基于 DBSCAN 和 Fourier 变换的交通信号灯周期估计研究

摘要

交通问题与人们的生活息息相关。探究行车轨迹与交通信号灯周期之间的关系、提升交通管理与导航准确性一直以来是科学界和决策者共同关心的问题。

针对问题一，为精确估计信号灯周期，本文首先计算以每个数据点为圆心、指定半径内的点数量，识别车辆密度最高的位置。本文假设这些高密度点最接近信号灯，据此得到这些范围内所有车辆的启动时间点，并分析相邻时间点之间的差值，估计信号灯的周期。然后考虑计算红灯持续时间。本文基于欧氏距离计算每辆车的速度曲线。在确定适宜速度阈值后，将速度低于阈值的时刻定义为红灯状态。本文进一步记录这些时刻及车辆的启动时间，并计算红灯持续时长。针对不同的数据集，本文考虑使用 **DBSCAN 模型**和 **K-MEANS 聚类**等分析方法，并探索这些聚类分析方法的可行性，最终采用 **DBSCAN 模型**对车辆的停止和启动时刻进行聚类，从而得到相应的红灯持续时间。

针对问题二，为探究车辆比例的影响，本文考虑对每个路口的数据分别随机提取 10%, 30%, 50%, 70%，模拟不同的样本覆盖率。重复分析过程，并对比各个子集的信号周期估计结果。为探究车流量的影响，本文采用 **DBSCAN 模型**分析车辆的停止时刻数据。并通过对每个聚类的结束时刻进行逐一增加的微调，或选取每个聚类簇的停车时间最早的车辆，以模拟高车流量和低车流量的情况。为探究定位误差的影响，本文采用一种模拟定位误差的方法，该方法基于**随机均匀分布**对车辆实际行车轨迹坐标进行修改，并发现 **KALMAN 滤波**对其有明显纠正效果。通过分析，可以得出样本车辆比例、车流量、定位误差因素对上述模型估计精度的影响。

针对问题三，为判断路口相应方向的信号灯周期在这段时间内的变化，并求出周期切换的时刻以及新旧周期参数，本文首先考虑利用**傅里叶变换**从历史轨迹数据中提取出信号灯的主要频率成分，以估算出信号周期长度。然后通过计算信号周期的最大峰值，确定红灯周期。为判断信号灯周期的变化，以**滑动窗口技术**，对路口的交通流量进行时段划分，监测每个时段的车辆通过次数。若某时段内的车辆通过次数超过阈值，则认为该周期改变。最后，为求出周期切换时间以及新周期，本文应用 **DBSCAN 模型**。

针对问题四，为估计路口所有方向信号灯的运行周期，本文首先分析车辆轨迹数据并对数据进行二维可视化，发现可以将该路口近似视为一个十字路口。在此基础上，本文定义了四种行驶方向（向东，向南，向西，向北）以及三种基于路口的行驶状态（左转，直走，右转）。本文考虑使用 **K-MEANS 聚类**将数据集分为十二类不同行驶状态的车辆并分别对其进行信号灯的周期分析。其次，本文计算以车辆为圆心的特定半径内的点的密度，识别出了车流密集区域。在密集区域内，本文进一步计算并排序了静止车辆的启动时间，对信号灯变换周期进行估计。最后本文使用**傅里叶变换**以及 **GEV 算法**计算红灯周期时间。信号灯的总周期与红灯周期时间的差值被估计为绿灯周期。

关键字：DBSCAN 模型　K-MEANS 聚类　傅里叶变换　滑动窗口　GEV 算法

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。由于许多信号灯未接入网络，无法直接从交通管理部门获取所有信号灯的数据，也不可能在所有路口安排人工读取信号灯周期信息。所以，该公司计划使用大量客户的行车轨迹数据估计交通信号灯的周期。请帮助该公司解决这一问题，完成以下任务。已知所有信号灯只有红、绿两种状态。

1.2 问题提出

问题一：若信号灯周期固定不变，且已知所有车辆的行车轨迹，建立模型，利用车辆行车轨迹数据估计信号灯的红绿周期。附件 1 中是 5 个不相关路口各自一个方向连续 1 小时内车辆的轨迹数据，尝试求出这些路口相应方向的信号灯周期，并按格式要求填入表 1。

问题二：实际上，只有部分用户使用该公司的产品，即只能获取部分样本车辆的行车轨迹。同时，受各种因素的影响，轨迹数据存在定位误差，误差大小未知。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。附件 2 中是另外 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，尝试求出这些路口相应方向的信号灯周期，按同样的格式要求填入表 2。

问题三：如果信号灯周期有可能发生变化，能否尽快检测出这种变化，以及变化后的新周期？附件 3 中是另外 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，判断这些路口相应方向的信号灯周期在这段时间内是否有变化，尝试求出周期切换的时刻，以及新旧周期参数，按格式要求填入表 3，并指明识别出周期变化所需的时间和条件。

问题四：附件 4 是某路口连续 2 小时内所有方向样本车辆的轨迹数据，请尝试识别出该路口信号灯的周期。

二、 问题分析

2.1 问题一分析

本问题目的为估计五个不相关路口固定周期信号灯的运行周期。通过分析一小时内提供的车辆轨迹数据，本文首先对每个路口的 X 轴和 Y 轴数据进行了综合分析，以确定各路口的方向和车道数，并对数据进行了可视化处理以辅助分析。

为精确估计信号灯周期，本文基于欧氏距离计算了每辆车的速度曲线。在广泛查阅相关文献 [2] 后，确定了适宜的速度阈值 [4]，并将速度下降至此阈值以下的时刻定义

为红灯状态。本文进一步记录了这些时刻及车辆的重新启动时间，并计算了红灯持续时长。针对不同的数据集，本文通过实验比较了 DBSCAN 聚类 [1] 和 K-MEANS 聚类等聚类分析方法的有效性，最终采用 DBSCAN 算法对车辆的停止和启动时刻进行聚类，从而得到相应的红灯持续时间。

此外，本文通过计算以每个数据点为圆心、指定半径内包含的点数量，识别了车辆密度最高的位置。本文假设这些高密度点最接近信号灯，据此计算了这些位置上速度为零的所有车辆的启动时间，并进行了排序。通过分析这些时间点之间的差值，估计了信号灯的周期。信号灯的总周期与红灯持续时间的差值被估计为绿灯周期。

2.2 问题二分析

本问题要求讨论样本车辆比例、车流量、定位误差等因素对上述模型估计精度的影响。本文通过以下三个角度进行建模：

- **角度一：**对每个路口的数据分别随机提取 10%, 30%, 50%, 70%，模拟不同的样本覆盖率。重复上述分析过程，并对比各个子集的信号周期估计结果；
- **角度二：**采用了 DBSCAN 聚类算法来分析车辆的停止时刻数据。进一步通过对每个聚类的结束时刻进行逐一增加的微调、选取每个聚类簇的停止时间最长的车辆以模拟高车流量和低车流量的情况。
- **角度三：**采用了一种模拟定位误差的方法，该方法基于随机均匀分布对车辆实际行车轨迹坐标进行修改，根据总周期的估计结果与原数据比较。通过分析，可以得出样本车辆比例、车流量、定位误差因素对上述模型估计精度的影响。

2.3 问题三分析

本问题要求判断路口相应方向的信号灯周期在这段时间内是否有变化，并求出周期切换的时刻以及新旧周期参数。

本文利用傅里叶变换从历史轨迹数据中提取出路口信号灯的主要频率成分，从而估算出信号周期的平均长度。随后，通过计算信号周期的最大峰值，确定最长红灯周期时间。进一步，本文采用了滑动窗口技术，对路口的交通流量进行时段划分，监测每个时段的车辆通过次数。本文设定了阈值判断标准：如果某个时段内的车辆通过次数与平均值的差的绝对值大于或等于 1，则将该时段标记为周期异常时段。最后，为了对这些异常时段进行深入分析，本文应用了 DBSCAN 聚类算法。通过聚类，能够确定变化后周期的长度，并找出周期切换的具体时刻。

2.4 问题四分析

本问题要求估计某路口所有方向信号灯的运行周期。通过分析两小时内提供的车辆轨迹数据，本文首先对数据进行二维可视化，发现可以将该路口近似视为一个十字路口。在此基础上，本文定义了四种行驶方向（向东，向南，向西，向北）以及三种基于路口的行驶状态（左转，直走，右转）。

为估计路口所有方向信号灯的运行周期，本文首先考虑使用 K-MEANS 聚类方法将数据集分为十二类不同行驶状态的车辆并分别对其进行信号灯的周期分析。其次，本文计算以车辆为圆心的特定半径内的点的密度，识别出了车流密集区域。在这些区域内，本文进一步计算并排序了静止车辆的启动时间，对信号灯变换周期进行估计。最后本文使用傅里叶变化以及广义极值（GEV）模型计算出红灯持续时间。信号灯的总周期与红灯持续时间的差值被估计为绿灯周期。

