基于 DBSCAN 聚类算法的交通信号灯周期预测

摘要

交通拥堵已成为城市发展的严峻挑战，准确估计交通信号灯周期至关重要。本文使用基于 DBSCAN 的聚类算法根据车流轨迹数据估计不同条件下交通信号灯的周期。同时，对轨迹坐标等因素进行误差分析，并进一步探究信号灯周期跳变规律。

针对问题一，首先，对数据进行预处理得到有效数据，包含缺失值与异常值的检查；然后对数据进行标准化处理。通过对车辆轨迹数据进行可视化处理，可以得到 5 个路口的形态，故根据相邻时间间隔的数据计算得到车辆的瞬时速度和瞬时加速度。其次，根据运动形态的不同分为四类，接着设置**速度阈值**、**加速度阈值**和**距离阈值**；进而提取运动状态发生变化的**启停时间**，将的启停时间的数据依次输入 **DBSCAN 聚类**模型；最后，对每个路口中每批次车流计算得到的信号灯的红、绿周期时长结果进行统计分析，确定最可能的交通信号灯周期。

针对问题二，考虑定位误差、样本车辆比例、车流量三个因素对模型精度的影响。其中，定位误差本文采用对车辆坐标数据添加**随机噪声扰动**的方式考虑；对于样本车辆比例采用**均衡步长遍历**的方式车流量以**信号灯可视距离阈值**改变的方式考虑。计算出在各因素影响下的**相对误差**，发现定位误差和样本车辆比例对模型估计精度影响相对较大，而车流量对精度不具有显著影响。因此，取最终误差大小为三个误差值的平均值。最后，将 5 个路口的数据输入经过**误差修正**后的模型得到信号灯的红、绿周期时长。

针对问题三，交通信号灯的周期可能会因为实际情况和交通管理部门的需求而调整。首先计算技术周期内所有车辆的启停时间并刻画时序关系。将同一辆车的启停时间之差作为红灯周期变化样本点，时序相邻车辆的启动时间之差作为交通灯完整变化周期样本点，对间隔较大的样本点数值进行**均衡放缩**，以得到交通灯完整周期变化趋势。然后使用**贝叶斯变点检测**算法检测红灯变化时序中的突变值，以此得出 C1-C6 每个路口的交通灯变化规律。

针对问题四，首先，对车辆轨迹数据进行可视化处理，发现该路口是为十字路口。考虑十字路口的车流转向情况较为复杂，根据车辆起始点坐标测算**转向运动路径**，将路口的所有车流划分为 12 个不同的运动路线，每条路线由单独的交通信号灯控制。然后使用**阈值搜索法**将整个路口车流数据分配到 12 条路线之中，使用前文建立的**修正误差 DBSCAN 模型**分别对每条路线的车流进行聚类分析，即可得到所有路线的交通信号灯变换周期。

关键字：**交通信号灯周期　DBSCAN 聚类算法　统计分析　时序预测**

一、 问题重述

1.1 问题背景

随着互联网、汽车电子和无线手持设备对导航需求的剧增以及中国网民对于电子地图的逐渐认可，网上电子地图的应用步伐大大加快 [1]。其中，交通信号灯的红绿周期问题是电子地图服务商所需要的重要参数。由于各种原因，许多信号灯未接入网络，所有信号灯的数据无法直接从交通管理部门获取，在路口安排人工读取信号灯的周期信息不切实际，故公司计划使用大量客户的行车轨迹数据来估计交通信号灯的周期。为了给司机提供更好的交通服务，交通信号灯的红绿周期时长是亟待解决的问题。

1.2 问题提出

已知所有信号灯只有红、绿两种状态，同一车道可能只允许一个方向行进，也可能允许两个方向行进，比如直行或右转、直行或左转等。需要构建数学模型解决下列问题:

问题一：在信号灯周期固定不变且已知所有车辆车轨迹的前提下，建立数学模型，使用大量车辆行车轨迹数据来估计信号灯的红绿周期。附件 1 中给出的是 5 个不相关的路口各自一个方向连续 1 个小时内车辆的行车轨迹数据，需要求出这些路口相应方向的信号灯周期，且按照相应的格式要求填入表格 1 中。

问题二：事实上，并不是所有的司机均使用该公司的产品，即只能获取部分样本车辆的行车轨迹。而且受多种因素的影响，轨迹数据存在一定的定位误差，但误差大小未知。考虑定位误差、样本车辆比例、车流量等因素对数学模型估计精度的影响。附件 2 中是另外 5 个不相关的路口各自一个方向连续 1 小时内样本车辆的轨迹数据，需要求出这些路口相应方向的信号灯周期，并且按照和问题一同样的格式要求填入表格 2 中。

问题三：假设信号灯的周期可能发生变化，是否能尽快检测出这种变化以及变化后的新周期。附件 3 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，需要判断这些路口相应方向的信号灯周期在该段时间内有无变化，并且求出周期改变的时刻以及新旧周期的参数，按照与问题一相同的格式要求填入表格 3 中，且指明识别出周期变化所需的时间和条件。

问题四：附件 4 中是某路口连续 2 小时内所有方向样本车辆的轨迹数据，需要建立数学模型求出该路口信号灯的周期。

二、 问题分析

2.1 问题一的分析

题给数据均为车辆行驶在路段的轨迹坐标，可通过绘制散点图与箱线图来观测异常值与缺失值。要预测交通信号灯的周期变化规律，需判断车辆在何时遇到红灯停止，何时遇到绿灯启动。单纯的位移轨迹数据无法得出结论，需考虑将位移的时序数据转化为速度与加速度时序数据，然后分别设定阈值来判断汽车的启停时间。然而，题给数据跨度较大，路口的整条车流横跨了多个信号灯变换周期。因此，首先可考虑采用基于密度的聚类算法，将大致处于相同信号灯周期下的车流汇聚成同一类别，然后分析相邻类别车辆的启动时间，即可得到信号灯完整变换周期；分析同一类别内的启动时间与停止时间，即可得到红灯的持续时间。

2.2 问题二的分析

问题二需在问题一基础之上考虑定位误差、车流量、样本车辆比例对预测结果的影响。对于定位误差，可考虑对车辆的 X、Y 坐标分别添加小幅度的随机的噪声来造成轻微扰动；对于样本车辆比例，可按照符合实际情况的一系列比例从完整的车辆数据种随机抽取一定数量车辆输入到模型中计算，再与使用全部数据的结果进行对比，从而得到误差；对于车流量，可通过对聚类后的类内样本点数量对流量大小进行划分，对不同车流大小内的车辆分别进行分析以预测信号灯周期。

2.3 问题三的分析

由于车流量不同，信号灯的周期可能会随车流量大小而改变。由于在计时周期内，交通信号灯周期有可能能会发生改变，故直接聚类的结果不再精确。应当首先计算所有车辆的启停时间点，相同汽车的启动时刻减去停止时刻即作为一个红灯周期样本点，时序相邻的辆启动时间之差即作为一个完整周期样本点，然后绘制相关折线统计图来对数据进行统计分析。需要注意的是，相邻时序的汽车启动时间之差并非一定是一个信号灯完整周期，也有可能是完整周期的整数倍。最后，对一段时间内相对稳定的数据进行聚类计算，即可得到对应时间段内的信号灯变化周期。

2.4 问题四的分析

本问数据量较大，应绘制出所有车辆的运行轨迹以判断路口形状。然后，根据车辆的起点与终点对车辆运动路线进行分类，不同的路线由不同的交通信号灯进行控制。将每辆车分配到对应的路线后即可得到每条路线上的车流数据，利用前文所建立的模型对所有路线车流量进行聚类求解，即可分别得到路口中所有路线的信号灯变换周期。

三、 模型假设

1. 监测设备无故障，检测数据均为实际数值。
2. 路面所有车辆均正常行驶，红灯停止绿灯通行，无交通事故等异常情况发生。
3. 假设检测数据均以米为计量单位。
4. 假设路面行驶的车辆车身长度一致，不考虑车身长度差异带来的影响。
5. 车辆变道所产生的横向位移忽略不计。

四、 符号说明

符号	说明	单位
$x_{i,t}$	ID 为 i 的车辆在 t 时刻的横坐标	m
$y_{i,t}$	ID 为 i 的车辆在 t 时刻的纵坐标	m
$v_{i,t}$	ID 为 i 的车辆在 t 时刻的瞬时速度	m/s
$a_{i,t}$	ID 为 i 的车辆在 t 时刻的瞬时加速度	m/s^2
v_i^*	为 ID 为 i 的车辆设置的速度阈值	m/s
a_i^*	为 ID 为 i 的车辆设置的加速度阈值	m/s^2
D	距离阈值	m
x_0	红绿灯的 x 坐标	m
y_0	红绿灯的 y 坐标	m
$d_{i,t}$	ID 为 i 的车辆在 t 时刻离红绿灯的距离	m
t_{red}	一个周期内红灯持续时长	s
t_{green}	一个周期内绿灯持续时长	s
TD	两波车流所处时间点的差值	s
T_i	一个红绿灯周期持续时长	s
$\boldsymbol{\varepsilon_x}$	x 坐标上的相对误差	/
$\boldsymbol{\varepsilon_y}$	y 坐标上的相对误差	/

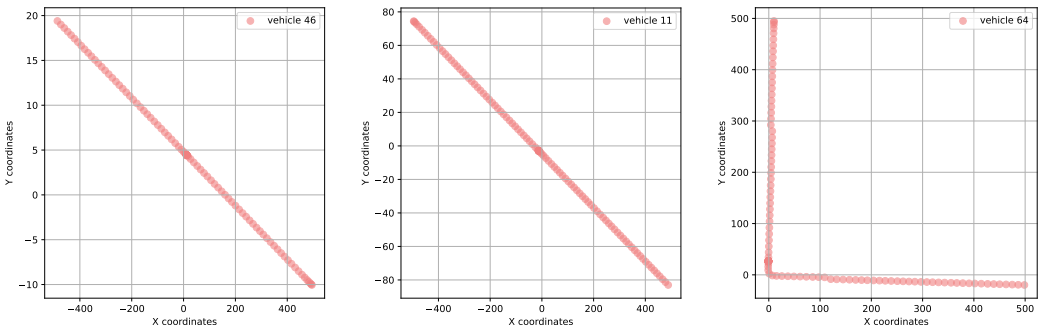
符号	说明	单位
$\overline{\varepsilon}$	平均相对误差	/
x^*	误差修正后车辆的 x 坐标	m
y^*	误差修正后车辆的 y 坐标	m

五、 模型的建立与求解

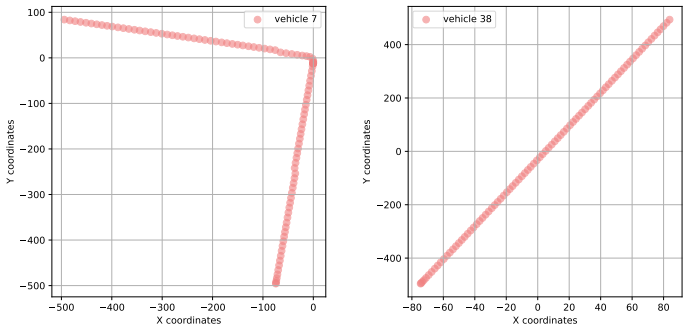
5.1 数据预处理

缺失值的检查 如果数据中存在缺失值，那么会对交通信号灯的红绿周期估计产生一定的影响。因此，对数据进行缺失值的检查是有必要的。

对附件中给出的各个路口中每一个路口相同 ID 的车辆逐一进行缺失值的检查，缺失值的检查采用对每个路口中的每一辆车的轨迹数据绘制散点图，通过数据可视化和人为观察判断检验的方法对其进行检查。以 A1 路口 ID 为 46、A2 路口 ID 为 11、A3 路口 ID 为 64 和 A4 路口 ID 为 7 的车辆轨迹数据绘制的散点图为例，具体的轨迹散点图如下图1所示。



