by=['vehicle_id', 'time'], inplace=True)

vehicle_data.loc[:, 'x_diff'] =
    vehicle_data.groupby('vehicle_id')['x'].diff().fillna(0)
vehicle_data.loc[:, 'y_diff'] =
    vehicle_data.groupby('vehicle_id')['y'].diff().fillna(0)
vehicle_data.loc[:, 'time_diff'] =
    vehicle_data.groupby('vehicle_id')['time'].diff().fillna(1)

vehicle_data.loc[:, 'speed'] = np.sqrt(vehicle_data['x_diff']**2 +
    vehicle_data['y_diff']**2) / vehicle_data['time_diff']

vehicle_data.loc[:, 'slowing_down'] =
    vehicle_data.groupby('vehicle_id')['speed'].diff() < 0
vehicle_data.loc[:, 'stopped'] = vehicle_data['speed'] <= 1

vehicle_data.loc[:, 'red'] = (vehicle_data['stopped'] &
    (vehicle_data['y'] < 11.8)).astype(int)
vehicle_data.loc[:, 'prev_x'] =
    vehicle_data.groupby('vehicle_id')['x'].shift(1)
vehicle_data.loc[:, 'green'] = (vehicle_data['x'] *
    vehicle_data['prev_x'] < 0).astype(int)

red_time_differences = []
```

```
    green_time_differences = []

    for k in range(1, 34):
    start = T * k - 23
    end = T * k + 81
    period_data = vehicle_data[(vehicle_data['time'] >= start) &
        (vehicle_data['time'] <= end)]

    if not period_data[period_data['red'] == 1].empty:
    first_red_time = period_data[period_data['red'] == 1]['time'].iloc[0]
    red_time_difference = first_red_time - start
    if red_time_difference > 0:
    red_time_differences.append(red_time_difference)

    if not period_data[period_data['green'] == 1].empty:
    last_green_time = period_data[period_data['green'] ==
        1]['time'].iloc[-1]
    green_time_difference = last_green_time - start
    if green_time_difference > 0:
    green_time_differences.append(green_time_difference)

    red_time_differences = [val for val in red_time_differences if not
        np.isnan(val) and val != 0]
    green_time_differences = [val for val in green_time_differences if
        not np.isnan(val) and val != 0]

    return min(red_time_differences), max(green_time_differences)


    def monte_carlo_simulation(data, num_samples=100, start_prop=0.2,
        end_prop=1.0, step=0.1):
    data = pd.read_csv(data)

    # 获取唯一车辆ID列表
    vehicle_ids = data['vehicle_id'].unique()
```

```
54
55      # 存储每次模拟结果
56      results = []
57
58      for prop in np.arange(start_prop, end_prop + step, step):
59      prop_results = []
60      for _ in range(num_samples):
61      # 随机选择一定比例的车辆ID
62      sampled_ids = np.random.choice(vehicle_ids,
            size=int(len(vehicle_ids) * prop), replace=True)
63
64      # 一个车辆要保证有完整数据
65      sampled_data = data[data['vehicle_id'].isin(sampled_ids)]
66
67      # 使用封装函数计算最小的红灯时间差和最大的绿灯时间差
68      red_min, green_max = analyze_signal_cycles(sampled_data)
69
70      prop_results.append((red_min, green_max))
71
72      # 计算平均最小红灯时间差和平均最大绿灯时间差
73      avg_red_min = np.mean([res[0] for res in prop_results if res[0] is
            not None])
74      avg_green_max = np.mean([res[1] for res in prop_results if res[1] is
            not None])
75      diff = avg_red_min - avg_green_max
76
77      # 计算精度
78      accuracy = ((T - diff) / (T - 0)) * 100 if diff < T else 0
79      accuracy = max(0, min(accuracy, 100))
80
81      # 存储这个比例下的平均结果和精度
82      results.append((prop, avg_red_min, avg_green_max, accuracy))
83
84      return results
85
```

```
# 调用函数并打印结果
simulation_results = monte_carlo_simulation('A3.csv')
for result in simulation_results:
print(f"Proportion: {result[0]:.2f}, Average Min Red Time Difference: {result[1]:.2f}, Average Max Green Time Difference: {result[2]:.2f}, Accuracy: {result[3]:.2f}%")
```

7. 问题二的 python 代码 2