下图为论文整体流程分析图1：

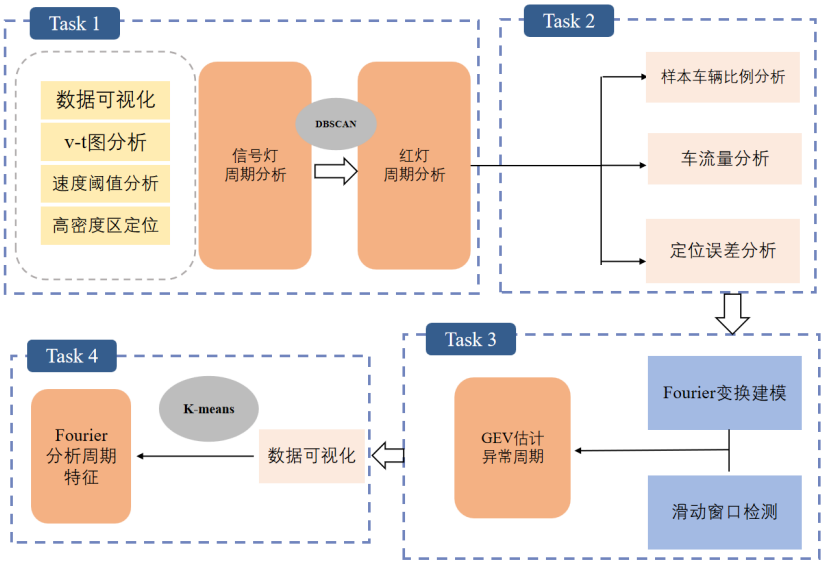


图 1　流程分析图

三、 模型的假设

- 假设给出的数据均为真实数据，真实有效；
- 假设对于一些较为异常的数据的出现具有一定的合理性；
- 假设车辆在红灯亮起时会减速至阈值以下并停止，而在绿灯亮起时重新加速；
- 假设分析计算的高密度点是最接近于信号灯的点。

四、 符号说明

符号	意义
$\vec{d}$	单位时间位移（m）
v	瞬时速率（m/s）
f	角频率（Hz）
V(f)	频域内的信号
T	周期（s）
$p(\tau)$	时间窗口内的单个时间点（s）
$f_{\max}$	频率的最大值（Hz）
P_t	时间窗口 t 到 t+Δt 内的累计数量

五、 模型的建立与求解

5.1 问题 1 的模型建立与求解

问题一要求通过一小时内所有车辆的行车数据，估计五个不相关路口固定周期信号灯的运行周期。本文通过以下五个步骤进行建模：

- **步骤一：**对每个路口的 X 轴和 Y 轴数据进行了综合分析，并对数据进行了可视化处理以辅助分析；
- **步骤二：**基于欧氏距离计算了每辆车的速度曲线，确定了适宜的速度阈值，并将速度下降至此阈值以下的时刻定义为红灯状态；
- **步骤三：**采用 DBSCAN 算法对车辆的停止和启动时刻进行聚类，以确定一小时内车辆的主要停车和启动时刻；
- **步骤四：**通过计算以每个数据点为圆心、指定半径内包含的点数量，识别了车辆密度最高的位置；
- **步骤五：**计算了这些位置上速度为零的所有车辆的启动时间并进行排序。通过分析这些时间点之间的差值，估计了信号灯的周期和绿灯周期；

5.1.1 数据预处理

在对附件数据检查与浏览后，本团队发现原始数据基本不存在异常。本文采用 Q-Q 图对附件 A2 给定数据中的 x 列和 y 列分析，它通过将两个概率分布的分位数进行比较的方式来图形化展示数据分布特性。如果数据点大致在一条直线上，这表明数据分布与正态分布相符；如果数据点显著偏离直线，则表明数据分布不是正态的。通过分析附件

A2 的 Q-Q 图2，发现数据点在两端偏离直线，显示了尾部的厚度超出了正态分布的预期，即发生了较为严重的重尾分布。

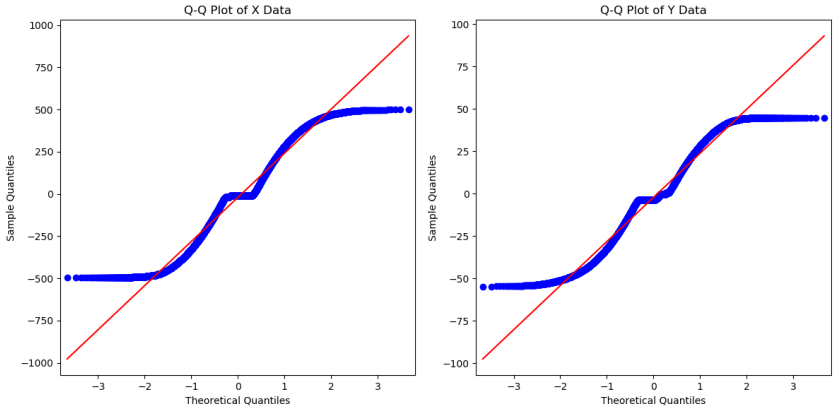


图 2　关于 x 和 y 的 Q-Q 图

对于非正态分布，本文使用箱型图判定数据异常值。箱线图通过绘制数据的四分位数和离群值来检测异常值。数据点如果落在箱线图上下四分位距的 1.5 倍之外，则被认为是离群值。通过分析附件 A2 的箱型图3，发现这些数据没有明显的异常值，因为没有点落在箱须外面。

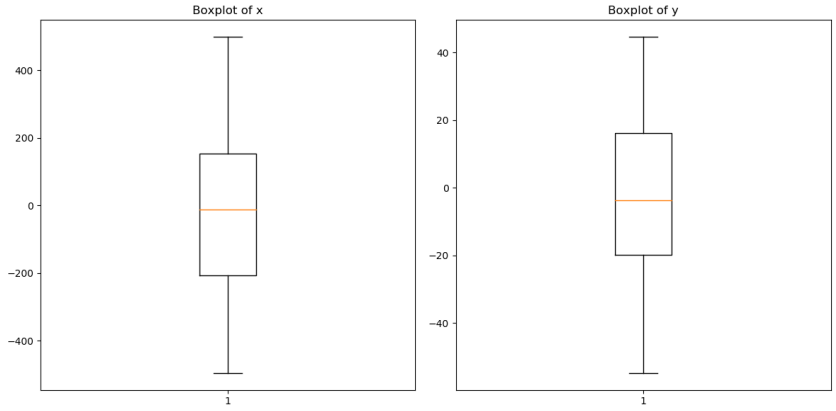


图 3　关于 x 和 y 的箱型图

另外，为了对路口及车道有更加清晰的认识，本文对数据进行了可视化处理。通过附件每个文件中的每辆车的轨迹进行二维可视化，结果如图4所示：

5.1.2 车辆速度曲线计算以及速度阈值的确定

在本数据集的处理过程中，对于每一辆车辆而言，其在连续时间点上的空间位置坐标被纪录为 x 轴与 y 轴上的有序对。为了计算车辆在相继时间点间的即时速率，我们采用了微分几何中的欧几里得距离公式来量化位移的幅度。具体实现上，通过对每一辆车辆标识符 vehicle_id 进行分组，保证了速率的计算是在每辆车辆的轨迹上单独进行。

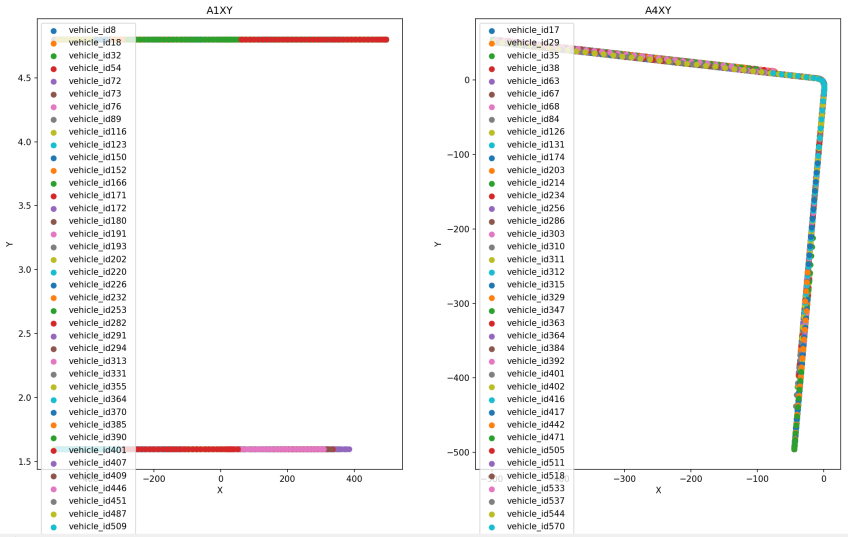


图 4　附件 A1、A4 的路径

车辆在时间 Δt=t−(t−1) 内的位移 $\vec{d}$ 可以计算为：

$$\vec{d} = (x_t - x_{t-1}, y_t - y_{t-1}) \tag{1}$$

车辆的即时速率 v 的大小，可以通过位移的欧几里得范数来计算：

$$v = \frac{\|\vec{d}\|}{\Delta t} = \frac{\sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2}}{\Delta t} \tag{2}$$

在这种情况下，由于 Δt 被设定为 1 秒，因此速度的计算简化为：

$$v = \sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2} \tag{3}$$

该方法实质上基于了经典物理学中的速度定义，即速度矢量的模长等于位移矢量的模长除以时间间隔。在此处，时间间隔假定为恒定值（即每个时间点之间的时间差为一秒），因而每一次的差分计算即可视为单位时间内的位移变化量。通过将该位移变化量赋值于新创建的 speed 列，本文完成了对数据集中每辆车每秒速度的精确计算 [5]。