(a) 路口 A1：ID46 车辆的轨迹图 (b) 路口 A2：ID11 车辆的轨迹图 (c) 路口 A3：ID64 车辆的轨迹图

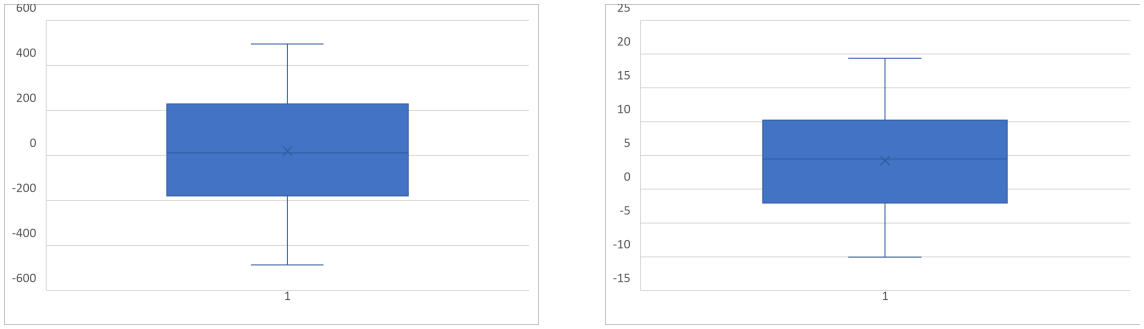


(d) 路口 A4：ID7 车辆的轨迹图 (e) 路口 A5：ID38 车辆的轨迹图

图 1 部分车辆轨迹图

从上图1中的散点图中可以看出：A1 路口 ID 为 46、A2 路口 ID 为 11、A3 路口 ID 为 64 和 A4 路口 ID 为 7 的车辆数据不存在缺失值。然后，采用相同的方法，对附件中各个路口的所有车辆轨迹数据逐一进行检查，并对附件中各个表格的数据进行观察，也可以看出所有车辆的轨迹数据均不存在缺失值。

异常值的检查　由于箱线图能够直观地展示数据的中心趋势、离散程度和异常值，所以本文通过采用对数据绘制箱线图的方式检查是否存在极端异常值。对各个路口的每一辆车的 X、Y 坐标逐一进行检查，本文以路口 A1、ID 为 46 的车辆其 X、Y 坐标数据分别绘制的箱线图为例，如下图2所示。



(a) X 值箱线图　　**(b) Y 值箱线图**

图 2　A1 路口：ID46 车辆的 X、Y 数据的箱线图

通过观察图2可以看出，A1 路口、ID 为 46 的车辆其 X、Y 数据均不存在极端异常值，对各个路口所有车辆其 X、Y 数据进行逐一绘制箱型图检查后发现所有车辆数据均不存在极端异常值，故异常值的检查完成。

5.2 问题一模型的建立与求解

首先对数据进行预处理和标准化处理，计算得到各个车辆的瞬时速度、瞬时加速度。然后设置速度阈值、加速度阈值和距离阈值，进而提取关键的时间节点，最后将数据输入 DBSCAN 聚类模型进行求解，得到最可能的信号灯红、绿周期时长。问题一的流程图如下图3所示。

5.2.1 问题一模型的建立

数据标准化　由于在不同的路口，车辆的轨迹必然不相同，为了度量方便，需要将车辆的轨迹数据全部统一为瞬时速度。由于是相隔 1s 采样，间隔时间足够短，故可以采用 1s 内车辆的平均速度近似作为车辆的瞬时速度。t 时刻瞬时速度的具体定义如下式(1)所示。

$$v_{i,t}=\frac{\sqrt{(x_{i,t}-x_{i,t-1})^2+(y_{i,t}-y_{i,t-1})^2}}{t-(t-1)} \tag{1}$$

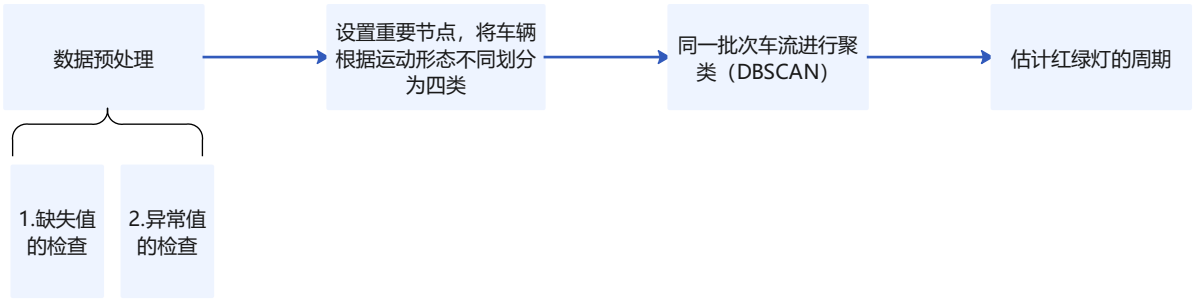


图 3　问题一流程图

其中，$v_{i,t}$ 表示 ID 为 i 的车辆在 t 时刻的瞬时速度；$x_{i,t}$ 表示 ID 为 i 的车辆在 t 时刻的横坐标；$x_{i,t-1}$ 表示 ID 为 i 的车辆在 $t-1$ 时刻的横坐标；$y_{i,t}$ 表示 ID 为 i 的车辆在 t 时刻的纵坐标；$y_{i,t-1}$ 表示 ID 为 i 的车辆在 $t-1$ 时刻的纵坐标；t 表示时刻。

再计算速度的变化率进而得到瞬时加速度，t 时刻瞬时加速度的具体定义式如下式(2)所示。

$$a_{i,t} = \frac{v_{i,t} - v_{i,t-1}}{t - (t-1)} \tag{2}$$

$a_{i,t}$ 表示 ID 为 i 的车辆在 t 时刻的瞬时加速度；$v_{i,t}$ 表示 ID 为 i 的车辆在 t 时刻的瞬时速度；$v_{i,t-1}$ 表示 ID 为 i 的车辆在 $t-1$ 时刻的瞬时速度；t 表示时刻。

运动状态的分类　根据不同的速度和加速度可以将车辆的运动状态划分为四类，具体定义如下式(3)所示。

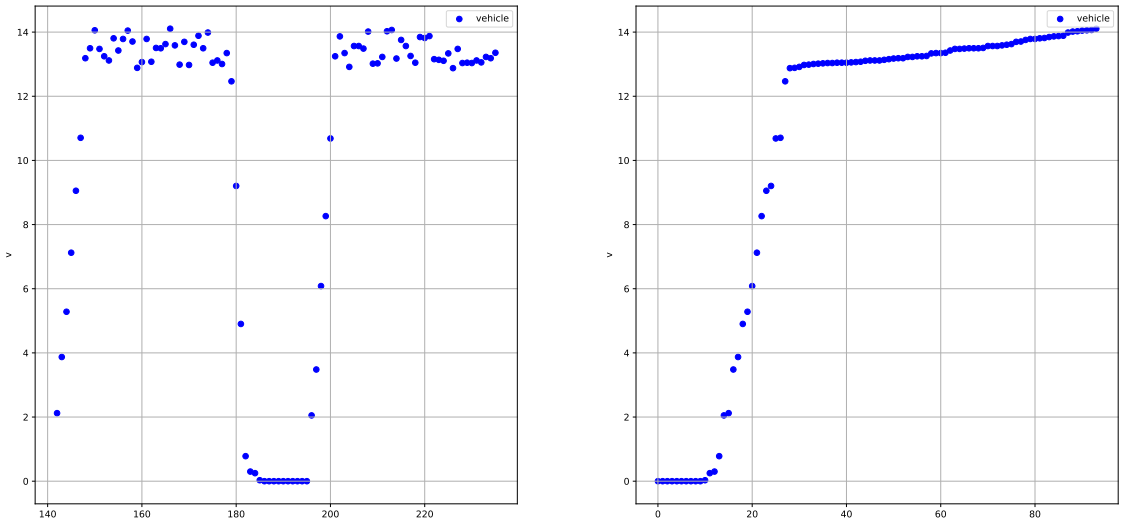
$$\begin{cases} \textbf{平稳行驶:} & v_{i,t} \geq v_i^*, |a_{i,t}| \leq |a_i^*| \\ \textbf{开始减速:} & a_{i,t} \leq a_i^* \\ \textbf{车辆停止:} & v_{i,t} = 0, a_{i,t} \leq 0 \\ \textbf{车辆加速:} & v_{i,t} > 0, a_{i,t} > 0 \end{cases} \tag{3}$$

其中，$v_{i,t}$ 表示 ID 为 i 的车辆在 t 时刻的瞬时速度；，v_i^* 表示 ID 为 i 的车辆设置的速度阈值；$a_{i,t}$ 表示 ID 为 i 的车辆在 t 时刻的瞬时加速度；a_i^* 表示 ID 为 i 的车辆设置的加速度阈值。

速度阈值的确定　本文根据每一辆车各自不同的运动特征，为其设置不同的速度阈值，例如 A1 路口的 46 号车，其 $v-t$ 图像如图 4b 所示。图4 是对图 4b 中速度大小进行升序排序后的散点图。

Step1 数据的过滤:　剔除绝对值小于 r_1 大于 r_1^* 的点。

Step2 重新排序:　对剩余数据按升序排序。



(a) v-t 图 **(b) v-t 图 2**

图 4　速度散点图

Step3 初始值的选取:　选择其中数值最小的，记为 x_{01}，$x_{01} = \min |v_t|$。

Step4 定义条带区域 D_1:　条带的宽度记为 d_1。

Step5 遍历:　以 d_1 大小为步长遍历所有数据。

Step6 统计样本点个数:　统计各个条带中样本点个数，记为 N_1。

Step7 取均值:　选取 N_1 中数值最大的条带内所有点数值的平均值作为阈值，当有多个条带使得点数值最大时，优先选取使得速度值较大的。具体计算公式如式(4)所示。

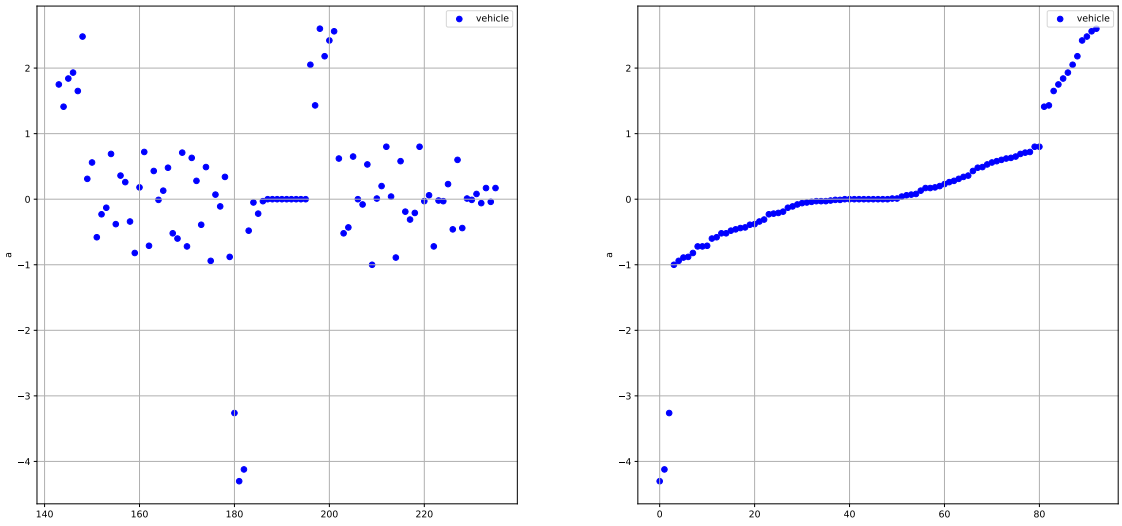
$$v_i^* = \frac{\sum\limits_{x_{n1} \in D_1^{(n)}} v_n}{N} \tag{4}$$