```
import pandas as pd
import numpy as np

T = 88

def analyze_signal_cycles(vehicle_data, cycles):
vehicle_data = vehicle_data.copy() # 创建数据的副本
vehicle_data.sort_values(by=['vehicle_id', 'time'], inplace=True)

vehicle_data.loc[:, 'x_diff'] = vehicle_data.groupby('vehicle_id')['x'].diff().fillna(0)
vehicle_data.loc[:, 'y_diff'] = vehicle_data.groupby('vehicle_id')['y'].diff().fillna(0)
vehicle_data.loc[:, 'time_diff'] = vehicle_data.groupby('vehicle_id')['time'].diff().fillna(1)

vehicle_data.loc[:, 'speed'] = np.sqrt(vehicle_data['x_diff']**2 + vehicle_data['y_diff']**2) / vehicle_data['time_diff']

vehicle_data.loc[:, 'slowing_down'] = vehicle_data.groupby('vehicle_id')['speed'].diff() < 0
vehicle_data.loc[:, 'stopped'] = vehicle_data['speed'] <= 1

vehicle_data.loc[:, 'red'] = (vehicle_data['stopped'] & (vehicle_data['y'] > -12.1)).astype(int)
vehicle_data.loc[:, 'prev_y'] = vehicle_data.groupby('vehicle_id')['y'].shift(1)
```

```
    vehicle_data.loc[:, 'green'] = (vehicle_data['y'] *
        vehicle_data['prev_y'] < 0).astype(int)

    red_time_differences = []
    green_time_differences = []

    for k in cycles:
    start = T * k - 42
    end = T * k + 45
    period_data = vehicle_data[(vehicle_data['time'] >= start) &
        (vehicle_data['time'] <= end)]

    if not period_data[period_data['red'] == 1].empty:
    first_red_time = period_data[period_data['red'] == 1]['time'].iloc[0]
    red_time_difference = first_red_time - start
    if red_time_difference > 0:
    red_time_differences.append(red_time_difference)

    if not period_data[period_data['green'] == 1].empty:
    last_green_time = period_data[period_data['green'] ==
        1]['time'].iloc[-1]
    green_time_difference = last_green_time - start
    if green_time_difference > 0:
    green_time_differences.append(green_time_difference)

    red_time_differences = [val for val in red_time_differences if not
        np.isnan(val) and val != 0]
    green_time_differences = [val for val in green_time_differences if
        not np.isnan(val) and val != 0]

    return min(red_time_differences), max(green_time_differences)

    def calculate_accuracy(min_red, max_green, T=88):
    # 计算时间差的差值
    delta = min_red - max_green
```

```
    # 根据定义的公式计算精度
    accuracy = ((T - delta) / T) * 100
    # 确保精度为非负值
    accuracy = max(0, min(accuracy, 100))
    return accuracy

    data = pd.read_csv('A5.csv')

    data.sort_values(by=['vehicle_id', 'time'], inplace=True)

    # 定义和统计路口区车辆
    data['distance_from_intersection'] = np.sqrt(data['x']**2 +
        data['y']**2)
    intersection_area_data = data[data['distance_from_intersection'] <=
        180]
    intersection_counts = intersection_area_data.groupby('time').size()

    # 设置车辆数阈值
    q = 4

    # 查找高峰周期
    high_traffic_periods = set((time + 42) // T for time in
        intersection_counts.index if intersection_counts[time] >= q)

    # 所有可能周期
    all_periods = set(range(1, 42))

    # 普通周期
    normal_traffic_periods = all_periods - high_traffic_periods

    # 分析高峰周期
    high_traffic_data = data[data['time'].apply(lambda x: (x + 42) // T
        in high_traffic_periods)]
    high_min_red, high_max_green =
        analyze_signal_cycles(high_traffic_data, high_traffic_periods)
```