基于这样的方法，计算了每辆车的速度曲线，部分数据如下图5所示。

本文提出了一种基于速度阈值的识别方法，即通过设定一个速度阈值，以便将车辆速度的降低至该阈值以下的时刻，判定为信号灯处于红灯状态的概率较高的时间点。

本文计算了每辆车速度的平均数、四分位数、最小值、最大值、方差等统计指标，数据如表1所示。

通过对车辆速度数据的统计分析，以及考虑速度分布的中位数、四分位数等统计指标，并查阅相关文献 [3] 后，本文确立了适当的速度阈值为 1。在设定了阈值后，本研究对每辆车的速度时间序列数据进行了扫描，用红色框标记出所有速度降至阈值以下的时间段。部分数据如图6所示：

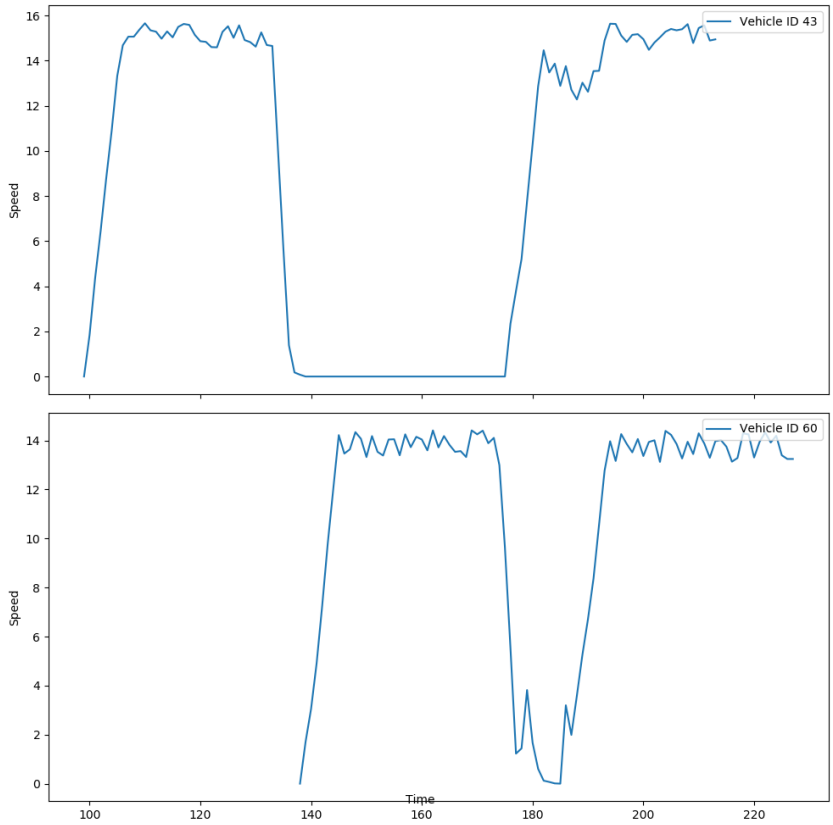


图 5　速度时间曲线部分处理结果

表 1　不同路口速度统计摘要

统计指标	路口 A1	路口 A2	路口 A3	路口 A4	路口 A5
均值（Mean）	8.758318	9.558914	8.612692	8.804271	8.702008
标准差（STD）	5.871468	5.550282	5.969917	5.972441	5.809334
最小值（Min）	0.000000	0.000000	0.000000	0.000000	0.000000
25% 分位数	4.210212	4.803793	4.523595	4.304453	3.943451
50% 分位数（中位数）	11.635000	12.079752	11.860000	11.858853	11.617061
75% 分位数	13.320000	13.416087	13.300000	13.536488	13.268323
最大值（Max）	17.530000	16.783292	17.219942	18.322800	16.783292

5.1.3 聚类分析停止和启动时刻

在交通流分析中，正确识别车辆的停止和启动时刻对于估计交通信号灯的周期尤为关键。为了实现这一目标，本文采用了密度基准的空间聚类应用与噪声（DBSCAN, Density-Based Spatial Clustering of Applications with Noise）算法。DBSCAN 算法的选择基于其独特的优势：能够有效处理任意形状的空间聚类，并能在聚类过程中识别并剔除

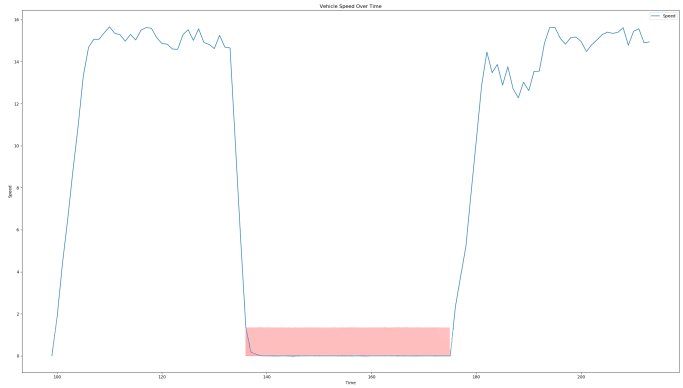


图 6　根据车辆速度判定红灯情况

噪声点。

本文首先导入了车辆的停止时刻数据，该数据包含每个观测周期内车辆的开始停止和结束停止时刻。这些时刻是通过检测车辆速度降至预先设定阈值以下的时间点来确定的。为了进行有效的聚类分析，我们选择了 start_time 和 end_time 这两列数据作为聚类特征，这反映了每个停车事件的时间范围。本文将 DBSCAN 算法的关键参数设置为：邻域半径（eps）为 50 秒和最小样本数（min_samples）为 1。

通过 DBSCAN 算法，本文成功地将停止时刻数据划分为多个聚类，每个聚类代表一组密集的停止和启动活动，可能对应于一个完整的信号灯周期。对于每个聚类，本文计算了从最早的开始时刻到最晚的结束时刻的总持续时间，以估计信号灯的周期时长。部分聚类结果如表2所示：

表 2　聚类结果摘要

聚类编号	车辆 ID	开始时间	结束时间	持续时间	周期时长
0	43, 46, 60	136	185	40, 34, 6	49 秒
1	74	211	263	53	52 秒
2	119, 124	314	351	38, 20	37 秒
3	190	473	527	55	54 秒
...	...	...	...	...	...
24	1384, 1386, 1401, 1405	3203	3266	53, 43, 11, 7	63 秒
...	...	...	...	...	...

在对 DBSCAN 与不同 k 值（3, 5, 7, 10）的 K-MEANS 算法的轮廓系数进行对比后，如表3所示. 结果一致表明，DBSCAN 的聚类效果显著优于 K-MEANS。因此，结合轮廓

系数的定量分析和本研究的具体情景，本文选择采用 DBSCAN 聚类算法.

表 3　聚类效果的轮廓系数比较

	k=3	k=5	k=7	k=10	DBSCAN
轮廓系数	0.6133	0.5891	0.549	0.597	0.8398

5.1.4 计算车辆密度最高的位置

- **定义搜索参数：**设置搜索圆的半径为 0.3 米，这个半径可以根据具体的道路和交通条件调整，以适应不同的分析需求；
- **迭代搜索密集中心：**通过迭代过程寻找最多车辆集中的位置。在每次迭代中，使用每个点作为圆心，计算该圆内的车辆数量。通过 KDTree 的 query_radius 方法实现，这种方法能够快速返回指定半径内的点数；
- **选取和更新：**在每次迭代中，记录找到的车辆数最多的圆心位置，并从数据集中移除该圆内的所有点。这一步骤是为了在后续的迭代中不再考虑已经标记为高密度区域的点。

通过这种方法，分析附件数据后，我们能够确定车辆最密集的地点。附件 A4 部分结果如图7所示：

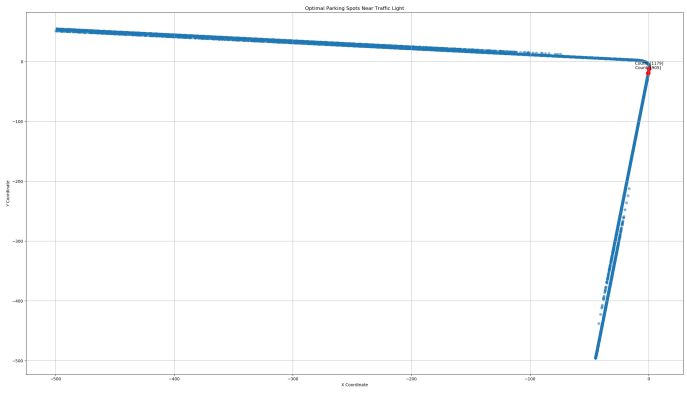


图 7　判定车辆密集点坐标结果

5.1.5 计算位置上速度为零车辆启动时间并进行排序

本文计算了每辆车在每一秒是否位于这两个圆形区域内，并创建了相应的布尔指标。这些指标帮助我们准确捕捉到车辆在这两个关键区域的停车事件。通过筛选每辆车

最后一次在任一圆内的时间点，可以得到每辆车的关键停车时刻。这些时间点非常重要，因为它们可能对应于交通信号灯从绿灯变为红灯的转换。进一步地，对这些时间点进行了排序，并计算了相邻两次停车时间的时间差，这些时间差反映了信号灯周期的长度。