加速度阈值的确定　根据图4b，可以进一步得出 A1 路口 ID46 的车辆其 a-t 图，具体形式如下图5a所示。同样，根据所有加速度的大小从小到大绘制成加速度散点图，具体形式如下图5b所示。

Step1 数据的过滤:　剔除绝对值小于 r_2、大于 0 的点。

Step2 重新排序:　对剩余数据按升序排序。

Step3 初始值的选取:　选择其中数值绝对值最大的，记为 x_{02}，$x_{02} = \max |a_t|$。



(a) a-t 图 **(b) a-t 图 2**

图 5　加速度散点图

Step4 定义条带区域 D_2:　条带的宽度记为 d_2。

Step5 遍历:　以 d_2 大小为步长遍历所有数据。

Step6 统计样本点个数:　统计各个条带中样本点个数，记为 N_2。

Step7 取均值:　选取 N_2 数值最大的条带内所有点数值的平均值作为阈值，当有多个条带使得点数值最大时，优先选取使得加速度值较小的。具体计算公式如式(5)所示。

$$a_i^* = \frac{\sum\limits_{x_{n2} \in D_2^{(n)}} a_n}{N} \tag{5}$$

距离阈值的确定　定义 ID 为 i 的车辆在 t 时刻离红绿信号灯的距离为 $d_{i,t}$，具体的定义式如下式(6)所示，即:

$$d_{i,t} = \sqrt{(x_{i,t} - x_0)^2 + (y_{i,t} - y_0)^2} \tag{6}$$

其中，$x_{i,t}$ 表示 ID 为 i 的车辆在 t 时刻的横坐标；$y_{i,t}$ 表示 ID 为 i 的车辆在 t 时刻的纵坐标；x_0 表示红绿灯的横坐标；y_0 表示红绿灯的纵坐标。可知，x_0, y_0 为红绿信号灯的坐标，本文定义距离阈值 T，当 $d_{i,t} < T$ 时，认为车辆进入了路口区域。

重要节点的确定　根据上述定义的四个不同的运动状态阶段，可以确定下述重要节点:

当 $d_{i,t} \leq D$ 时，且首次 $a_{i,t} \leq a_i^*$，此时的 t 即为车辆开始减速的时刻 Time1，即从该时刻开始信号灯由绿转红，车辆开始减速；

当 $v_{i,t}$ 首次为 0，$a_{i,t} \leq 0$，此时，车辆刚好停止，记该时刻为 Time2。

当 $v_{i,t}$ 、$a_{i,t}$ 为 0 的最后一刻，该刻为车辆开始起步的时刻，记为 Time3，可以认为信号灯从此刻开始由红转绿。

根据上述几个重要的时间点，可以近似将 Time3-Time1 作为一个信号灯的红灯的时长。

由于上述近似只在理想条件下完全精确，实际上，由于每天天气状况不同、不同司机反映时长不同以及当时人流量的大小等因素均会对结果产生影响。因此，我们需要对信号灯的红绿周期进行进一步的判断。

5.2.2 DBSCAN 算法

DBSCAN（Density-Based Spatial Clustering of Applications with Noise）是一种基于密度的空间聚类算法 [2]。该算法将“簇”定义为密度相连的点的最大集合,具有足够密度的区域划分为簇，并在具有噪声的空间数据库中发现任意形状的簇。在本题的问题情景下，不同车辆的不同启停时刻对应着不同的红绿灯周期。为分析红绿灯变化周期，本文使用 DBSCAN 聚类算法将整个时间轴上的所有车辆根据其启动时刻与停止时刻聚为相同类别。依据此聚类结果，可进一步分析出红绿灯周期变化规律。

DBSCAN 将样本分别标记为核心点、边界点、噪声点，并对核心点与边界点进行以簇为单位的聚类。在样本分布中，DBSCAN 根据邻域半径（ϵ）确定样本点的聚类范围，对于领域半径内点数满足 MinPts 的点标记为核心点；对不满足 MinPts 但处于核心点领域内的点则标记为边界点。然后对每个尚未聚类的核心点，以其为起点形成一个新簇，将该核心点及其邻域内的所有核心点和边界点分配到该簇中，然后递归地将所有簇内核心点都分配到该簇中。最终不属于任何簇的样本点即为噪声点。本文所采用的 DNSCAN 算法具体流程如算法 1 所示。

构造样本点集 路口处同行的车辆车辆存在两种情况：

（1） 车辆通过时，路口刚好是绿灯，所以车辆无需进行减速，可直接通过路口。

（2） 看到红灯车辆开始减速直至停止，后又启动加速通过；

对于第一种情况，车辆一直保持一定的速度稳定向前行驶，其通过路口时交通信号灯一直呈绿灯，故无法根据其运动数据预测信号灯周期。因此，本文选取所有在计数时间范畴内存在启停现象的车辆数据，来预测周期信号灯周期变化规律。以 A1 路口为例，首先绘制出如图 6(a) 所示的所有车辆的启停时间散点图。

为了避免时间跨度对结果的影响,本文将上述所提到的关键时间节点 Time1、Time2、Time3 进行正则化处理后转化为均值为 0，标准差为 1 的标准正态分布，以停止时刻点为横坐标，启动时刻点为纵坐标绘制出如图 6(b) 所示的样本空间，并采用 DBSCAN 算法进行聚类。

Algorithm 1: 对路口车辆启停时间的 DBSCAN 聚类

Data: 所有车辆停止时间与启动时间构成的样本点集 $\mathcal{D}$；领域半径 ϵ；构成核心点的领域内点数最小值 MinPts

Result: $\mathcal{D}$ 中每个样本点所属类别 $\mathcal{C}$

对 $\mathcal{D}$ 进行数值归一化处理　//将样本空间缩放到 $(0,1)$ 范围内;

for 每个样本点 i **to** $\mathcal{D}$ **do**

  for 每个样本点 j **to** $\mathcal{D}$ **do**

    if $(x_i - x_j)^2 + (y_i - y_j)^2 \leq \epsilon^2$ **then**

      样本点 $j \rightarrow$ 样本点 i 的邻域 δ_i 中

for 每个样本点 i **to** $\mathcal{D}$ **do**

  if δ_i 中样本点数量 >MinPts **then**

    $i \rightarrow \mathcal{D}_1$　//将 i 标记为核心点;

  else

    if 样本点 $i \in \delta$ **then**

      $i \rightarrow \mathcal{D}_2$　//i 在任一样本点的领域内;

for 每个样本点 i **to** 核心点集 $\mathcal{D}_1$ **do**

  if 样本点 $i \notin \Psi$ **then**

    创建新簇 Ψ;

    将所有 δ_i 中的核心点与边界点递归地纳入到 Ψ 中;

return 所有簇 Ψ 的集合

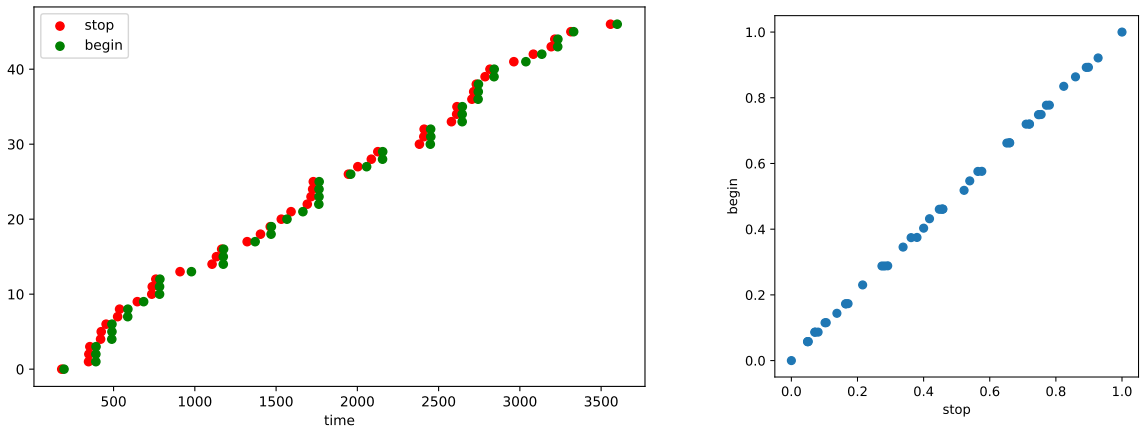
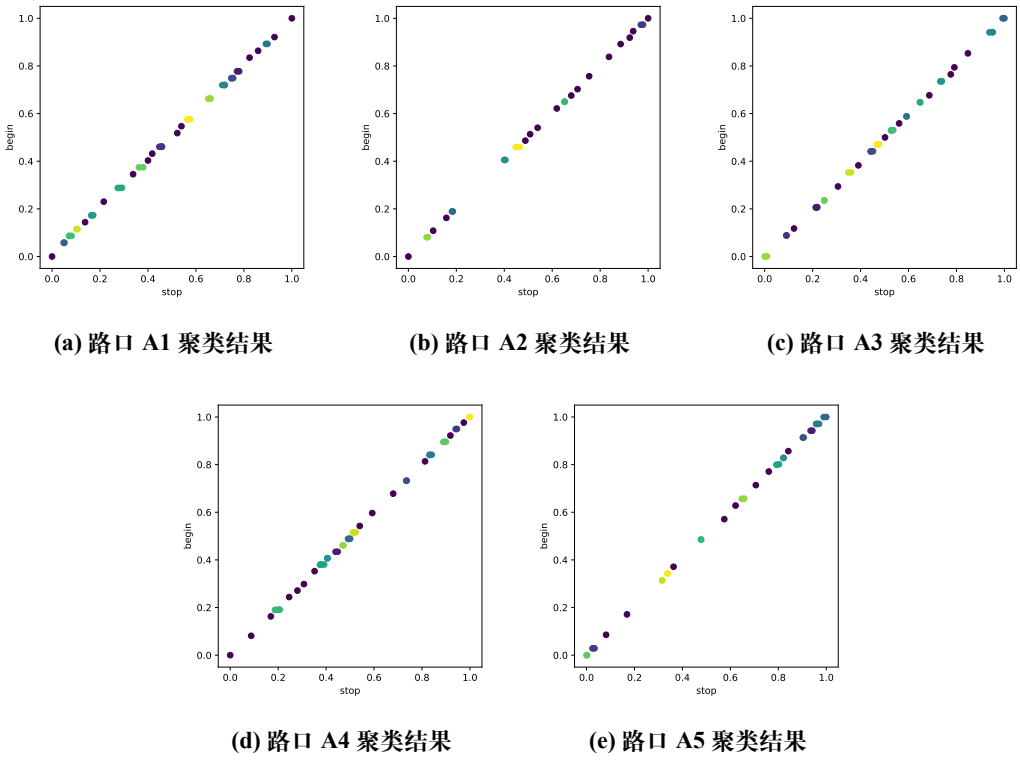


图 6　(a) 所有车辆启停时间点，红色为停止时刻，绿色为停止时刻。(b) 归一化后的启停时间样本空间。

DBSCAN 聚类结果 本文对 5 个路口选择的 ϵ 参数空间为（0.025，0.025，0.025，0.025），$MinPts$ 参数空间为（2，1，2，2，2）。5 个路口的聚类结果如下图7所示。