```python

    # 分析普通周期
    normal_traffic_data = data[data['time'].apply(lambda x: (x + 42) //
        T in normal_traffic_periods)]
    normal_min_red, normal_max_green =
        analyze_signal_cycles(normal_traffic_data, normal_traffic_periods)

    print(f"High Traffic - Min Red Time Difference: {high_min_red}, Max
        Green Time Difference: {high_max_green}")
    print(f"Normal Traffic - Min Red Time Difference: {normal_min_red},
        Max Green Time Difference: {normal_max_green}")

    # 分析高峰和普通周期
    high_min_red, high_max_green =
        analyze_signal_cycles(high_traffic_data, high_traffic_periods)
    normal_min_red, normal_max_green =
        analyze_signal_cycles(normal_traffic_data, normal_traffic_periods)

    # 计算高峰和普通周期的精度
    high_accuracy = calculate_accuracy(high_min_red, high_max_green, T)
    normal_accuracy = calculate_accuracy(normal_min_red,
        normal_max_green, T)

    print(f"High Traffic Accuracy: {high_accuracy:.2f}%")
    print(f"Normal Traffic Accuracy: {normal_accuracy:.2f}%")
```

8. 问题三的 python 代码

```python
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import ruptures as rpt

    def fill_with_neighbors(series):
    """ 使用前后非缺失值的平均值填充缺失值 """
    # 向前填充和向后填充的临时序列
```

```
    filled_forward = series.fillna(method='ffill')
    filled_backward = series.fillna(method='bfill')

    # 计算前后平均
    filled_series = (filled_forward + filled_backward) / 2

    return filled_series

    data = pd.read_csv('C1.csv')

    data.sort_values(by=['vehicle_id', 'time'], inplace=True)

    # 计算每个车辆每个时间点的速度
    data['x_diff'] = data.groupby('vehicle_id')['x'].diff().fillna(0)
    data['y_diff'] = data.groupby('vehicle_id')['y'].diff().fillna(0)
    data['time_diff'] =
        data.groupby('vehicle_id')['time'].diff().fillna(1) # 防止除以0

    # 计算速度
    data['speed'] = np.sqrt(data['x_diff']**2 + data['y_diff']**2) /
        data['time_diff']

    # 标注减速数据：如果当前速度小于前一时间点的速度，则认为是在减速
    data['slowing_down'] = data.groupby('vehicle_id')['speed'].diff() < 0

    # 标注停车数据：速度接近零
    data['stopped'] = data['speed'] <= 1

    # 标注红灯数据点
    data['red'] = (
    (data['stopped'] & ((data['x'] < 12.5) ))).astype(int)

    # 标注绿灯通过路口的时间点
    data['prev_x'] = data.groupby('vehicle_id')['x'].shift(1)
    data['green'] = ((data['x'] - 2) * (data['prev_x'] - 2) <
```

```
    0).astype(int)

# 导出结果
data.to_csv('processed2_C1.csv', index=False)

data = pd.read_csv('processed2_C1.csv')

unique_vehicle_ids = data['vehicle_id'].nunique()
print("不同的车辆ID数量: ", unique_vehicle_ids)

# 检测从停止到运动的转变，且前两个时间点x坐标都是11.83
data['prev_x_1'] = data.groupby('vehicle_id')['x'].shift(1)
data['prev_x_2'] = data.groupby('vehicle_id')['x'].shift(2)
data['prev_stopped_1'] =
    data.groupby('vehicle_id')['stopped'].shift(1)

data['green_light'] = ((data['speed'] > 0) & (data['prev_stopped_1']
    == True) &
(data['prev_x_1'] == 11.83) & (data['prev_x_2'] == 11.83))