为了提高分析的准确性，本文只关注那些速度接近零并且相邻时间差大于 10 秒的记录，从而过滤掉了那些频繁启停或因交通拥堵而频繁变化的数据点。通过这一筛选机制，确保了分析结果的可靠性。最后，本文计算了这些时间差的众数（最常出现的周期长度），认为这可能是该路口信号灯的典型周期。

最终实验结果如表4所示：

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）	71.7	57.43	78.42	70.97	61.69
绿灯时长（秒）	33.3	30.57	26.58	17.03	26.31

5.2 问题 2 的模型建立与求解

问题二要求讨论样本车辆比例、车流量、定位误差等因素对上述模型估计精度的影响。本文通过以下三个部分进行建模：

- **部分一：**对每个路口的数据分别随机提取 $10\%, 30\%, 50\%, 70\%$，模拟不同的样本覆盖率。重复上述分析过程，并对比各个子集的信号周期估计结果；
- **部分二：**采用了 DBSCAN 聚类算法来分析车辆的停止时刻数据。进一步对每个聚类的结束时刻进行了逐一增加的微调后再次聚类，再进行前后聚类的结果对比；
- **部分三：**采用了一种模拟定位误差的方法，该方法基于随机均匀分布对车辆实际行车轨迹坐标进行修改，根据总周期的估计结果与原数据比较。

5.2.1 车辆比例对模型估计精度的影响

本文进一步探讨了车辆比例对模型估计精度的影响。为此，本文不仅分析了全部数据，还通过抽取数据的不同比例（$10\%, 30\%, 50\%, 70\%$）来模拟不同的样本覆盖率。对于每个数据子集，本文重复上述分析过程，并对比各个子集的信号周期估计结果。结果如5所示：

从表中可以得到如下结果：

车辆比例较大的鲁棒性：部分路口（B1、B2、B4 和 B5）显示了在不同的样本车辆比例下，周期估计的稳定性，这意味着模型在这些路口即使数据缺失也能准确估计信号

表 5　不同比例抽样样本对模型结果的影响

抽样样本比例	B1	B2	B3	B4	B5
10%	126	116	-	-	228
30%	105	116	-	210	116
50%	105	116	88	210	116
70%	105	116	88	105	116
100%	105	116	88	105	116

灯周期，特别地，70% 的数据在五个路口的数据集中均不会对周期计算结果产生影响。

车辆比例较小的局限性：部分路口出现了抽样样本比例在 10%-30% 区间内时出现因数据过少，无法得到确切的数值的问题，以及数据过少，导致判定周期加倍或出错的问题。

5.2.2 车流量对模型估计精度的影响

为了探究车流量对信号灯周期估计精度的影响，本文采用了 DBSCAN 聚类算法来分析车辆的停止时刻数据。本文进一步对每个聚类的结束时刻进行了逐一增加的微调。这种调整策略反映了在高车流量环境下，每辆车的停止时间可能因后续车辆的堵塞而延长。本文对每个聚类内的车辆，按其等待时间进行排序，并对每个聚类簇保留等待时间最长的车辆，以反映低车流量环境。调整策略如下：

- 对原始数据进行 DBSCAN 聚类；
- 对于高车流量的影响分析
  - 对每个聚类内的车辆，按其 start_time 进行排序；
  - 对于排序后的每个数据点，其 end_time 被递增地增加，第一个数据点增加 1 秒，第二个增加 2 秒，依此类推；
  - 按照调整后的样本重新进行 DBSCAN 聚类。
- 对于低车流量的影响分析
  - 对每个聚类内的车辆，按其等待时间进行排序；
  - 每个聚类簇只保留 start_time 最小的的车辆；
  - 按照调整后的样本重新进行 DBSCAN 聚类。

对附件 B2 数据集按照策略模拟修改后，和修改前部分比较结果如表所示：

将模拟高车流量结果、低车流量结果与附件数据预测结果对比，如表7所示。

表 6　高车流量：B2 部分数据对比

(a) 处理后数据聚类结果				(b) 处理前数据聚类结果			
车号	启动时刻	停止时长	聚类情况	车号	启动时刻	停止时长	聚类情况
17	87	7	0	17	86	6	0
70	204	3	1	70	203	2	1
157	435	37	2	157	434	36	2
188	480	8	3	188	479	7	3
188	489	7	3	188	487	5	3

表 7　车流量与预测红灯周期关系表

	低车流量	正常车流量	高车流量
预测红灯周期	82	87	92

实验结果证明，车流量对模型聚类情况虽并无影响，但车流量的增加和减少可能会导致停车时间的延长以及缩短，导致红灯周期判定发现变化。这对于信号灯周期的估计精度有显著影响。

5.2.3 定位误差对模型估计精度的影响

为了深入分析定位误差对信号灯周期估计精度的影响，本研究采用了一种模拟定位误差的方法，该方法基于随机均匀分布对车辆实际行车轨迹坐标进行修改。模拟的误差范围定为原始坐标的 ±5%，即每个坐标点都被随机调整其值，在原始值的基础上增减最多 5%。

为了全面评估定位误差对信号灯周期估计精度的影响，本文设计了四个实验。每个实验分别模拟了不同的误差情形，以揭示定位误差在不同方向上的影响。具体实验设计如下：

- 仅对 X 坐标修改；
- 仅对 Y 坐标修改；
- 对 X 和 Y 坐标同时修改；
- 对 X 和 Y 坐标同时修改并利用 KALMAN 滤波处理。

通过实验，得到结果如表8所示：

从总周期的减少可以看出，定位误差尤其是 Y 方向或者 X 和 Y 方向的组合误差对信号灯周期估计有显著影响。对于红灯周期，X 方向的误差并没有影响周期估计，而

周期类型	X 扰动	Y 扰动	X-Y 扰动	正常数据	KALMAN 处理 X-Y 扰动
总周期	111	66	66	116	106
红灯周期	83	46	57	83	76
绿灯周期	28	20	11	33	30

表 8　定位误差对信号灯周期估计影响的实验结果

Y 方向和 X-Y 方向的组合误差则显著降低了周期估计。这可能表明车辆在红灯期间的停止状态更容易受到 Y 方向定位误差的影响。绿灯周期估计在 X-Y 扰动时受影响最大，这可能是因为车辆在绿灯期间的移动更加频繁和不规则，因此定位误差的累积效应在绿灯周期的估计中更为明显。

另外，本文发现 KALMAN 滤波对有定位误差的数据处理后，有明显的矫正效果，可以较为明显地减小误差。

本文求出这些路口相应方向的信号灯周期，最终实验结果如表9所示：

表 9　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）	77.52	90.01	71.2	76.04	92.23
绿灯时长（秒）	27.8	25.99	16.8	11.96	24.77

本文的实验分析揭示了车辆比例、车流量以及定位误差对于结果的显著影响，表明在进行现实交通分析时，这三个变量是不可忽视的关键因素。在实验中，应该考虑通过加权的方式消除这三点因素对实验误差的影响。

5.3 问题 3 的模型建立与求解

问题三要求判断路口相应方向的信号灯周期在这段时间内是否有变化，并求出周期切换的时刻以及新旧周期参数。

5.3.1 傅里叶变换建模

为了分析速度数据中的周期性特征，本文提出了一种新颖的方法来识别交通信号控制中的周期性变化。首先，利用傅里叶变换从历史轨迹数据中提取出路口信号灯的主要

频率成分，从而估算出信号周期的平均长度。随后，通过计算信号周期的最大峰值，能够确定最长红灯周期时间。

通过这种方法，可以将时间序列从时域转换到频域，从而识别出影响交通流的主要频率成分。变换后，本文仅考虑正频率部分，因为负频率在物理意义上通常不被考虑。对速度数据 v(t) 应用傅里叶变换，得到频域表示：

$$V(f) = \int_{-\infty}^{\infty} v(t) e^{-i2\pi ft}\, dt \tag{4}$$

在频谱图中，本文观察到的最大峰值对应的频率即为主要频率成分，这反映了交通信号周期的主要波动。通过计算该频率的倒数，可以得到交通信号的主要周期，这是评估信号调整和优化交通流的关键参数。一旦确定了主要频率成分，信号周期 T 可以通过下式计算：

$$T = \frac{1}{f_{\max}} \tag{5}$$

这个周期 T 反映了交通信号的基本循环时间。

本文对每一个路口数据进行傅里叶变换，得到频率谱如图8所示：

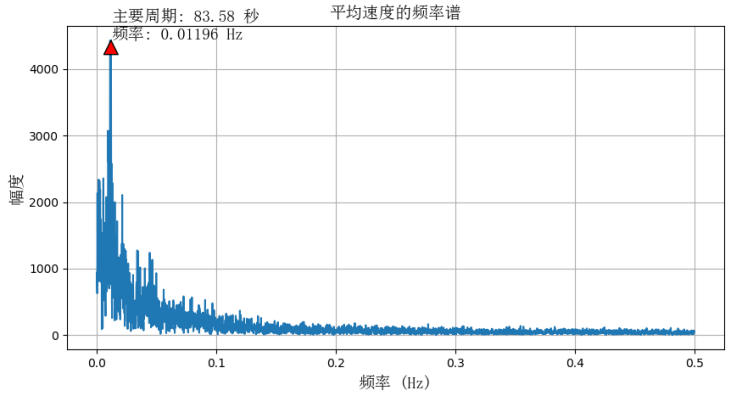


图 8　路口 C1 傅里叶变换结果

进一步地，本文利用速度阈值定义车辆的停止状态，并识别出每次停止的开始和结束时间点。通过这些时间点，计算了每个红灯周期的持续时间，这为理解信号灯的具体调控提供了直接证据。红灯持续时间的分析可以使用基本的统计公式，例如平均时间 μ、最小时间 min 和最大时间 max 计算如下：

$$\mu = \frac{1}{n}\sum_{i=1}^{n} d_i \quad \min = \min(d_i), \quad \max = \max(d_i) \tag{6}$$