(a) 路口 A1 聚类结果 **(b) 路口 A2 聚类结果** **(c) 路口 A3 聚类结果**

(d) 路口 A4 聚类结果 **(e) 路口 A5 聚类结果**

图 7 A1-A5 路口聚类结果图

在对车辆进行聚类后，对每一批次的车，可以通过 $Time3 - Time1$ 得到其红灯的时长，记为 t_{red}，如下式(7)所示；通过不同批次的车辆数据，根据 $Time3_i - Time_{i-1}$ 可以得到信号灯红绿周期的总时长，记为 TD，具体如下式(8)所示。

$$t_{red} = Time3_i - Time1_i \tag{7}$$

$$TD = Time3_i - Time3_{i-1} \tag{8}$$

其中，i 代表车辆的批次，$i-1$ 表示上一批车流。以 A1 路口为例，对于各个不同批次的车辆的 t_{red} 进行统计分析，各批次的红灯时长数据如下图所示。对于红灯周期时长的确定，应尽可能取各批次中数值较大的那一个。

取相邻两批次车辆重新启动时的时间差值，即为一个信号灯红绿周期或其整数倍，故应该选取每个路口所有批次 DT 中数值最小的，本文认为该值就是一个红绿周期总时长。绿灯的时长可以通过红绿周期总时长 DT 减红灯时长 t_{red} 进行确定。

A1-A5 路口信号灯的红绿周期时长如表1所示。

表 1　路口 A1-A5 各自一个方向信号灯红、绿周期时长

路口	A1	A2	A3	A4	A5
红灯时长（秒）	71	59	68	77	59
绿灯时长（秒）	27	30	28	15	32

5.3 问题二模型的建立与求解

考虑定位误差、样本车辆比例、车流量三个因素对模型精度的影响。计算出在各因素影响下的相对误差，最终的误差大小即为三个误差值的平均值。最后，将 5 个路口的数据输入经过误差修正后的模型，得到信号灯的红、绿周期时长。

5.3.1 定位误差影响

电子地图无法做到完全精确的车辆定位，必然存在一定幅度的定位误差，进而影响到对交通信号灯的周期预测。本文对车辆坐标数据添加随机噪声扰动。查阅资料 [3] 知由于定位导致的坐标相对误差近似于服从参数空间为 (0,0.025) 的正态分布，即 $\epsilon \sim N(0, 0.025)$，具体形式如式(9)所示。进而，可以得到修正后的值，具体公式如式(10)所示。

$$\varepsilon_x = \frac{x^* - x}{x} \tag{9}$$

$$x^* = x\left(1 + \epsilon_x\right) \tag{10}$$

其中，x 表示定位位置的横坐标，ϵ_x 代表 x 坐标上的相对误差；x^* 表示经过修正后车辆的 x 坐标，纵坐标上定义与横坐标相同。

考虑定位误差后的结果对比　将修正后的定位输入问题一的模型中，得到修正后的红绿灯周期总时长、红灯周期、与修正前的相对误差，与原始数据的对比结果如表 2 所示。

表 2　考虑定位误差前后的各路口信号灯红、绿周期

路口	A1		A2		A3		A4		A5	
	原始	修正	原始	修正	原始	修正	原始	修正	原始	修正
红灯时长（秒）	71	**72**	59	**59**	68	**68**	77	**77**	56	**59**
绿灯时长（秒）	27	**25**	31	**31**	30	**30**	15	**15**	32	**32**

根据表 2 中各个路口的原始交通信号灯周期数据与考虑误差修正后的交通信号灯周期数据，可采用式 (11) 来计算定位误差。

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{N}\frac{(t_{red}^{i} - t_{red}^{i'}) - (t_{green}^{i} - t_{green}^{i})}{t_{red}^{i} + t_{green}^{i'}} \tag{11}$$

其中 N 表示路口数量，t_{red}^{i} 与 t_{green}^{i} 分别表示第 i 个路口的原始红绿灯周期，$t_{red}^{i'}$ 与 $t_{green}^{i'}$ 分别表示误差修正后的周期数据。计算得到定位误差的最终误差大小为 0.00421。

5.3.2 车辆比误差影响

使用电子地图的车辆只是经过路口车流的一部分，因此需要适当减少样本容量，使模型在样本数量少的情况下具有足够的鲁棒性与预测能力。本文按照如式所示的比例 P 随机抽取部分车辆数据来进行红绿灯周期预测。

$$P \in (0.3, 0.975) \quad \text{step} = 0.025 \tag{12}$$

以一个符合实际情况且相对较小的比例 0.3 作为初始值，以足够小的步长 0.025 向比例增大的方向遍历。通过所有路口每次随机抽取对应比例后输入问题一中的模型，得到每个路口的红绿灯总周期，且重复采样 20 次取平均值，将结果与问题一中的结果采用式 (11) 来计算相对误差，得到具体数值变化趋势如图 8。

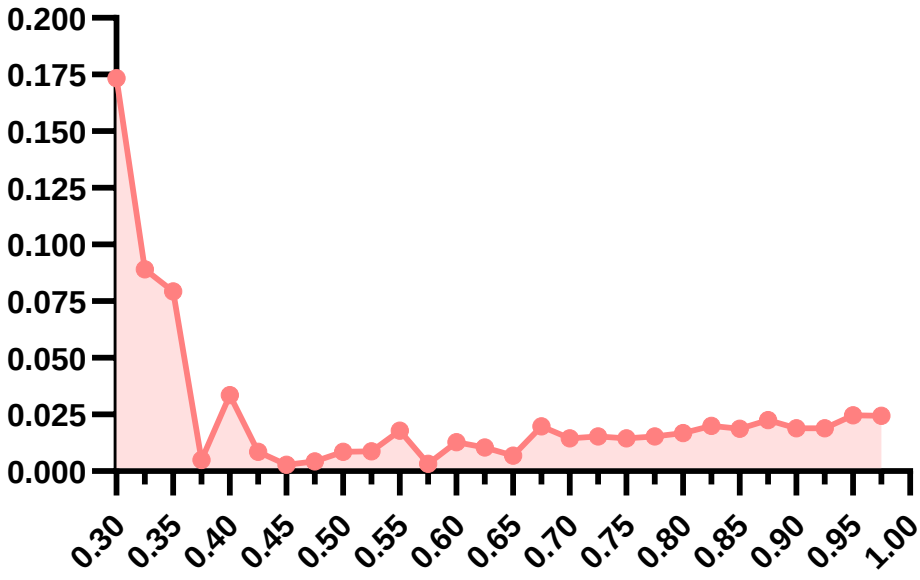


图 8　抽取车辆样本比例的相对误差

通过观察图像，找到相对误差值最小时的比例值；若同时有多个比例值使得相对误差最小，选取比例值较大的。得到最佳抽取车辆比例为 0.575，其对应的最终误差大小为 0.0032844。

5.3.3 车流量的影响

车流量的大小也会对信号灯周期的预测产生影响，因为车流量较大时，汽车的启停动作都会更加缓慢，滞后于交通信号灯变换信号的时刻。要考虑车流量对信号灯周期预测的影响，首先需要确定每个红绿灯周期下的车流大小。

本文根据问题一的聚类结果来考虑车流量大小。以簇内点数量的多少划分为低、中、高三种类型车流。对三种不同类型的车流分别赋予不同的信号灯距离阈值，具体划分如式 (13) 所示。

$$\begin{cases} \textbf{低车流量:} & 1 \leq \textbf{簇内点数} \leq 2 \\ \textbf{中车流量:} & \textbf{簇内点数} = 3 \\ \textbf{高车流量:} & \textbf{簇内点数} \geq 4 \end{cases} \tag{13}$$

对于低车流量，将式 (6) 赋予的距离阈值改为 $d = 75$；对于中车流量，赋予距离阈值 $d = 100$；对于高车流量，赋予距离阈值 $d = 125$ 更大的。通过多次选取不同的距离阈值计算得到最终的结果，可知距离阈值对最终结果不具有显著影响。

综合误差　综合上述定位误差、样本车辆比例误差和车流量的影响，得到综合三个因素的误差大小为 0.0025003。

5.3.4 B1-B5 路口交通信号灯变换周期

将附件 2 中 5 个路口的数据依次输入所建立的模型，得到 5 个路口信号灯的红、绿周期，如表3所示。

表 3　问题二各路口信号灯红、绿周期

路口	B1	B2	B3	B4	B5
红灯时长（秒）	72	92	63	61	101
绿灯时长（秒）	31	34	28	37	26

5.4 问题三模型的建立与求解

首先根据各个路口形状确定交通信号灯的数量。由于红绿灯周期会根据车辆流量大小发生变化，故应使用 DBSCAN 算法对路口车流批次进行聚类后，记录每一组车流的信号灯变化周期，分析其总周期跳变规律与红灯周期跳变规律。

确定路口形状　为确定本问中各个路口的信号灯数量，首先通过车辆运动轨迹绘制出如图 9 所示 C1-C6 的路口形状。

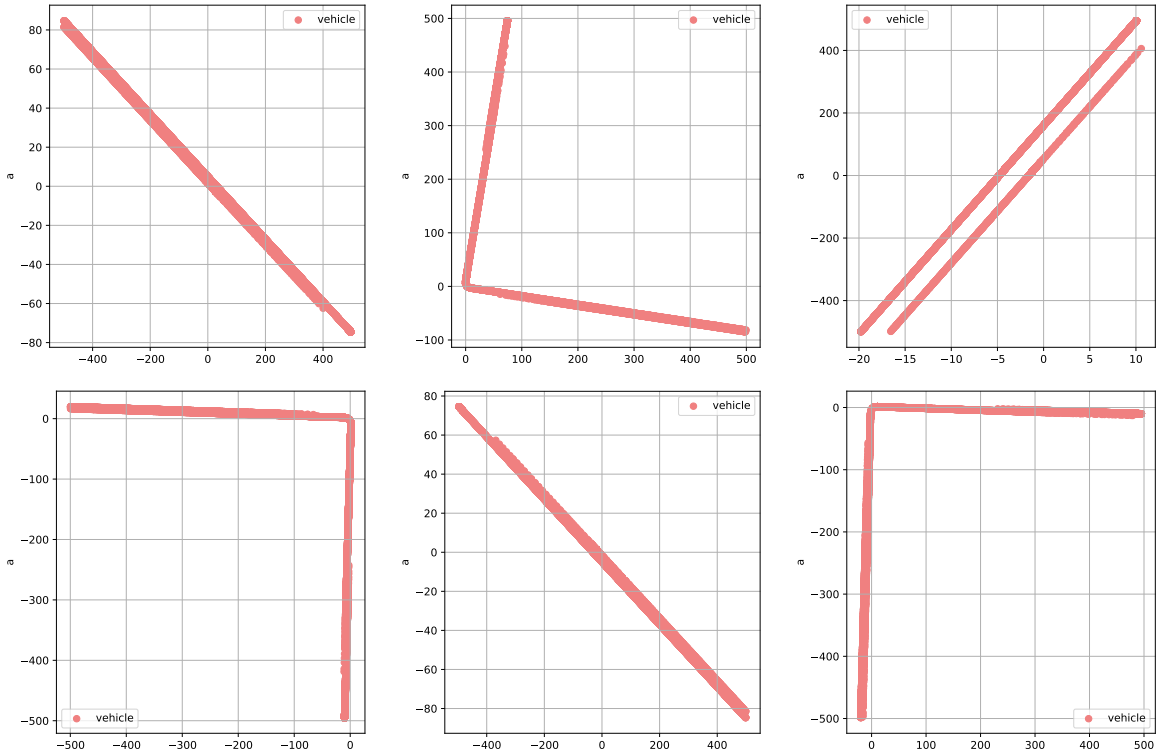


图 9　C1-C6 路口形状（从左至右）

各路口 DBSCAN 聚类　观察图 9 可知，C1-C6 路口均无复杂的车辆转向情况，车流轨迹为直线与直角转弯，故只使用单独的交通信号灯控制即可。为更加明显地观测到每一相邻车流地交通信号灯跳变规律，本文使用 DSCAN 算法对每个路口车流进行聚类时，参数调整为 $\epsilon = 0.015$，MinPts=1，C1-C6 的聚类效果如图所示。