# 寻找并打印关键时间点
green_light_times = data[data['green_light']]['time'].unique()
print("绿灯切换的关键时间点: ", green_light_times)
# 绘制绿灯切换关键时间点图
plt.figure(figsize=(12, 5))

general_added = False
exception_added = False

# 存储异常切换点
exception_times = []

# 绘制每个时间点的散点图
for time in green_light_times:
y_diff_value = data.loc[data['time'] == time, 'y_diff'].iloc[0]
```

```
    if time % 88 == 0:
    color = 'green'
    label = '一般切换点'
    if not general_added:
    plt.scatter(time, 0, color=color, marker='o', label=label)
    general_added = True
    else:
    plt.scatter(time, 0, color=color, marker='o')
    elif y_diff_value == -3.18:
    continue
    else:
    color = 'red'
    label = '异常切换点'
    if not exception_added:
    plt.scatter(time, 0, color=color, marker='o', label=label)
    exception_added = True
    else:
    plt.scatter(time, 0, color=color, marker='o')
    exception_times.append(time)

    plt.title('路口C1绿灯切换关键时间点图')
    plt.xlabel('时间')
    plt.ylabel(' ')
    plt.legend()
    plt.grid(False)
    plt.yticks([])
    plt.gca().set_facecolor('#ffe4e1')
    plt.show()

    print("异常切换点时间: ", exception_times)

    data = pd.read_csv('processed2_C1.csv')

    # 设置周期长度
    T = 88
```

```
109
110     # 选取时间段
111     k = 8
112     start = T * k - 88
113     end = T * k - 1
114     period_data = data[(data['time'] >= start) & (data['time'] <= end)]
115
116     plt.figure(figsize=(10, 5))
117
118     # 检查并绘制红灯时间点的红点
119     if 'red' in period_data.columns and not
            period_data[period_data['red'] == 1].empty:
120     plt.scatter(period_data[period_data['red'] == 1]['time'],
121     [1] * len(period_data[period_data['red'] == 1]),
122     color='red', label='红灯')
123     else:
124     print(f"在第{k}周期内没有红灯标记。")
125
126     # 检查并绘制绿灯时间点的绿点
127     if 'green' in period_data.columns and not
            period_data[period_data['green'] == 1].empty:
128     plt.scatter(period_data[period_data['green'] == 1]['time'],
129     [1] * len(period_data[period_data['green'] == 1]),
130     color='green', label='绿灯')
131     else:
132     print(f"在第{k}周期内没有绿灯标记。")
133
134     plt.title(f'第{k}周期红绿灯标记图')
135     plt.xlabel('时间')
136     plt.ylabel('红/绿灯标记')
137     plt.yticks([1], ['信号灯'])
138     plt.legend()
139     plt.xlim(start, end)
140     plt.show()
141
```

```
142 red_time_differences = []
143 green_time_differences = []
144
145 T = 88
146
147 for k in range(1, 82):
148 start = T * k - 88
149 end = T * k - 1
150 period_data = data[(data['time'] >= start) & (data['time'] <= end)]
151
152 # 寻找第一个红灯时间点
153 if not period_data[period_data['red'] == 1].empty:
154 first_red_time = period_data[period_data['red'] == 1]['time'].iloc[0]
155 red_time_difference = first_red_time - start
156 else:
157 red_time_difference = np.nan
158
159 # 寻找最后一个绿灯时间点
160 if not period_data[period_data['green'] == 1].empty:
161 last_green_time = period_data[period_data['green'] ==
    1]['time'].iloc[-1]
162 green_time_difference = last_green_time - start
163 else:
164 green_time_difference = np.nan
165
166 # 存储时间差
167 red_time_differences.append(red_time_difference)
168 green_time_differences.append(green_time_difference)
169
170 print("每个周期的第一个红灯时间点与周期开始时间的差：")
171 print(red_time_differences)
172 print("每个周期的最后一个绿灯时间点与周期开始时间的差：")
173 print(green_time_differences)
174
175 # 绘制红灯和绿灯时间差的分布图
```

```
176     plt.figure(figsize=(14, 7))
177     plt.bar(range(1, 82), red_time_differences, color='#FF6347',
            label='红灯时间差')
178     plt.bar(range(1, 82), green_time_differences, color='#32CD32',
            label='绿灯时间差', alpha=0.5)
179     plt.title('C1时间差分布图')
180     plt.xlabel('周期编号')
181     plt.ylabel('时间差（秒）')
182     plt.legend()
183     plt.grid(True)
184     plt.show()
185
186     # 填补缺失值
187     red_filled = fill_with_neighbors(pd.Series(red_time_differences))
188     green_filled = fill_with_neighbors(pd.Series(green_time_differences))
189
190     # 使用滑动窗口生成特征坐标
191     window_size = 10
192     points = []
193
194     for i in range(len(red_filled) - window_size + 1):
195     red_min = red_filled[i:i + window_size].min()
196     green_max = green_filled[i:i + window_size].max()
197     points.append([red_min, green_max])
198
199     points = np.array(points)
200
201     # 应用变点检测
202     model = rpt.Pelt(model="rbf")
203     model.fit(points)
204     result = model.predict(pen=3)
205     result = result[:-1]
206
207     # 取变化最明显的三个变点
208     top_change_points = sorted(result, key=lambda x:
```