其中，d_i 是第 i 次红灯持续时间的观测值，n 是观测值的总数。

最终得到结果如表10所示：

	C1	C2	C3	C4	C5	C6
周期长度	83.58	103	102.12	101.73	83.58	115.2
最长红灯周期	56	67	79	77	56	78

表 10　周期长度及最长红灯周期

5.3.2 滑动窗口监测

进一步，本文采用了滑动窗口技术，对路口的交通流量进行时段划分，监测每个时段的车辆通过次数。对于每个窗口，本文计算该时段内的车辆通过次数 P_t。即：设 P_t 为时段 t 的车辆通过次数，计算方法为：

$$P_t = \sum_{\tau=t}^{t+\Delta t} p(\tau) \tag{7}$$

其中 $p(\tau)$ 是时刻 τ 的车辆通过情况。平均通过次数 $\bar{P}$ 可以通过以下公式计算：

$$\bar{P} = \frac{1}{T} \sum_{t=1}^{T} P_t \tag{8}$$

通过与平均通过次数进行对比，本文设定了阈值判断标准：如果某个时段内的车辆通过次数与平均值的差的绝对值大于或等于 1，即：

$$\left| P_t - \bar{P} \right| \geq 1 \tag{9}$$

则将该时段标记为周期异常时段。

5.3.3 GEV 估计异常周期长度

为了对这些异常时段进行深入分析，找出异常周期的最大值，本文应用了 DBSCAN 聚类算法。通过聚类，能够确定变化后的周期长度分布。然而，不同的周期长度分布可能对真实的异常周期的预测有不同的影响。本文使用应用 GEV 模型分析异常时段的具体信号灯周期。

首先，本文将通过 GEV 模型来拟合车辆的红灯等待时长数据。拟合这个模型可以帮助了解数据的分布特性，尤其是极端值的分布。GEV 模型的拟合公式如公式10所示：

$$GEV(x;\mu,\sigma,\xi) = \exp\left(-\left[1 + \xi \left(\frac{x-\mu}{\sigma} \right) \right]^{-1/\xi} \right), \quad \text{for } 1+\xi(x-\mu)/\sigma > 0 \tag{10}$$

其中，μ 是位置参数，σ 是尺度参数，ξ 是形状参数。C3 路口拟合曲线如图9所示。

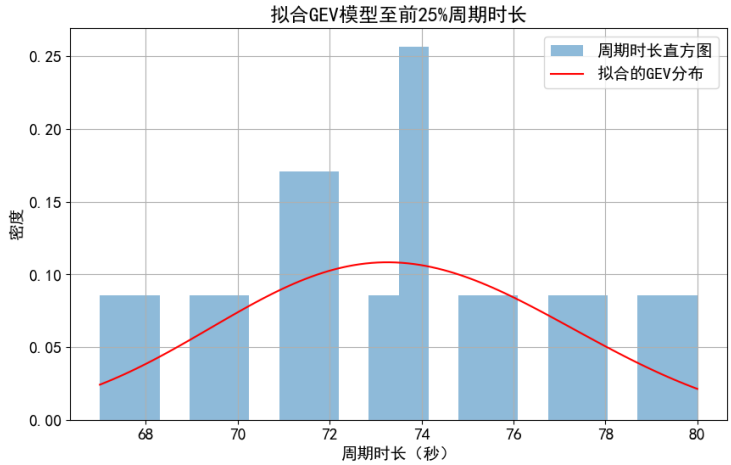


图 9　路口 C3 GEV 拟合结果

其次，本文计算 95% 置信区间上限。对于红灯时长的最大可能值，本文关注的是数据的上尾，因此使用 GEV 分布的上尾来估计这个上限。具体地，本文使用 GEV 分布的分位数函数11计算 95% 置信区间上限:

$$q(p) = \mu + \frac{\sigma}{\xi}\left[(-\log(1-p))^{-\xi} - 1\right] \tag{11}$$

然后，本文比较 95% 置信区间上限和已观测到的最大红灯时长。如果上限大于已观测到的最大值，则认为这两者之间的差额 p 是需要在观测最大值上需要添加的额外时长，以便更接近真实的最大红灯时长。确定额外的时长值 p 的公式如公式12所示:

$$p = \max(0, q(0.95) - x_{\max}) \tag{12}$$

最后，本文将这个额外时长 p 加到观测到的最大红灯时长上，得到一个修正后的估计值。

综上所述，第三问结果如表13所示。

表 11　各路口不同周期的红灯和绿灯时长

路口	C1	C2	C3	C4	C5	C6
周期 1 红灯时长（秒）	56	63	81.45	82.33	52	77.44
周期 1 绿灯时长（秒）	32	25	23.55	22.67	36	27.56
周期切换时刻	835	3083	945	1847	2068	2239
周期 2 红灯时长（秒）	46	66	81.45	74	43	70
周期 2 绿灯时长（秒）	42	22	25.55	31	45	35

续下页

表 11 – 续上页

路口	C1	C2	C3	C4	C5	C6
周期切换时刻	2067	-	1363	2272	4354	4451
周期 3 红灯时长（秒）	52	-	81.45	70	55	68.03
周期 3 绿灯时长（秒）	36	-	23.55	35	33	36.97
周期切换时刻	3565	-	2011	3736	5498	6337
周期 4 红灯时长（秒）	46	-	72	76	49	67
周期 4 绿灯时长（秒）	42	-	33	29	39	38
周期切换时刻	-	-	3259	4888	-	6958
周期 5 红灯时长（秒）	-	-	80	79	-	76
周期 5 绿灯时长（秒）	-	-	25	26	-	29
周期切换时刻	-	-	4001	-	-	-
周期 6 红灯时长（秒）	-	-	71	-	-	-
周期 6 绿灯时长（秒）	-	-	34	-	-	-
周期切换时刻	-	-	6199	-	-	-
周期 7 红灯时长（秒）	-	-	77	-	-	-
周期 7 绿灯时长（秒）	-	-	28	-	-	-
周期切换时刻	-	-	6835	-	-	-
周期 8 红灯时长（秒）	-	-	74	-	-	-
周期 8 绿灯时长（秒）	-	-	31	-	-	-

5.4 问题 4 的模型建立与求解

附件 4 是某路口连续 2 小时内所有方向样本车辆的轨迹数据，请尝试识别出该路口信号灯的周期。

本研究首先对附件四提供的每辆车的轨迹数据进行了二维可视化，通过对车辆在路口的 x 和 y 坐标进行描绘，明确了路口的布局，发现近似是一个十字路口。可视化结果如图10所示。

在此基础上，本文定义了十二种车辆通行模式，包括直行和转弯，如表12所示.

将分析结果用于 K-MEANS 聚类分析，成功将不同行驶模式的车辆区分开来。部分分离情况如图11a11b所示。

接着，本文聚焦于单个模式的深入分析。通过计算以车辆为圆心的特定半径内的点的密度，识别出了车流密集区域。在这些区域内，本文进一步计算并排序了静止车辆的

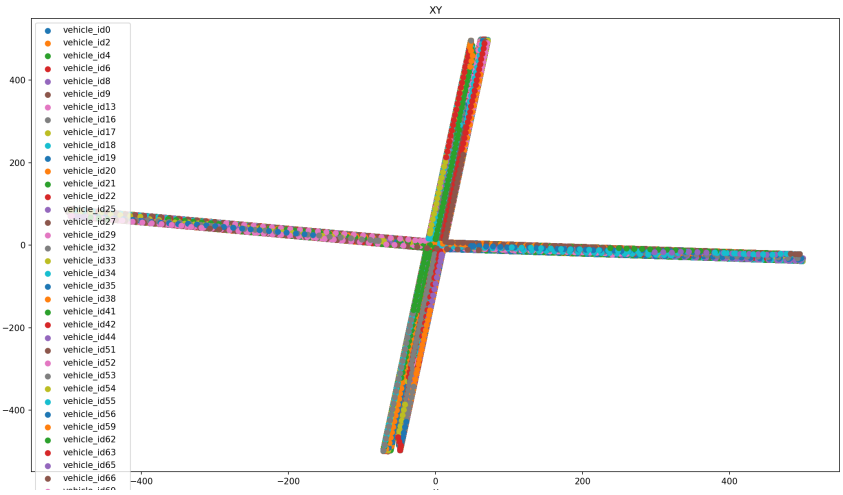
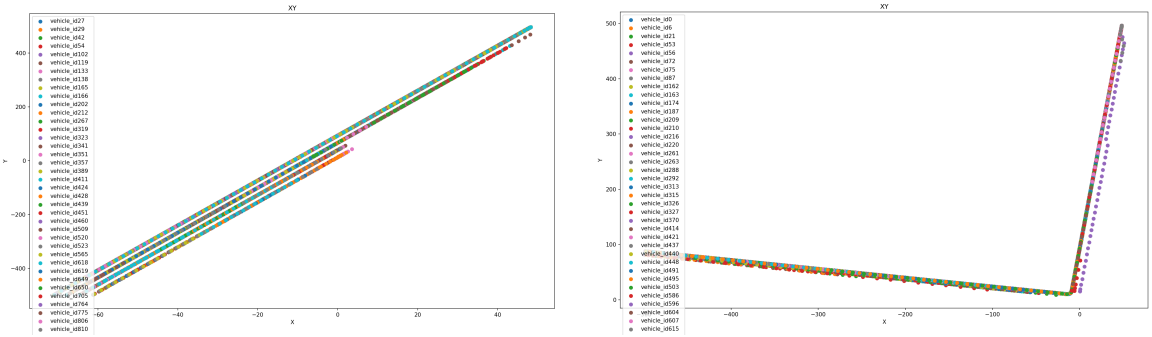