交通信号灯跳变周期　实际交通信号灯周期可能根据车流量变化而调整。因此需分别计算聚类后每一簇的红灯周期与总周期，然后分析变化趋势。对于每一批次车流红灯周期与总周期的计算，可提取每一簇关键时间点 Time1、Time2、Time3，然后使用式(14) 计算红灯周期，使用式(15) 计算信号灯总周期。

$$t_{red} = Time3_i - Time1_i \tag{14}$$

$$TD = Time3_i - Time3_{i-1} \tag{15}$$

以提取出的一系列启动时间为横坐标，红灯时长与信号灯总时长为纵坐标，绘制出各个路口的信号灯变化规律跳变趋势如图 6 所示。

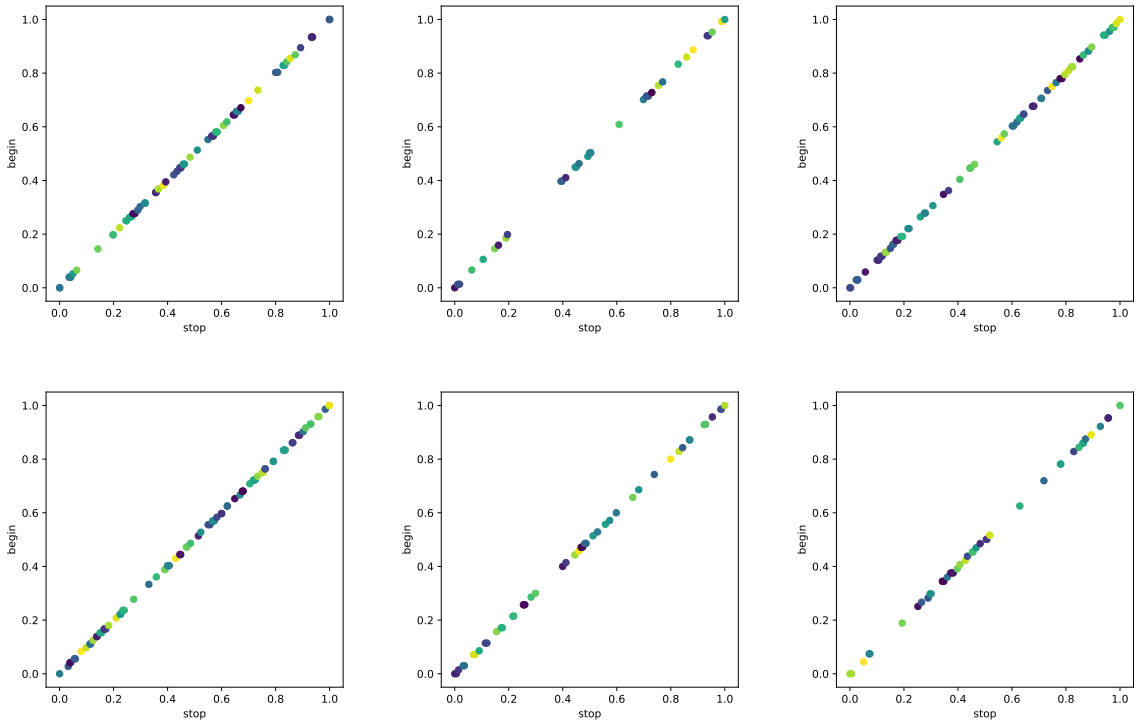


图 10　C1 - C6 路口聚类结果图

在图 11 中每个路口的总周期跳变规律有两种变化趋势。左图红色折线为红灯变化周期，绿色折线为前后车流启动时间之差。考虑到前后车流相隔有可能为信号灯变换周期的倍数，根据绿色折现变换的基本规律将不同大小的车流时间差除以整数，以将车流时间差放缩到信号灯周期左右，压缩后的变化趋势即为右图种的蓝色曲线。

总周期变换趋势　观察图 11 可知，在 C1-C6 中，C1,C2,C4,C5,C6 五个路口的交通信号灯整体周期均波动均匀，维持在同一水平，少数几个突变值持续时间较短，不具有参考价值。通过聚类算法得到五个信号灯的整体周期分别为：93,93,97,90,103 秒。

对于 C3，观察图 11 中的蓝色数据线可知，信号灯总周期分别在 $(0, 2031), (2031, 3991), (3991, 7199)$ 呈现不同的水平波动，故判断辛信号灯总周期在 2031 秒与 3991 秒两个时刻发生了跳变。分别对 3 个不同时间段内的车流进行聚类计算，得到信号灯总周期时间分别为 103.93,103 秒。

红灯周期变化趋势——贝叶斯变点检测　贝叶斯变点检测是一种用于发现时序数据中周期性变化点的有效方法。它建立在概率模型的基础之上，利用贝叶斯推断来识别数据序列中的变化点。具体而言，贝叶斯变点检测假设数据序列可以由不同的概率模型描述，每个模型对应于序列的一个状态。通过计算不同模型的后验概率，可以确定序列中最可能出现变化点的位置。与其他变点检测方法相比，贝叶斯变点检测不仅能够检测出

图 11　C1-C6 路口红灯与信号灯总周期跳变规律。

变化的时间点, 还可以量化变化的概率, 从而更好地反映变化的程度和确定性。

因此本文使用贝叶斯变点检测检测出周期变化点，然后将红灯变化行为的时序数据沿变化点进行分段，然后对时间点分割，分别对分割后的数据采用前文模型，得到具体的红灯持续时间。具体步骤如下：

Step1 输入观测变量:x_1,x_2,x_3,...,x_n;

Step2 设定 t=0 的时刻与任意一个 θ 值 (0<θ<1);

Step3 观测新数据 xt,(i) 初始化后,t 从 1 开始, 或 (ii) t 是从上一个周期开始标识的一个断点;

Step4 通过式 (16) 计算 $\omega_{t+1}^{(j)}$:

$$\omega_{t+1}^{(j)} = \begin{cases} \dfrac{\sqrt{t-j+1}\Gamma\left(\frac{t-j+2\alpha+1}{2}\right)(B+2\beta)^{-\frac{t-j+2\alpha+1}{2}}}{\sqrt{\pi(t-j+2)}\Gamma\left(\frac{t-j+2\alpha}{2}\right)}(A+2\beta)^{-\frac{t-j+2\alpha}{2}}, & j<t \\ (\pi)^{-\frac{1}{2}}2^{\alpha-\frac{1}{2}}\Gamma\left(\frac{2\alpha+1}{2}\right)\left(\frac{x_{t+1}^2}{2}+2\beta\right)^{\frac{2\alpha+1}{2}}, & j=t \end{cases} \tag{16}$$

其中，Γ 为 Gamma 函数。

Step5 根据式 (17) 计算转移概率 $P\left(R_{t+1}=j \mid R_t=i\right)$;

$$P\left(R_{t+1}=j \mid R_t=i\right)=\begin{cases}\frac{1-G(t-i)}{1-G(t-i-1)} & j=i \\ \frac{G(t-i)-G(t-i-1)}{1-G(t-i-1)} & j=t \\ 0 & \text{其他}\end{cases} \tag{17}$$

其中，$G(\quad)$ 为门函数。

Step6 对所有的 $j \leq t$, 根据 **Step5** 中的式子计算变点的后验概率 $P\left(R_{t+1}=j \mid R_t=i\right)$, 如果后验概率的最大值达到 $\tilde{p_1}$, 并且大于阈值 p_0, 则 $\tilde{\tau_1}$=$\tilde{p_1}$, 并跳转到 **Step6**。否则，没有更多的变点。

Step7 返回到 **Step3**∼**Step6**, 使用 $t=\tilde{p_1}$ 直到不再识别到变点,$\tilde{p_1}, \tilde{p_2}, ..., \tilde{p_m}$ 将是变点位置的最终估计值

Step8 估计 $\tilde{\tau_1}$, 分段均值及置信区间。

5.5 红灯变点检测

通过将红灯的特征时序数据处理分析, 去除趋势部分, 得到周期部分经过贝叶斯变点检测, 结果表明只有 C3 路口在 2030 秒和 3991 秒左右存在突变点. 这与我们在前文中预测一致，结果如表4所示。

表 4　问题三结果表

路口	C1	C2	C3	C4	C5	C6
周期 1 红灯时长（秒）	59	69	74	67	59	72
周期 1 绿灯时长（秒）	34	24	29	30	34	31
周期切换时刻	无	无	2031	无	无	无
周期 2 红灯时长（秒）			77			
周期 2 绿灯时长（秒）			21			
周期切换时刻	无	无	3991	无	无	无
周期 3 红灯时长（秒）			77			
周期 3 绿灯时长（秒）			26			

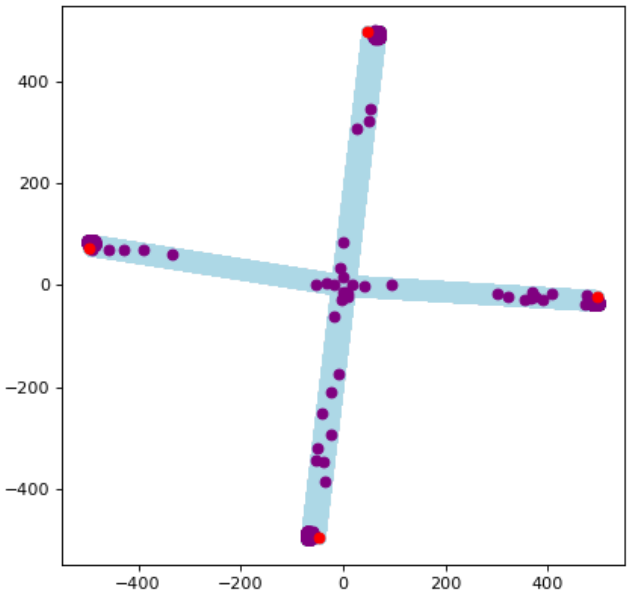


图 12　路口所有车辆的运动坐标

5.6 问题四模型的建立与求解

本题首先按照所有车辆的起始点坐标与行动轨迹分析该路口车流的流动方向，根据不同的车流将该路口枢纽分为 12 个不同的可行路线，每个可行路线由 1 个交通信号等控制。因此，将所有车辆运动轨迹拆分为 12 个不同运动路线的车流，然后根据不同车流，使用 DBSCAN 算法分别预测每一条可行路线上的交通信号灯周期。

5.6.1 划分可行路线

为观察路口形状，首先根据题给数据中每辆车的位置信息，绘制如图 12 所示的散点图。其中蓝色点为所有车辆的运动坐标，红色点为所有车辆进入入口的起始点，紫色点为所有车辆的运动终点。

通过观察图 12 可知，整个路口形状成十字形，车辆均从十字路口四个方向的可观测最远端出发，具体起点坐标为 $(48.27, 496.27)$,$(-496.22, 71.29)$,$(-48.27, -496.27)$,$(495.39, -21.7)$。分别记为 B_1,B_2,B_3,B_4。且运动终点分布在四个路口与路口中心，可知所有车辆均沿十字路口进行运动。

判断车辆路口转向情况　为进一步确定车辆在十字路口枢纽处的转向运动情况，根据不同的出发点，分别观测所其属车辆的终点坐标，以判断其具体行径。四个起始点的终点坐标集合如图 13 所示。其中红色点为车辆起点，紫色点为车辆终点。