```
        np.linalg.norm(points[x-1] - points[x]), reverse=True)[:3]

    # 绘制变点检测结果
    plt.figure(figsize=(14, 7))
    for i, feature in enumerate(points.T):
    plt.plot(feature, label=f'Feature {i+1}')
    for cp in top_change_points:
    plt.axvline(x=cp, color='red', linestyle='--', label=f'Change Point
        at {cp}')
    plt.title('Change Point Detection with Coordinate Features')
    plt.xlabel('Index')
    plt.ylabel('Feature Value')
    plt.legend()
    plt.show()

    # 定义周期长度
    cycle_length = T
    # 滑动窗口中心校正
    window_center_correction = window_size // 2

    # 计算每个变点对应的原始时间点
    original_time_points = [(cp + window_center_correction) *
        cycle_length for cp in top_change_points]

    # 打印每个变点对应的周期开始时间
    print("变点对应的周期开始时间：")
    for idx, original_time in enumerate(original_time_points):
    print(f"变点 {idx+1} 在时间 {original_time} 秒")
```

9. 问题四的 python 代码 1

```
    import matplotlib.pyplot as plt
    import numpy as np

    D1 = np.array([76, 218, 360, 502, 644, 786, 928, 1070, 1212, 1354,
        1496, 1638, 1780, 1922, 2064, 2111, 2206, 2348, 2482, 2624, 2766,
```

```
    2908, 2994, 3050, 3107, 3192, 3334, 3476, 3618, 3760, 3902, 4044,
    4186, 4328, 4470, 4612, 4754, 4896, 5038, 5180, 5322, 5464, 5606,
    5748, 5890, 6032, 6174, 6316, 6458, 6600, 6742, 6884, 7168])
D2 = np.array([503, 929, 1497, 2065, 2207, 2349, 2483, 2625, 2767,
    2909, 3193, 3335, 3477, 3619, 4045, 4187, 4329, 4471, 4613, 5465,
    5749, 5891, 6317, 6743, 6885, 7027])
D3 = np.array([114, 256, 398, 540, 824, 966, 1108, 1250, 1392, 1676,
    1960, 2102, 2244, 2524, 2666, 2808, 3092, 3234, 3376, 3660, 3802,
    4086, 4228, 4370, 4654, 4796, 4938, 5364, 5506, 5648, 5790, 5932,
    6074, 6216, 6358, 7068])
D4 = np.array([142, 284, 568, 710, 800, 852, 994, 1278, 1420, 1704,
    1846, 2130, 2272, 2414, 2556, 2698, 2982, 3124, 3266, 3408, 3550,
    3692, 3834, 3976, 4118, 4402, 4544, 4686, 5112, 5396, 5538, 5964,
    6106, 6390, 6816, 6940, 6958, 7100])
D5 = np.array([334, 476, 618, 902, 1044, 1186, 1470, 1754, 1896,
    2038, 2180, 2322, 2601, 3027, 3169, 3453, 3595, 3737, 4021, 4731,
    4873, 5015, 5441, 5725, 5867, 6009, 6293, 6435, 6577, 7003, 7145])
D6 = np.array([427, 853, 995, 1279, 1421, 1847, 2415, 2557, 2690,
    2699, 3115, 3125, 3462, 3551, 3693, 3977, 4119, 4261, 4829, 5255,
    5462, 5539, 6044, 6107, 6391, 6896, 6941])
D7 = np.array([76, 218, 360, 502, 644, 786, 928, 1070, 1212, 1354,
    1496, 1638, 1780, 1922, 2064, 2206, 2348, 2482, 2624, 2766, 2908,
    3050, 3192, 3334, 3476, 3618, 3902, 4044, 4186, 4328, 4612, 4896,
    5038, 5180, 5322, 5464, 5606, 5890, 6032, 6174, 6316, 6458, 6600,
    6742])
D8 = np.array([114, 256, 398, 540, 682, 824, 966, 1108, 1250, 1392,
    1676, 1818, 2244, 2386, 2524, 2666, 2808, 2950, 3092, 3234, 3376,
    3518, 3660, 3944, 4086, 4228, 4370, 4512, 4654, 4796, 4938, 5080,
    5364, 5506, 5648, 5932, 6216, 6358, 6500, 6642, 6926, 7068])
D9 = np.array([503, 645, 914, 929, 1302, 1355, 1419, 1497, 1545,
    1639, 1781, 1974, 2065, 2414, 2483, 2625, 2984, 3051, 3285, 3335,
    3419, 3420, 3477, 3981, 4045, 4187, 4329, 4526, 4613, 4850, 4897,
    4960, 5039, 5181, 5323, 5607, 6317, 6377, 6459, 6601, 6885, 6956,
    7027, 7097, 7098, 7169])
D10 = np.array([284, 426, 710, 994, 1136, 1278, 1420, 1562, 1704,
```