图 10　路口 D 可视化结果

表 12　各方向行走情况对应表

行走情况	向北	向南	向西	向东
左转	向北左转	向南左转	向西左转	向东左转
直行	向北直行	向南直行	向西直行	向东直行
右转	向北右转	向南右转	向西右转	向东右转

启动时间，从而揭示了信号灯变换周期的初步估计。最后，为了更精确地确定红灯周期，本文采用了傅里叶变换来分析周期性特征，并使用广义极值 (GEV) 模型对红灯持续时间进行了统计建模和拟合，得出了信号灯周期的详细分析结果如表13所示。



(a) 情况 1　　**(b) 情况 2**

图 11　十字路口分离部分情况

表 13　信号灯周期时间表

情况	向东左转	向东右转	向东直行	向北左转	向北右转	向北直行
红灯周期 (s)	123.22	97	102	120	93	94
总周期 (s)	142	142	142	142	142	142
情况	向南左转	向南右转	向南直行	向西左转	向西右转	向西直行
红灯周期 (s)	126	91	97	122	97	103.33
总周期 (s)	142	142	142	142	142	142

六、 模型评价和推广

6.1 模型的评价

6.1.1 模型的优点

对于问题一，本文基于欧氏距离计算了每辆车的速度曲线，在广泛查阅相关文献后，确定了适宜的速度阈值，并将速度下降至此阈值以下的时刻定义为红灯状态。本文进一步记录了这些时刻及车辆的重新启动时间，并计算了红灯持续时长。针对不同的数据集，本文通过实验比较了 DBSCAN 聚类和 K-MEANS 聚类等聚类分析方法的有效性，最终采用 DBSCAN 算法对车辆的停止和启动时刻进行聚类，以确定一小时内车辆的主要停车和启动时刻。此外，本文通过计算以每个数据点为圆心、指定半径内包含的点数量，识别了车辆密度最高的位置。通过分析这些时间点之间的差值，估计了信号灯的周期。信号灯的总周期与红灯持续时间的差值被估计为绿灯周期。

对于问题二，本文对每个路口的数据分别随机提取 10%, 30%, 50%, 70%，模拟不同的样本覆盖率。重复上述分析过程，并对比各个子集的信号周期估计结果，以分析车辆比例对模型估计精度的影响。本文采用了 DBSCAN 聚类算法来分析车辆的停止时刻数据，进一步对每个聚类的结束时刻进行了逐一增加的微调，或只取每簇中停止时间最长的车辆，以模拟高车流量、低车流量的真实情况。本文基于随机均匀分布对车辆实际行车轨迹坐标进行修改，以模拟定位误差。

对于问题三，利用傅里叶变换从历史轨迹数据中提取出路口信号灯的主要频率成分，从而估算出信号周期的平均长度。本文采用了滑动窗口技术，对路口的交通流量进行时段划分，监测每个时段的车辆通过次数。本文应用了 DBSCAN 聚类算法，以确定变化后的周期长度分布，并使用应用 GEV 模型分析异常时段的具体信号灯周期。

对于问题四，本文对附件四提供的每辆车的轨迹数据进行了二维可视化。本文定义了十二种车辆通行模式，对数据进行 K-MEANS 聚类分析，将不同行驶模式的车辆区分开来。本文聚焦于单个模式的深入分析。通过计算以车辆为圆心的特定半径内的点的密

度，识别出了车流密集区域，并揭示了信号灯变换周期的初步估计。本文采用了傅里叶变换来分析周期性特征，并使用广义极值 (GEV) 模型对红灯持续时间进行了统计建模和拟合。

6.1.2 模型的缺点

高数据依赖性：本模型对数据的质量和数量有很高的要求。如果采集的数据量不足或数据质量低（例如位置精度不够或数据采集间隔过长），可能会直接影响模型的分析精度和可靠性。

实时响应限制：模型处理和分析数据的时间可能较长，这限制了其在需要快速响应的实际交通管理场景中的应用潜力。例如，在快速变化的交通状况下，模型可能难以提供实时更新。

技术鲁棒性问题：虽然模型采用了先进的聚类算法和频谱分析技术，但在处理非常规交通数据或应对突发交通事件时的鲁棒性还需进一步提高。在非标准交通流或异常模式下，模型的预测准确性可能会下降。

6.2 模型的推广

本文设计了基于频谱分析和 DBSCAN 聚类的交通信号灯问题处理模型，包括利用车辆速度的傅里叶变换识别交通流信号周期和通过聚类分析低速行驶时段来评估信号灯持续时间。这些模型在交通管理领域可应用于智能交通系统，帮助交通管理部门优化信号灯控制，减少交通拥堵，并提高路面安全性。在人工智能领域，模型可集成至智能城市管理系统，为城市交通提供数据支持和决策依据。此外，这些模型也可用于驾驶员行为分析和自动驾驶系统中，提高系统对交通信号反应的精确性和时效性。

在企业管理领域，该模型可应用于物流和运输管理，通过优化路线规划和货车调度，减少运输成本和提升效率。同时，模型的分析能力可以支持紧急响应系统，例如在救护车和消防车的调度中优先处理信号灯控制，以便它们更快地穿越城市。

综上所述，本文设计的模型具有广泛的实际应用价值，将为相关领域的智能交通管理、企业物流优化、紧急响应系统以及自动驾驶技术提供有效的支持和参考。

参考文献

[1] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. KDD'96, page 226–231. AAAI Press, 1996.

[2] Marc Green. "how long does it take to stop?" methodological analysis of driver perception-brake times. Transportation Human Factors, 2(3):195–216, 2000.

[3] 张文博. 基于多源数据融合的异构车辆数据匹配算法研究及应用. PhD thesis, 吉林大学, 2019.

[4] 白跃升. 基于 GSM 数据的交通出行预测模型的构建与应用. PhD thesis, 中北大学, 2019.

[5] 苏丽娜. 运动车辆的轨迹分析算法研究. PhD thesis, 重庆邮电大学, 2011.

附录 A　A1park.py

```python
import pandas as pd

# 加载数据
df = pd.read_csv('A1.csv', delimiter=',')

# 计算每辆车每一秒的速度
df['speed'] = df.groupby('vehicle_id')[['x', 'y']].diff().pow(2).sum(axis=1).pow(0.5)

# 标记在 x=11.4 处的停止
df['at_11_4'] = (df['x'] == 11.4)

# 排序数据，确保时间顺序
df = df.sort_values(by=['vehicle_id', 'time'])

# 使用groupby和transform找到每辆车最后一次在11.4的时间点
df['last_time_at_11_4'] = df[df['at_11_4']].groupby('vehicle_id')['time'].transform('max')

# 现在选择这些最后时刻的记录
last_times_at_11_4 = df[df['time'] == df['last_time_at_11_4']]

# 选择需要的列和行
last_moving_times = last_times_at_11_4[last_times_at_11_4['at_11_4']][['vehicle_id', 'time', 'x',
    'y', 'speed']]

# 打印每次车辆最后一次在11.4的时间
print("Last times at x=11.4 with vehicle details:")
print(last_moving_times)

# 计算相邻启动时间点的差值，并将其与车辆ID关联
last_moving_times['time_diff'] = last_moving_times['time'].diff()

# 打印时间差值及其对应的车辆ID和时间点
print("Time differences between consecutive last events at x=11.4 with vehicle details:")
print(last_moving_times.dropna(subset=['time_diff']))  # 输出具有有效时间差的行

# 计算时间差的众数
time_diff_mode = last_moving_times['time_diff'].mode()[0]  # 取众数的第一个元素，防止有多个众数

# 打印时间差众数
print("Most common time difference at x=11.4:")
print(time_diff_mode)
```

附录 B　DBSACN.py

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from scipy.stats import genextreme
import matplotlib.pyplot as plt
import matplotlib
```