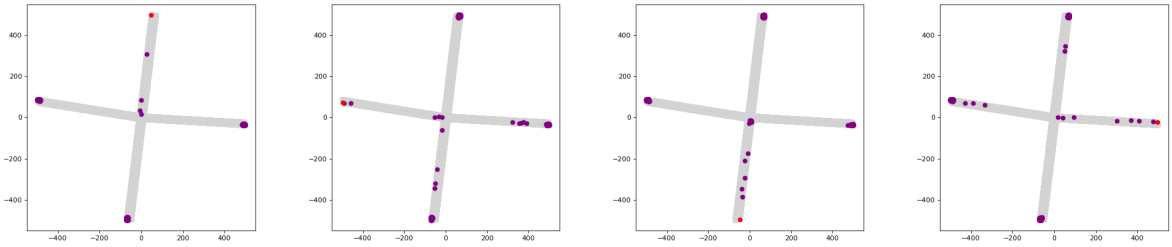


图 13　分别以 B_1,B_2,B_3,B_4 为起点的车辆运动终点集合

观察图 13 中的终点坐标可知，任一起点出发的车辆均存在左转，直行，右转三种运动情况，故本题中的车流有 12 个不同的运动方向，每个方向的车流交通由一台交通信号灯控制。为分别计算每一条路线上的交通信号灯周期，给每一条车流运动路线进行如图 14 所示的编号。

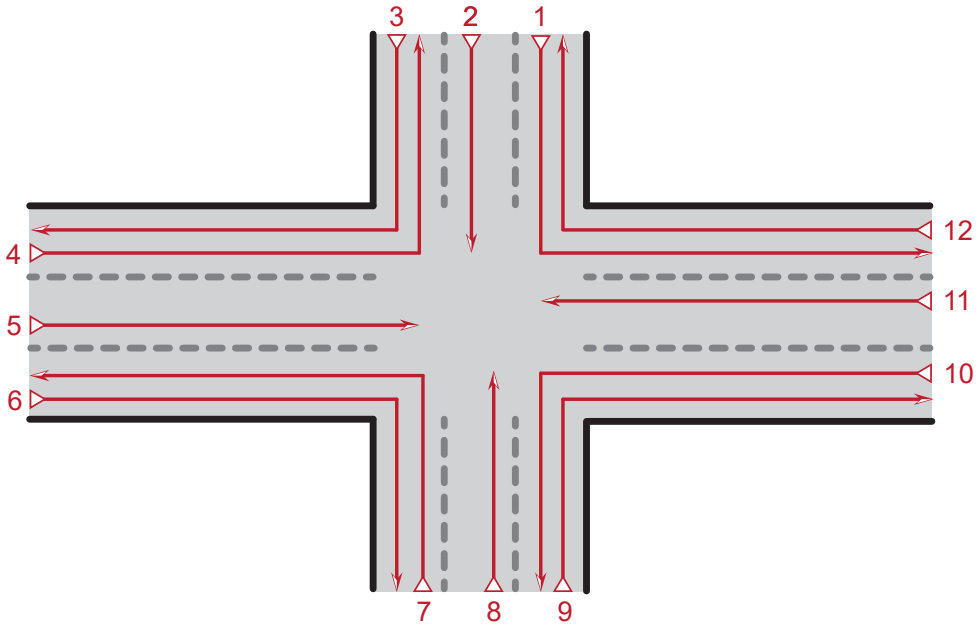


图 14　路口的 12 条道路编号

在图 14 中，每一条箭头指向即代表一种运动路径，白色三角端为车辆起点方向，箭头端为车辆在十字路口中心的转向方向，按照顺时针的顺序为每条路线进行编号，每条路线的具体方向由表 5 所示。然而，通过观察图 13 的车辆终点可知，所有从起点出发的车辆并非都行驶到了路口，且在路口处没有明显的转向趋势。由于计时周期足够长，可判断此类车辆数据为异常值，需进行进一步处理。

异常值处理　根据图 13 的样本分布情况，考虑以下两种情况的异常值：

（1）　车辆未行驶至路口处的；

（2）　车辆在路口处无转向趋势的；

表 5　12 条路线编号的车辆运动方向

编号	运动方向	编号	运动方向	编号	运动方向	编号	运动方向
1	B_1 出发左转	4	B_2 出发左转	7	B_3 出发左转	10	B_4 出发左转
2	B_1 出发直行	5	B_2 出发直行	8	B_3 出发直行	11	B_4 出发直行
3	B_1 出发右转	6	B_2 出发右转	9	B_3 出发右转	12	B_5 出发右转

对于第一种情况，由于计时周期足够车辆从起点运动到终点，故此类车辆可能中途一直停靠在路边，并未接触交通信号灯，其启停时间不具有预测交通信号灯的参考价值。本文使用式 (18) 来筛选此类异常值。

$$(x_i - x_{B_i})^2 + (y_i - x_{y_i})^2 \leq 200^2. \tag{18}$$

其中 x_i,y_i 分别为第 i 辆车终点位置的横纵坐标，x_{B_i},x_{y_i} 表示终点为 (x_i, y_i) 车辆的起点横纵坐标坐标。判定车辆的最终停点落在起点 $200m$ 距离之内的，均未接触到交通信号灯，均视为异常车辆，程序筛选此类车辆在 12 条路线上共计 12 辆。

对于第二种情况，此类车辆的终点停顿在交通信号灯路口处，没有明显的转向行进趋势，其启停时间不在相同交通信号灯的周期下，不具有参考价值。本文使用 (19) 来筛选此类异常值。

$$x_i^2 + y_i^2 \leq 200^2. \tag{19}$$

其中 x_i,y_i 分别为第 i 辆车终点位置的横纵坐标，由于路口中心坐标为 $(0, 0)$。判定终点坐标距离路口中心小于 $200m$ 的车辆为异常车辆，程序筛选此类车辆在 12 条路线上共计 16 辆。

5.6.2 车辆行进路线判定

剔除异常值后，从起点出发的车辆在路口中心处有三种行驶方向的可能：左转、直行、右转。现需根据车辆的起点坐标与终点坐标判定每辆车的行进路线。所有正常值的终点坐标如图 15 所示。

通过图 15（b）划分的阈值，可判断车辆终点的大致方向，结合其起点坐标与图 14 划分的路线编号，即可确定每一辆车所属的车流路线，具体数值如表 6 所示。

根据表 6 的划分依据，可得到 12 条路线上的汽车数量如表 7 所示。表中的路线编号所代表的实际路线位置参考图 14 。

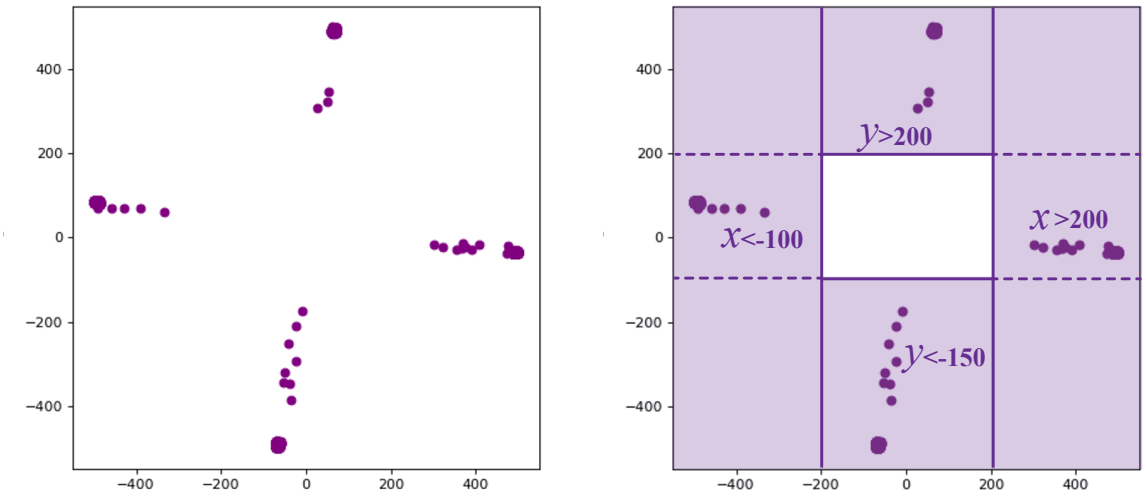


图 15　(a) 所有正常行驶车辆的终点坐标。(b) 根据终点坐标确定划分路线的阈值。

表 6　根据车辆起始坐标进行 12 条路线分流

起点坐标	终点坐标范围	线路	起点坐标	终点坐标范围	线路
$(48.27, 496.27)$	$x > 200$	1	$(-48.27, -496.27)$	$x < -100$	7
$(48.27, 496.27)$	$y < -150$	2	$(-48.27, -496.27)$	$y > 200$	8
$(48.27, 496.27)$	$x < -200$	3	$(-48.27, -496.27)$	$x > 200$	9
$(-496.22, 71.29)$	$y > 200$	4	$(495.39, -21.7)$	$y < -150$	10
$(-496.22, 71.29)$	$x < 200$	5	$(495.39, -21.7)$	$x < -100$	11
$(-496.22, 71.29)$	$y < -150$	6	$(495.39, -21.7)$	$y > 200$	12

5.6.3 DBSCAN 聚类

测算 12 条路线上每条车流每辆车的加速度阈值与速度阈值，确定每辆车的启停时刻点，然后分别对每条路线构建车流样本空间，使用 DBSCAN 对车流进行聚类，得到聚类结果如图 16 所示。

通过聚类，采用与前文相同的方式，可求得 12 条路线上各自的红灯周期与绿灯周期，如表所示。

表 7　每条路线上的汽车数量

路线编号	1	2	3	4	5	6	7	8	9	10	11	12
汽车数量	175	212	210	200	218	174	220	202	209	224	220	204

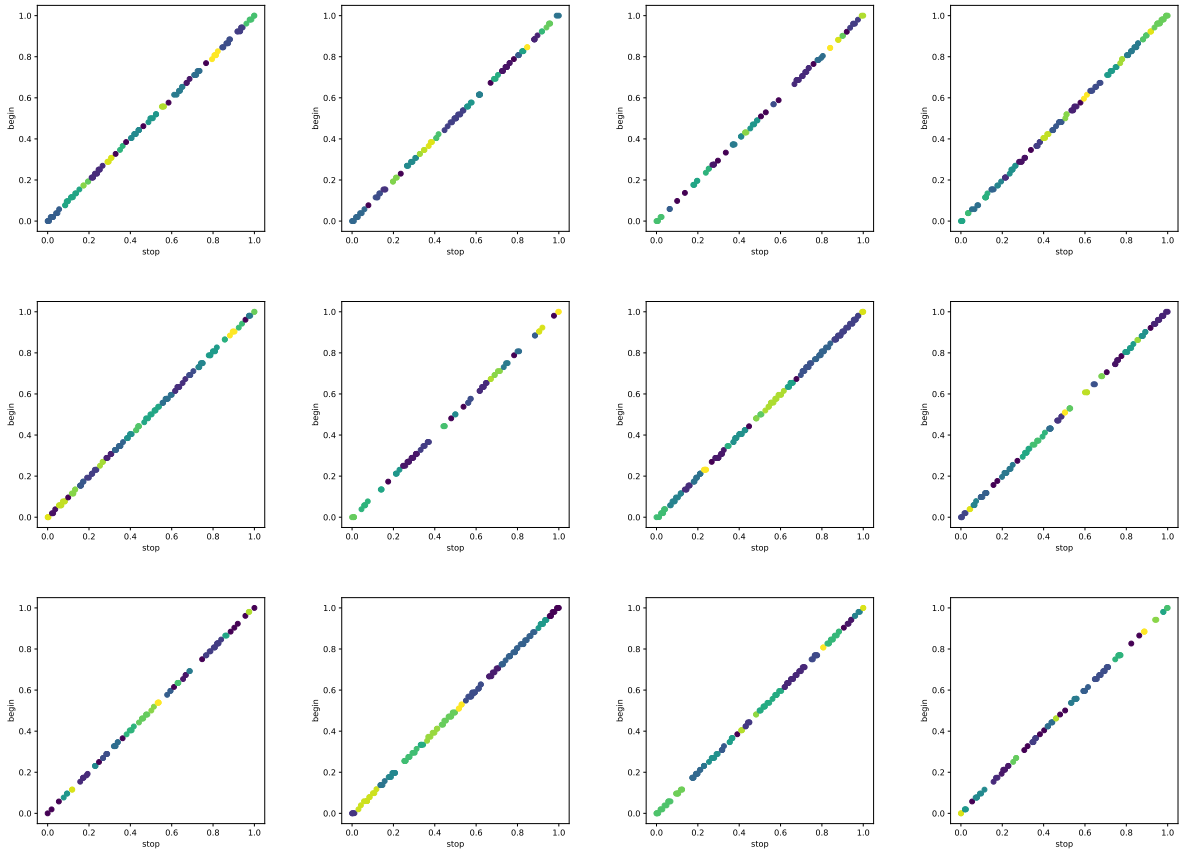


图 16　D1-D12 聚类结果

六、 模型的评价、改进与推广

6.1 模型的优点

1. 基于欧氏度量与时间序列建立数学模型，精确真实的反应红绿灯在特定时刻下的状态，结合 DESCAN 聚类算法进行问题求解，增强了模型的适用性及可信度。
2. 通过提取每辆车的运行特征来划定阈值，增强了模型的可靠性。
3. 使用贝叶斯变点检测法更好地反映变化的程度和确定性, 提高了模型的准确度。