```
    1846, 1940, 1988, 2130, 2272, 2414, 2840, 2982, 3266, 3550, 3692,
    3834, 3976, 4260, 4402, 4544, 4686, 5112, 5254, 5396, 5538, 5680,
    5964, 6248, 6390, 6532, 6674, 6816, 7100])
D11 = np.array([143, 285, 427, 569, 1137, 1279, 1399, 1421, 1705,
    1941, 1989, 2189, 2273, 2342, 2415, 2481, 2557, 2699, 2841, 3125,
    3409, 3551, 3693, 3803, 3835, 3973, 3977, 4261, 4403, 4687, 4829,
    5113, 5397, 5539, 5965, 6249, 6391, 6533, 6675, 6748, 6817, 6959])
D12 = np.array([192, 334, 476, 618, 902, 1044, 1186, 1470, 1612,
    1754, 1896, 2038, 2180, 2601, 2885, 3027, 3169, 3311, 3453, 3595,
    3737, 3879, 4021, 4305, 4731, 4873, 5299, 5583, 5725, 5867, 6009,
    6151, 6293, 6577, 6861])

sequences = [D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12]

# 设置绘图
fig, ax = plt.subplots(figsize=(12, 8))
colors = plt.cm.nipy_spectral(np.linspace(0, 1, len(sequences)))

# 绘制每个序列
for i, seq in enumerate(sequences):
y = np.full_like(seq, fill_value=i) # 不同高度
ax.scatter(seq, y, color=colors[i], label=f'D{i+1}', s=65) # 点的大小

ax.axvline(x=2414, color='red', linestyle='--', label='周期变化处')

ax.set_xlabel('时间')
ax.set_yticks(range(len(sequences)))
ax.set_yticklabels([f'D{i+1}' for i in range(len(sequences))])
ax.set_ylim(-1, len(sequences))
ax.legend(loc='upper left', bbox_to_anchor=(1,1))

plt.tight_layout()
plt.show()
```

10. 问题四的 python 代码 2

```python
import pandas as pd
import numpy as np

data = pd.read_csv('向西行驶.csv')

data.sort_values(by=['vehicle_id', 'time'], inplace=True)

data['x_diff'] = data.groupby('vehicle_id')['x'].diff().fillna(0)
data['y_diff'] = data.groupby('vehicle_id')['y'].diff().fillna(0)
data['time_diff'] =
    data.groupby('vehicle_id')['time'].diff().fillna(1) # 防止除以0