```
8
9  # 设置 matplotlib 的字体为支持中文的字体
10 matplotlib.rcParams['font.family'] = 'SimHei'  # 'SimHei' 是黑体的意思
11 matplotlib.rcParams['font.size'] = 14
12 matplotlib.rcParams['axes.unicode_minus'] = False  # 正确显示负号
13
14 # 加载数据
15 data = pd.read_csv('low_speed_intervals_modifiedS5.csv')
16
17 # 选择我们用于聚类的列
18 X = data[['start_time', 'end_time']]
19
20 # 配置DBSCAN参数
21 dbscan = DBSCAN(eps=50, min_samples=1)
22
23 # 进行聚类
24 clusters = dbscan.fit_predict(X)
25
26 # 将聚类结果添加到原始数据中
27 data['cluster'] = clusters
28
29 # 打印结果
30 print("聚类结果统计：")
31 print(data['cluster'].value_counts())
32
33 ## 存储每个聚类的周期时长及其最小start_time
34 cycle_durations = {}
35
36 # 计算每个聚类的时间跨度并存储周期及其最小start_time
37 for cluster in sorted(set(clusters)):
38     if cluster != -1:  # 不显示噪声点
39         cluster_data = data[data['cluster'] == cluster]
40         # 确保每个聚类中的end_time相差不超过10
41         max_end_time_diff = cluster_data['end_time'].max() - cluster_data['end_time'].min()
42         if max_end_time_diff <= 10:
43             min_start = cluster_data['start_time'].min()
44             max_end = cluster_data['end_time'].max()
45             cycle_duration = max_end - min_start
46             cycle_durations[cluster] = (cycle_duration, min_start)
47             print(f"\nCluster {cluster}: ")
48             print(cluster_data)
49             print(f"Cycle Duration = {cycle_duration} seconds, Start Time = {min_start}")
50
51 # 提取周期时长，并按时长排序，同时保留最小start_time
52 sorted_cycle_durations = sorted(cycle_durations.items(), key=lambda x: x[1][0], reverse=True)
53
54 # 选取周期时长大到小的前25%
55 top_25_percent_index = int(len(sorted_cycle_durations) * 0.3)
56 top_25_percent_cycles = sorted_cycle_durations[:top_25_percent_index]
57
58 # 按 start_time 从小到大排序前 25% 的数据
59 top_25_percent_cycles_sorted_by_start = sorted(top_25_percent_cycles, key=lambda x: x[1][1])
60
61 # 打印高于平均周期时长的周期数据及其最小start_time，按start_time排序
62 print("\nTop 25% Cycle Durations sorted by Start Time:")
63 for cluster, (duration, start_time) in top_25_percent_cycles_sorted_by_start:
```

```
    print(f"Cluster {cluster}: Cycle Duration = {duration} seconds, Start Time = {start_time}")

# 可选：打印最小start_time的最大周期时长
min_start_time = min(top_25_percent_cycles_sorted_by_start, key=lambda x: x[1][1])[1][1]
max_duration_at_min_start = max([dur for clus, (dur, start) in
    top_25_percent_cycles_sorted_by_start if start == min_start_time])
print(f"\nCycle with the smallest Start Time: {min_start_time}, Duration:
    {max_duration_at_min_start} seconds")


# 提取周期时长数据
cycle_lengths = np.array([dur for _, (dur, _) in top_25_percent_cycles_sorted_by_start])

# 拟合GEV模型
c, loc, scale = genextreme.fit(cycle_lengths)

# 观察到的最大周期时长
observed_max_duration = max(cycle_lengths)

# 计算95%置信区间的上限
upper_bound_95 = genextreme.ppf(0.95, c, loc=loc, scale=scale)

# 确定额外的时长值 p
p = upper_bound_95 - observed_max_duration if upper_bound_95 > observed_max_duration else 0

# 最终的估计值是观察到的最大值加上 p
final_estimated_duration = observed_max_duration + p

print(f"观察到的最大周期时长：{observed_max_duration} 秒")
print(f"为近似实际红灯时长需要增加的时长：{p} 秒")
print(f"最终估计的红灯时长：{final_estimated_duration} 秒")

# 绘制数据及拟合曲线
x = np.linspace(min(cycle_lengths), max(cycle_lengths), 100)
y = genextreme.pdf(x, c, loc=loc, scale=scale)
plt.figure(figsize=(10, 6))
plt.hist(cycle_lengths, bins=20, density=True, alpha=0.5, label='周期时长直方图')
plt.plot(x, y, 'r-', label='拟合的GEV分布')
plt.legend()
plt.title('拟合GEV模型至前25%周期时长')
plt.xlabel('周期时长（秒）')
plt.ylabel('密度')
plt.grid(True)
plt.show()
```

附录 C　FindPark.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KDTree

# 加载数据
```

```
data = pd.read_csv('A4.csv')

# 散点图展示
plt.figure(figsize=(10, 6))
plt.scatter(data['x'], data['y'], alpha=0.5)
plt.title("Vehicle Position Scatter Plot")
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.grid(True)
plt.show()

# 使用KDTree优化搜索
tree = KDTree(data[['x', 'y']])

# 定义搜索的圆半径
radius = 0.3
# 初始化最大计数和最佳圆心位置
max_count = 0
best_center = None

# 遍历数据点，用每个点作为圆心，计算半径内的点数
for index, row in data.iterrows():
    center_point = np.array([[row['x'], row['y']]])
    # 计算在半径内的点数
    count = tree.query_radius(center_point, r=radius, count_only=True)
    if count > max_count:
        max_count = count
        best_center = center_point[0]

# 输出结果
print(f"最密集的停车点位坐标为：{best_center}，圆内包含的车辆数为：{max_count}")

# 可视化最佳停车点
plt.figure(figsize=(10, 6))
plt.scatter(data['x'], data['y'], alpha=0.5)
plt.scatter(best_center[0], best_center[1], color='red', s=100)  # 最佳点为红色
circle = plt.Circle((best_center[0], best_center[1]), radius, color='red', fill=False, linewidth=2)
plt.gca().add_patch(circle)
plt.title("Optimal Parking Spot Near Traffic Light")
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.grid(True)
plt.show()
```

附录 D　Error_X.py

```
import pandas as pd
import numpy as np

# 加载数据
df = pd.read_csv('B2.csv', delimiter=',')

# 为x坐标添加-5%到+5%的随机误差
```

```python
df['x'] += df['x'] * np.random.uniform(-0.05, 0.05, df.shape[0])


df['x'] = df['x'].round(2)

# 打印修改后的数据
print(df)

# 可以选择保存修改后的数据
df.to_csv('Error_B2_X.csv', index=False)
```

附录 E　Kalman_1.py

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

# 设置matplotlib配置，使用中文字体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 指定默认字体为黑体
matplotlib.rcParams['axes.unicode_minus'] = False  # 正常显示负号


def moving_average(data, window_size=5):
    """计算移动平均，使用一个简单的窗口平均滤波器"""
    return data.rolling(window=window_size, center=True).mean()


# 读取CSV文件
df = pd.read_csv("Error_B2_X.csv")

# 对每个车辆ID应用移动平均并可视化
for vehicle_id, group in df.groupby('vehicle_id'):
    group['x_smoothed'] = moving_average(group['x'])
    group['y_smoothed'] = moving_average(group['y'])

    plt.figure(figsize=(10, 6))
    plt.plot(group['x'], group['y'], 'o-', label='原始数据')
    plt.plot(group['x_smoothed'], group['y_smoothed'], 'x--', label='平滑后的数据')
    plt.title(f"车辆ID: {vehicle_id} 轨迹（移动平均平滑）")
    plt.xlabel('X 坐标')
    plt.ylabel('Y 坐标')
    plt.legend()
    plt.grid(True)
    plt.show()
```

附录 F　LackOfSample.py

```python
# 处理前10%，30%，50%，70%条数据
percentages = [0.1, 0.3, 0.5, 0.7]
for p in percentages:
```

```
    num_rows = int(len(data) * p)
    data_sample = data.head(num_rows)

    # 选择我们用于聚类的列
    X_sample = data_sample[['start_time', 'end_time']]

    # 配置DBSCAN参数
    dbscan_sample = DBSCAN(eps=50, min_samples=1)

    # 进行聚类
    clusters_sample = dbscan_sample.fit_predict(X_sample)

    # 将聚类结果添加到原始数据中
    data_sample['cluster'] = clusters_sample

    # 打印结果
    print(f"\n聚类结果统计 - {p*100}% sample:")
    print(data_sample['cluster'].value_counts())

    # 存储每个聚类的周期时长
    cycle_durations_sample = {}

    # 计算每个聚类的时间跨度并存储周期
    for cluster in sorted(set(clusters_sample)):
        if cluster != -1:  # 不显示噪声点
            cluster_data = data_sample[data_sample['cluster'] == cluster]
            min_start = cluster_data['start_time'].min()
            max_end = cluster_data['end_time'].max()
            cycle_duration = max_end - min_start
            cycle_durations_sample[cluster] = cycle_duration
            print(f"\nCluster {cluster}: ")
            print(cluster_data)
            print(f"Cycle Duration = {cycle_duration} seconds")

    # 提取周期时长，并排序
    sorted_cycle_durations_sample = sorted(cycle_durations_sample.values(), reverse=True)

    # 选取周期时长大到小的前25%
    top_25_percent_index_sample = int(len(sorted_cycle_durations_sample) * 0.25)
    top_25_percent_cycles_sample = sorted_cycle_durations_sample[:top_25_percent_index_sample]

    # 打印高于平均周期时长的周期数据
    print("\nTop 25% Cycle Durations:")
    for duration in top_25_percent_cycles_sample:
        print(f"Cycle Duration = {duration} seconds")

    # 打印最大周期时长
    max_duration_sample = max(top_25_percent_cycles_sample) if top_25_percent_cycles_sample else 0
    print(f"\nMaximum Cycle Duration: {max_duration_sample} seconds")
```

附录 G　FLY.py