6.2 模型的缺点

1. 第 2 问对于选取样本比率的确定，由于选取步长较长，使得结果置信度降低。

表 8　问题四交通信号灯周期

路线编号	1	2	3	4	5	6	7	8	9	10	11	12
红灯周期	100	87	76	102	93	85	109	83	76	103	97	84
绿灯周期	35	46	58	76	41	83	24	51	50	14	41	50

2. 当样本数据过多时，模型的空间复杂度较高，对于问题的求解时间较长。

6.3 模型的推广

该模型可应用于各大地图导航软件，帮助司机在陌生路段及时预测信号灯变化情况。尤其是在高拥堵城市路段堵塞较长时，若小汽车前方有大体积车辆在前方，很容易遮挡住小汽车司机与交通信号灯的视线，容易造成交通事故。因此，此模型可让司机随时掌控交通灯变换时间，提高道路驾驶的安全性。

参考文献

[1] 蔡晓桦. 网络地图应用与发展 [J]. 江西测绘,2012,(01):56-58.

[2] 葛程鹏, 赵东, 王蕊, 等. 基于改进 DBSCAN 和距离共识评估的分段点云去噪方法 [J/OL]. 系统仿真学报,1-11[2024-07-17].https://doi.org/10.16182/j.issn1004731x.joss.24-0153.

[3] 盛骤，谢式千，潘承毅. 《概率论与数理统计》. 北京：高等教育出版社，2008.6

附录 A　源代码

1.1　问题一源代码

Q1.py 运算结果

```
from utils import *
from mpl_toolkits.mplot3d import Axes3D

df = pd.read_csv('C://Users//xxx//Desktop//math//fujian//1//A1.csv')
car_ids = set(list(df['vehicle_id']))
if 472 in car_ids:
    car_ids.remove(472)
# 按顺序把stops和begins合并
def Guibing(stops, begins):
    stopsAndbegins = []
    for i in range(len(stops)):
        stopsAndbegins.append((stops[i], begins[i]))
    n = len(stopsAndbegins)
    for i in range(n):
        for j in range(0, n - i - 1):
            if stopsAndbegins[j][0] > stopsAndbegins[j + 1][0]:
                stopsAndbegins[j], stopsAndbegins[j + 1] = stopsAndbegins[j + 1],
                    stopsAndbegins[j]
    return stopsAndbegins

# 聚类
def Clustering(A1, distance=0.025):
    A1.sort()
    counts = []
    for i in A1:
        count = 0
        for j in A1:
            if math.fabs(i-j) < distance:
                count += 1

        counts.append(count)
    return counts

def get_a0(id, df):
    A = get_a(id, df)
    A1 = [x[1] for x in A if x[1] < -0.2 and x[0] < getStopTime(id, df)]
    counts = Clustering(A1)
    a0_index = counts.index(max(counts))
    a0 = A1[a0_index]

```

```
    t = A[A1.index(a0)][0]
    return a0, t

# 获得汽车的停止时间与启动时间
stops = []
stopsv = []
begins = []
for id in car_ids:
    if isStop(id, df) == 0:
        V = get_v(id, df)
        A = get_a(id, df)
        A1 = [x[1] for x in A]
        a0, t_a0 = get_a0(id, df)
        sequence = list(range(t_a0, getStopTime(id, df)+1))

        # 有效时间
        df_car = df[df['vehicle_id'] == id]
        filter_df = clearData(df_car)
        nearTimes = list(filter_df['time'])
        usefulTime = list(set(sequence) & set(nearTimes))

        a1 = []
        for i in A:
            if i[1] < a0 and i[0] in usefulTime:
                a1.append(i[1])

        if len(a1) == 0:
            stopa = '0.1'
        else:
            stopa = min(a1)

        if stopa == '0.1':
            stops.append(0)
        else:
            stops.append(A[A1.index(stopa)][0])

        # 计算begin
        last_zero_index = -1
        V1 = [x[1] for x in V]
        for index, value in enumerate(V1):
            if value == 0:
                last_zero_index = index
        begint = V[last_zero_index][0]
        begins.append(begint)


stopsAndbegins = Guibing(stops, begins)
```

```
delete = []
for i, j in enumerate(stopsAndbegins):
    if j[0] == 0:
        delete.append(i)
stopsAndbegins = [element for index, element in enumerate(stopsAndbegins) if index not in
    delete]
nStopsAndbegins = normalization(stopsAndbegins)

X = [x[0] for x in stopsAndbegins]
X2 = [x[1] for x in stopsAndbegins]
Y = range(len(X))
plt.figure(figsize=(8, 5), dpi=80)
plt.scatter(X, Y, c='red', marker='o', label='stop')
plt.scatter(X2, Y, c='green', marker='o', label='begin')
plt.xlabel('time')
plt.legend()
plt.show()

C = dbscan(nStopsAndbegins, 0.025, 2)

# 聚类图
x = []
y = []
for data in nStopsAndbegins:
    x.append(data[0])
    y.append(data[1])
plt.figure(figsize=(5, 5), dpi=80)
plt.scatter(x, y, marker='o')
plt.xlabel('stop')
plt.ylabel('begin')
plt.show()

Classes = set(C)
Classes = list(Classes)
if -1 in Classes:
    Classes.remove(-1)
begin_classes = []
stop_classes = []
for i in Classes:
    this_class_begin = []
    this_class_stop = []
    for k, j in enumerate(C):
        if j == i:
            this_class_begin.append(stopsAndbegins[k][1])
            this_class_stop.append(stopsAndbegins[k][0])

    # 当前车流的全灯周期
```

```
    this_begin = sum(this_class_begin) / len(this_class_begin)
    begin_classes.append(this_begin)

    # 当前车流的红灯周期
    stop_classes.append(this_class_begin[0] - this_class_stop[0])

# 整个红绿灯周期
begin_classes.sort()
print(begin_classes)
T = 9999
for i in range(len(begin_classes)):
    if i == 0:
        continue
    if begin_classes[i] - begin_classes[i-1] < T:
        T = begin_classes[i] - begin_classes[i-1]

T_red = -1
stop_classes.sort()
stop_classes.reverse()
for i in stop_classes:
    if i < T:
        T_red = i
        break
print(T, T_red)
```

1.2 问题二源代码

Q2-carFlow.py 划定车流量

```

from utils import *
from mpl_toolkits.mplot3d import Axes3D
from collections import Counter

df = pd.read_csv('C://Users//xxx//Desktop//math//fujian//2//B5.csv')
car_ids = set(list(df['vehicle_id']))

# 按顺序把stops和begins合并
def Guibing(stops, begins):
    stopsAndbegins = []
    for i in range(len(stops)):
        stopsAndbegins.append((stops[i], begins[i]))
    n = len(stopsAndbegins)
    for i in range(n):
        for j in range(0, n - i - 1):
            if stopsAndbegins[j][0] > stopsAndbegins[j + 1][0]:
```

```
                stopsAndbegins[j], stopsAndbegins[j + 1] = stopsAndbegins[j + 1],
                    stopsAndbegins[j]
    return stopsAndbegins

# 聚类
def Clustering(A1, distance=0.025):
    A1.sort()
    counts = []
    for i in A1:
        count = 0
        for j in A1:
            if math.fabs(i-j) < distance:
                count += 1

        counts.append(count)
    return counts

def get_a0(id, df):
    A = get_a(id, df)
    A1 = [x[1] for x in A if x[1] < -0.2 and x[0] < getStopTime(id, df)]
    counts = Clustering(A1)
    a0_index = counts.index(max(counts))
    a0 = A1[a0_index]

    t = A[A1.index(a0)][0]
    return a0, t

# 获得汽车的停止时间与启动时间
stops = []
stopsv = []
begins = []
id_stops = []
for id in car_ids:
    if isStop(id, df) == 0:
        V = get_v(id, df)
        A = get_a(id, df)
        A1 = [x[1] for x in A]
        a0, t_a0 = get_a0(id, df)
        sequence = list(range(t_a0, getStopTime(id, df)+1))

        # 有效时间
        df_car = df[df['vehicle_id'] == id]
        filter_df = clearData(df_car)
        nearTimes = list(filter_df['time'])
        usefulTime = list(set(sequence) & set(nearTimes))

        a1 = []
```

```
        for i in A:
            if i[1] < a0 and i[0] in usefulTime:
                a1.append(i[1])

        if len(a1) == 0:
            stopa = '0.1'
        else:
            stopa = min(a1)

        if stopa == '0.1':
            stops.append(0)
        else:
            stops.append(A[A1.index(stopa)][0])
            id_stops.append((id, A[A1.index(stopa)][0]))

        # 计算begin
        last_zero_index = -1
        V1 = [x[1] for x in V]
        for index, value in enumerate(V1):
            if value == 0:
                last_zero_index = index
        begint = V[last_zero_index][0]
        begins.append(begint)

stopsAndbegins = Guibing(stops, begins)
delete = []
for i, j in enumerate(stopsAndbegins):
    if j[0] == 0:
        delete.append(i)
stopsAndbegins = [element for index, element in enumerate(stopsAndbegins) if index not in
    delete]
nStopsAndbegins = normalization(stopsAndbegins)

C = dbscan(nStopsAndbegins, 0.025, 1)
print(C)
lowCar = []
lowClass = []
midCar = []
midClass = []
highCar = []
highClass = []
classNumber = max(C)
for i in range(classNumber+1):
    num = C.count(i)

    if num <= 2:
        lowClass.append(i)
```

```
    elif num == 3:
        midClass.append(i)
    else:
        highClass.append(i)

for j, i in enumerate(C):

    id = 0

    stop = stopsAndbegins[j][0]
    for k in id_stops:
        if stop == k[1]:
            id = k[0]

    if i in lowClass:
        lowCar.append(id)
    elif i in midClass:
        midCar.append(id)
    elif i in highClass:
        highCar.append(id)

print(lowCar)
print(midCar)
print(highCar)
```