# 计算速度
data['speed'] = np.sqrt(data['x_diff']**2 + data['y_diff']**2) /
    data['time_diff']

# 标注减速数据：如果当前速度小于前一时间点的速度，则认为是在减速
data['slowing_down'] = data.groupby('vehicle_id')['speed'].diff() < 0

# 标注停车数据：速度接近零
data['stopped'] = data['speed'] <= 1

# 标注红灯数据点
data['red'] = (
(data['stopped'] & ((data['y'] > 18.8) ))).astype(int)

# 标注绿灯通过路口的时间点
data['prev_x'] = data.groupby('vehicle_id')['x'].shift(1)
data['green'] = (data['x'] * (data['prev_x']) < 0).astype(int)

data.to_csv('processed2_D7.csv', index=False)

# 统计不同车辆ID的数量
unique_vehicle_ids = data['vehicle_id'].nunique()
print("不同的车辆ID数量：", unique_vehicle_ids)
```

```
    # 检测从停止到运动的转变
    data['prev_x_1'] = data.groupby('vehicle_id')['x'].shift(1)
    data['prev_x_2'] = data.groupby('vehicle_id')['x'].shift(2)
    data['prev_stopped_1'] =
        data.groupby('vehicle_id')['stopped'].shift(1)

    data['green_light'] = ((data['speed'] > 0) & (data['prev_stopped_1']
        == True) &
    (((data['prev_x_1'] == 18.84) & (data['prev_x_2'] == 18.84))))

    # 寻找并打印关键时间点
    green_light_times = data[data['green_light']]['time'].unique()
    print("绿灯切换的关键时间点: ", green_light_times)
```

11. 问题四的 python 代码 3

```
    import pandas as pd

    def read_and_process_csv(file_path):
    data = pd.read_csv(file_path)
    return data

    def get_quadrant(x, y):
    if x > 0 and y > 0:
    return "第一象限"
    elif x < 0 and y > 0:
    return "第二象限"
    elif x < 0 and y < 0:
    return "第三象限"
    elif x > 0 and y < 0:
    return "第四象限"
    else:
    return "坐标轴上"

    def analyze_movements(data):
```

```
    # 添加象限信息用于方向计算
    data['quadrant'] = data.apply(lambda row: get_quadrant(row['x'],
        row['y']), axis=1)

    # 聚合每个车辆的首尾象限
    quadrants =
        data.groupby('vehicle_id')['quadrant'].agg(first='first',
        last='last').reset_index()

    # 计算方向
    def get_direction(first, last):
    direction_rules = {
      ("第一象限", "第三象限"): "向南行驶",
      ("第三象限", "第一象限"): "向北行驶",
      ("第四象限", "第二象限"): "向西行驶",
      ("第二象限", "第四象限"): "向东行驶",
      ("第一象限", "第二象限"): "向南行驶再右转向西行驶",
      ("第二象限", "第一象限"): "向东行驶再左转向北行驶",
      ("第二象限", "第三象限"): "向东行驶再右转向南行驶",
      ("第三象限", "第二象限"): "向北行驶再左转向西行驶",
      ("第三象限", "第四象限"): "向北行驶再右转向东行驶",
      ("第四象限", "第三象限"): "向西行驶再左转向南行驶",
      ("第四象限", "第一象限"): "向西行驶再右转向北行驶",
      ("第一象限", "第四象限"): "向南行驶再左转向东行驶",
    }
    return direction_rules.get((first, last), "无法判断方向")

    # 计算每个车辆的方向
    quadrants['direction'] = quadrants.apply(lambda row:
        get_direction(row['first'], row['last']), axis=1)

    # 合并方向信息
    data = pd.merge(data, quadrants[['vehicle_id', 'direction']],
        on='vehicle_id')
```

```
    # 根据方向分组
    direction_groups = data.groupby('direction')
    return direction_groups

    def save_groups_to_csv(direction_groups):
    # 保存每个方向组的数据
    for direction, group_df in direction_groups:
    filename = f"{direction}.csv"
    group_df.drop(['quadrant'], axis=1).to_csv(filename, index=False)
    print(f"Saved '{filename}' for direction '{direction}' with
        {len(group_df)} records.")

    def main():
    file_path = "D.csv"
    data = read_and_process_csv(file_path)
    direction_groups = analyze_movements(data)
    save_groups_to_csv(direction_groups)

    if __name__ == '__main__':
    main()
```