```
import pandas as pd
```

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft, fftfreq
from collections import defaultdict

# 读取CSV文件
df = pd.read_csv('C1.csv')

# 计算每辆车的速度
def calculate_speed(group):
    group.sort_values(by='time', inplace=True)
    time_diff = group['time'].diff().fillna(0)
    x_diff = group['x'].diff().fillna(0)
    y_diff = group['y'].diff().fillna(0)
    speed = np.sqrt(x_diff**2 + y_diff**2) / time_diff
    speed.replace([np.inf, -np.inf], np.nan, inplace=True)
    group['speed'] = speed.fillna(0)
    return group

df['timestamp'] = pd.to_datetime(df['time'], unit='s')
df_speed = df.groupby('vehicle_id').apply(calculate_speed)
average_speed = df_speed.groupby('timestamp')['speed'].mean()

# 分析速度数据的频率特性
window_size = int(0.05 * len(average_speed))
overlap = window_size // 2
windowed_fft_results = []

for start in range(0, len(average_speed), overlap):
    end = start + window_size
    if end > len(average_speed):
        break
    window_data = average_speed[start:end].values
    fft_result = fft(window_data)
    freqs = fftfreq(len(window_data), d=1)
    positive_freqs = freqs > 0
    fft_magnitude = np.abs(fft_result[positive_freqs])
    freq_positive = freqs[positive_freqs]
    windowed_fft_results.append((start, end, freq_positive, fft_magnitude))

# 汇总显著频率及其时间
significant_freq_windows = defaultdict(list)
for start, end, freqs, magnitudes in windowed_fft_results:
    threshold = 0.9 * np.max(magnitudes)
    significant_indices = np.where(magnitudes > threshold)[0]
    significant_freqs = freqs[significant_indices]
    significant_periods = 1 / significant_freqs
    for freq, period in zip(significant_freqs, significant_periods):
        significant_freq_windows[freq].append((period, start, end))

# 打印结果并排序
sorted_freq_time_summary = sorted(significant_freq_windows.items(), key=lambda x: x[0])
for freq, data in sorted_freq_time_summary:
    print(f"Frequency: {freq:.6f} Hz")
    for period, start, end in data:
        print(f"  Period: {period:.2f} s, Start: {start}, End: {end}")
```

```
58
59 # 可视化频谱及时间-周期图
60 colors = plt.cm.viridis(np.linspace(0, 1, len(sorted_freq_time_summary)))
61 plt.figure(figsize=(12, 6))
62 plt.title('Time vs. Period for Different Frequencies')
63 plt.xlabel('Time Index')
64 plt.ylabel('Period (s)')
65 plt.grid(True)
66
67 for (freq, data), color in zip(sorted_freq_time_summary, colors):
68     for period, start, end in data:
69         plt.hlines(period, start, end, colors=color, linewidth=6, label=f"Freq: {freq:.6f} Hz" if
              start == min([d[1] for d in data]) else "")
70
71 plt.legend(loc='upper right', fontsize='small')
72 plt.show()
```

附录 H　GEV.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy.stats import genextreme
4
5
6  def estimate_cycle_duration(cycle_lengths):
7      # 拟合GEV模型
8      c, loc, scale = genextreme.fit(cycle_lengths)
9
10     # 观察到的最大周期时长
11     observed_max_duration = max(cycle_lengths)
12
13     # 计算95%置信区间的上限
14     upper_bound_95 = genextreme.ppf(0.95, c, loc=loc, scale=scale)
15
16     # 确定额外的时长值 p
17     p = upper_bound_95 - observed_max_duration if upper_bound_95 > observed_max_duration else 0
18
19     # 最终的估计值是观察到的最大值加上 p
20     final_estimated_duration = observed_max_duration + p
21
22     # 输出结果
23     print(f"Observed maximum cycle duration: {observed_max_duration} seconds")
24     print(f"Additional time to approximate actual red light duration: {p} seconds")
25     print(f"Final estimated red light duration: {final_estimated_duration} seconds")
26
27     # 绘制数据及拟合曲线
28     x = np.linspace(min(cycle_lengths), max(cycle_lengths), 100)
29     y = genextreme.pdf(x, c, loc=loc, scale=scale)
30     plt.figure(figsize=(10, 6))
31     plt.hist(cycle_lengths, bins=20, density=True, alpha=0.5, label='Cycle Lengths Histogram')
32     plt.plot(x, y, 'r-', label='Fitted GEV distribution')
33     plt.legend()
34     plt.title('Fit of GEV Model to Cycle Durations')
```

```
    plt.xlabel('Cycle Duration (seconds)')
    plt.ylabel('Density')
    plt.grid(True)
    plt.show()


# 假设你有一个包含周期时长的数组 cycle_lengths，例如：
cycle_lengths = np.array([72,71,69,75,86])
estimate_cycle_duration(cycle_lengths)
```

附录 I　MoveWindows.py

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

def load_data(filepath):
    data = pd.read_csv(filepath)
    # 打印列名以检查
    print("Column names in the dataset:", data.columns)
    return data

# 调整其他函数中的 'vehicle_id' 到正确的列名

def in_circle(x, y, center, radius):
    return (x - center[0]) ** 2 + (y - center[1]) ** 2 <= radius ** 2


def calculate_velocity(data):
    # 为数据框增加速度列
    data = data.sort_values(by=['vehicle_id', 'time'])
    data['x_diff'] = data.groupby('vehicle_id')['x'].diff()
    data['y_diff'] = data.groupby('vehicle_id')['y'].diff()
    data['time_diff'] = data.groupby('vehicle_id')['time'].diff()

    # 计算速度：速度是空间移动距离与时间差的比值
    data['velocity'] = np.sqrt(data['x_diff'] ** 2 + data['y_diff'] ** 2) / data['time_diff']
    return data


def filter_stopped_vehicles(data):
    # 判断每辆车是否有速度为0的记录
    stopped_vehicles = data[data['velocity'] == 0]['vehicle_id'].unique()
    # 返回没有停车记录的车辆数据
    return data[~data['vehicle_id'].isin(stopped_vehicles)]


def count_vehicles_in_circle(data, center, radius):
    mask = data.apply(lambda row: in_circle(row['x'], row['y'], center, radius), axis=1)
    filtered_data = data[mask]
    if filtered_data.empty:
        return 0, []
    return len(filtered_data['vehicle_id'].unique()), filtered_data['vehicle_id'].unique()
```

```python


def sliding_window_analysis(data, centers, radius, total_cycle, max_red_duration):
    data = calculate_velocity(data)
    data = filter_stopped_vehicles(data)

    step_size = total_cycle  # 将步长设置为信号灯的总周期
    vehicle_counts = []
    last_vehicle_times = []

    for start in range(0, data['time'].max(), step_size):
        end = start + max_red_duration  # 窗口大小设为最长红灯时长
        window_data = data[(data['time'] >= start) & (data['time'] < end)]
        vehicle_count = 0
        last_times = []
        for center in centers:
            count, vehicles = count_vehicles_in_circle(window_data, center, radius)
            vehicle_count += count
            last_times.extend(window_data[window_data['vehicle_id'].isin(vehicles)]['time'].max()
                for _ in vehicles if
                              not window_data.empty)

        vehicle_counts.append((start, vehicle_count))
        last_vehicle_times.append((start, max(last_times) if last_times else None))

    return vehicle_counts, last_vehicle_times


def plot_vehicle_counts(vehicle_counts):
    times, counts = zip(*vehicle_counts)
    plt.figure(figsize=(10, 5))
    plt.plot(times, counts, marker='o')
    plt.title('Vehicle Counts Over Time')
    plt.xlabel('Time (seconds)')
    plt.ylabel('Unique Vehicles in Circles')
    plt.grid(True)
    plt.show()

# 参数配置
filepath = 'C1.csv'
centers = [(12.1, 3.61), (11.77, 0.43)]  # 真实坐标替换
radius = 0.3
total_cycle = 88
max_red_duration = 56

# 加载数据
data = load_data(filepath)

# 分析滑动窗口
vehicle_counts, last_vehicle_times = sliding_window_analysis(data, centers, radius, total_cycle,
    max_red_duration)

# 绘制结果
plot_vehicle_counts(vehicle_counts)

# 打印每个窗口的结果
```

```
for start, count in vehicle_counts:
    print(f"Start Time: {start} seconds, Vehicle Count: {count}")

for start, last_time in last_vehicle_times:
    print(f"Start Time: {start} seconds, Last Vehicle Time: {last_time if last_time else 'No
        Vehicles'}")
```

附录 J　DivideInto12.py

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import numpy as np

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

# 提取每个车辆的起始点和结束点
grouped = data.groupby('vehicle_id')
features = grouped.agg(start_x=('x', 'first'), start_y=('y', 'first'),
                       end_x=('x', 'last'), end_y=('y', 'last')).reset_index()

# 计算方向和距离
features['direction'] = np.arctan2(features['end_y'] - features['start_y'],
                                   features['end_x'] - features['start_x'])
features['distance'] = np.sqrt((features['end_x'] - features['start_x'])**2 +
                               (features['end_y'] - features['start_y'])**2)

# 特征标准化
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features[['start_x', 'start_y', 'end_x', 'end_y',
    'direction', 'distance']])

# 应用K-means聚类
kmeans = KMeans(n_clusters=12, random_state=42)
features['cluster'] = kmeans.fit_predict(features_scaled)

# 输出聚类结果
features[['vehicle_id', 'cluster']].to_csv('clustering_output_pro_cleaned.csv', index=False)
```