Q2-carFlow2.py 计算车流量影响

```

from utils import *
from mpl_toolkits.mplot3d import Axes3D
file = 'A2'
df = pd.read_csv('C://Users//xxx//Desktop//math//fujian//1//' + file + '.csv')
car_ids = set(list(df['vehicle_id']))

if file == 'A1':
    lowCar = [46, 158, 162, 197, 284, 418, 446, 464, 486, 506, 620, 640, 668, 682, 898, 905,
        956, 993, 1027, 1037, 1067, 1149]
    midCar = [99, 101, 102, 125, 123, 134, 225, 226, 236, 347, 359, 367, 766, 776, 775, 829,
        840, 839, 869, 874, 880]
    highCar = [539, 546, 550, 552]
elif file == 'A2':
    lowCar = [11, 100, 104, 129, 191, 220, 223, 463, 470, 522, 538, 564, 586, 623, 713, 749,
        751, 781, 809, 865, 961, 1015, 1057, 1073, 1111, 1121, 1144]
    midCar = []
    highCar = []
    car_ids.remove(472)
elif file == 'A3':
```

```
    lowCar = [53, 64, 154, 156, 190, 325, 328, 388, 435, 447, 482, 565, 578, 603, 662, 701,
        702, 762, 765, 805, 903, 919, 982, 1140, 1150]
    midCar = [537, 538, 548, 631, 633, 643, 853, 860, 861, 1078, 1093, 1096]
    highCar = [287, 289, 293, 295]
elif file == 'A4':
    lowCar = [20, 119, 211, 301, 340, 369, 423, 482, 484, 521, 532, 556, 561, 604, 620, 638,
        692, 797, 860, 862, 950, 1072, 1097, 1103, 1137, 1162, 1165]
    midCar = [579, 582, 591, 970, 971, 983, 1038, 1041, 1046]
    highCar = [232, 240, 249, 254, 448, 450, 453, 468]
elif file == 'A5':
    lowCar = [84, 87, 113, 123, 175, 266, 428, 429, 449, 452, 478, 603, 601, 708, 758, 785,
        796, 848, 908, 942, 955, 974, 972, 998, 1061, 1066, 1096, 1105, 1157, 1167]
    midCar = [1118, 1124, 1135]
    highCar = []

# 按顺序把stops和begins合并
def Guibing(stops, begins):
    stopsAndbegins = []
    for i in range(len(stops)):
        stopsAndbegins.append((stops[i], begins[i]))
    n = len(stopsAndbegins)
    for i in range(n):
        for j in range(0, n - i - 1):
            if stopsAndbegins[j][0] > stopsAndbegins[j + 1][0]:
                stopsAndbegins[j], stopsAndbegins[j + 1] = stopsAndbegins[j + 1],
                    stopsAndbegins[j]
    return stopsAndbegins

# 聚类
def Clustering(A1, distance=0.025):
    A1.sort()
    counts = []
    for i in A1:
        count = 0
        for j in A1:
            if math.fabs(i-j) < distance:
                count += 1

        counts.append(count)
    return counts

def get_a0(id, df):
    A = get_a(id, df)
    A1 = [x[1] for x in A if x[1] < -0.2 and x[0] < getStopTime(id, df)]
    counts = Clustering(A1)
    print(A1)
    a0_index = counts.index(max(counts))
```

```
    a0 = A1[a0_index]

    t = A[A1.index(a0)][0]
    return a0, t

# 获得汽车的停止时间与启动时间
stops = []
stopsv = []
begins = []
for id in car_ids:
    if isStop(id, df) == 0:
        V = get_v(id, df)
        A = get_a(id, df)
        A1 = [x[1] for x in A]
        print(id)
        a0, t_a0 = get_a0(id, df)
        sequence = list(range(t_a0, getStopTime(id, df)+1))

        # 有效时间
        df_car = df[df['vehicle_id'] == id]
        if id in lowCar:
            filter_df = Q2_clearData(df_car, 75)
        elif id in midCar:
            filter_df = Q2_clearData(df_car, 100)
        elif id in highCar:
            filter_df = Q2_clearData(df_car, 125)
        else:
            continue
        nearTimes = list(filter_df['time'])
        usefulTime = list(set(sequence) & set(nearTimes))

        a1 = []
        for i in A:
            if i[1] < a0 and i[0] in usefulTime:
                a1.append(i[1])

        if len(a1) == 0:
            stopa = '0.1'
        else:
            stopa = min(a1)

        if stopa == '0.1':
            stops.append(0)
        else:
            stops.append(A[A1.index(stopa)][0])

        # 计算begin
```

```
        last_zero_index = -1
        V1 = [x[1] for x in V]
        for index, value in enumerate(V1):
            if value == 0:
                last_zero_index = index
        begint = V[last_zero_index][0]
        begins.append(begint)


stopsAndbegins = Guibing(stops, begins)
delete = []
for i, j in enumerate(stopsAndbegins):
    if j[0] == 0:
        delete.append(i)
stopsAndbegins = [element for index, element in enumerate(stopsAndbegins) if index not in
    delete]
nStopsAndbegins = normalization(stopsAndbegins)

C = dbscan(nStopsAndbegins, 0.025, 1)

# 聚类图
# x = []
# y = []
# for data in nStopsAndbegins:
#     x.append(data[0])
#     y.append(data[1])
# plt.figure(figsize=(5, 5), dpi=80)
# plt.scatter(x, y, c=C, marker='o')
# plt.xlabel('stop')
# plt.ylabel('begin')
# plt.show()

Classes = set(C)
Classes = list(Classes)
if -1 in Classes:
    Classes.remove(-1)
begin_classes = []
stop_classes = []
for i in Classes:
    this_class_begin = []
    this_class_stop = []
    for k, j in enumerate(C):
        if j == i:
            this_class_begin.append(stopsAndbegins[k][1])
            this_class_stop.append(stopsAndbegins[k][0])

    # 当前车流的全灯周期
```

```
    this_begin = sum(this_class_begin) / len(this_class_begin)
    begin_classes.append(this_begin)

    # 当前车流的红灯周期
    stop_classes.append(this_class_begin[0] - this_class_stop[0])

# 整个红绿灯周期
begin_classes.sort()
T = 9999
for i in range(len(begin_classes)):
    if i == 0:
        continue
    if begin_classes[i] - begin_classes[i-1] < T:
        T = begin_classes[i] - begin_classes[i-1]

T_red = max(stop_classes)

print(stop_classes)
print(T, T_red)
# plt.figure(figsize=(8, 6))
# plt.hist(stop_classes, bins=13, edgecolor='black') # bins参数指定分成的区间数
# plt.xlabel('Value')
# plt.ylabel('Frequency')
# plt.title('Histogram of Data')
# plt.grid(True)
# plt.tight_layout()
# plt.show()
```

Q2-车辆比误差

```

from utils import *
from mpl_toolkits.mplot3d import Axes3D

# 按顺序把stops和begins合并
def Guibing(stops, begins):
    stopsAndbegins = []
    for i in range(len(stops)):
        stopsAndbegins.append((stops[i], begins[i]))
    n = len(stopsAndbegins)
    for i in range(n):
        for j in range(0, n - i - 1):
            if stopsAndbegins[j][0] > stopsAndbegins[j + 1][0]:
                stopsAndbegins[j], stopsAndbegins[j + 1] = stopsAndbegins[j + 1],
                    stopsAndbegins[j]
    return stopsAndbegins

```

```
# 聚类
def Clustering(A1, distance=0.025):
    A1.sort()

    counts = []
    for i in A1:
        count = 0
        for j in A1:
            if math.fabs(i-j) < distance:
                count += 1

        counts.append(count)
    return counts

def get_a0(id, df):
    A = get_a(id, df)
    A1 = [x[1] for x in A if x[1] < -0.2 and x[0] < getStopTime(id, df)]
    counts = Clustering(A1)
    a0_index = counts.index(max(counts))
    a0 = A1[a0_index]

    t = A[A1.index(a0)][0]
    return a0, t

end_result1 = []
end_result2 = []
for P in np.arange(0.3, 1, 0.025):
    results1 = []
    results2 = []
    for i in range(20):
        df = pd.read_csv('C://Users//xxx//Desktop//math//fujian//1//A5.csv')
        car_ids = set(list(df['vehicle_id']))
        car_ids = list(car_ids)
        if 472 in car_ids:
            car_ids.remove(472)
        num = int(len(car_ids) * P)
        car_ids = random.sample(car_ids, num)
        stops = []
        stopsv = []
        begins = []
        for id in car_ids:
            if isStop(id, df) == 0:
                V = get_v(id, df)
                A = get_a(id, df)
                A1 = [x[1] for x in A]
                a0, t_a0 = get_a0(id, df)
                sequence = list(range(t_a0, getStopTime(id, df) + 1))
```

```
            # 有效时间
            df_car = df[df['vehicle_id'] == id]
            filter_df = clearData(df_car)
            nearTimes = list(filter_df['time'])
            usefulTime = list(set(sequence) & set(nearTimes))

            a1 = []
            for i in A:
                if i[1] < a0 and i[0] in usefulTime:
                    a1.append(i[1])

            if len(a1) == 0:
                stopa = '0.1'
            else:
                stopa = min(a1)

            if stopa == '0.1':
                stops.append(0)
            else:
                stops.append(A[A1.index(stopa)][0])

            # 计算begin
            last_zero_index = -1
            V1 = [x[1] for x in V]
            for index, value in enumerate(V1):
                if value == 0:
                    last_zero_index = index
            begint = V[last_zero_index][0]
            begins.append(begint)

    stopsAndbegins = Guibing(stops, begins)
    delete = []
    for i, j in enumerate(stopsAndbegins):
        if j[0] == 0:
            delete.append(i)
    stopsAndbegins = [element for index, element in enumerate(stopsAndbegins) if index not
        in delete]
    nStopsAndbegins = normalization(stopsAndbegins)

    C = dbscan(nStopsAndbegins, 0.025, 1)

    # 聚类图
    # x = []
    # y = []
    # for data in nStopsAndbegins:
    #     x.append(data[0])
```

```
        #     y.append(data[1])
        # plt.figure(figsize=(5, 5), dpi=80)
        # plt.scatter(x, y, c=C, marker='o')
        # plt.xlabel('stop')
        # plt.ylabel('begin')
        # plt.show()

        Classes = set(C)
        Classes = list(Classes)
        if -1 in Classes:
            Classes.remove(-1)
        begin_classes = []
        stop_classes = []
        for i in Classes:
            this_class_begin = []
            this_class_stop = []
            for k, j in enumerate(C):
                if j == i:
                    this_class_begin.append(stopsAndbegins[k][1])
                    this_class_stop.append(stopsAndbegins[k][0])

            # 当前车流的全灯周期
            this_begin = sum(this_class_begin) / len(this_class_begin)
            begin_classes.append(this_begin)

            # 当前车流的红灯周期
            stop_classes.append(this_class_begin[0] - this_class_stop[0])

        # 整个红绿灯周期
        begin_classes.sort()
        T = 9999
        for i in range(len(begin_classes)):
            if i == 0:
                continue
            if begin_classes[i] - begin_classes[i - 1] < T:
                T = begin_classes[i] - begin_classes[i - 1]

        T_red = max(stop_classes)
        results1.append(T)
        results2.append(T_red)
    end_result1.append(sum(results1) / len(results1))
    end_result2.append(sum(results2) / len(results2))

for i in end_result1:
    print(i)

for i in end_result2:
```

```
    print(i)
```

Q2-定位误差.py

```

from utils import *

df = pd.read_csv('C://Users\xxx\Desktop\math\data\Q2-xy\A5.csv')
car_ids = set(list(df['vehicle_id']))

# 按顺序把stops和begins合并
def Guibing(stops, begins):
    stopsAndbegins = []
    for i in range(len(stops)):
        stopsAndbegins.append((stops[i], begins[i]))
    n = len(stopsAndbegins)
    for i in range(n):
        for j in range(0, n - i - 1):
            if stopsAndbegins[j][0] > stopsAndbegins[j + 1][0]:
                stopsAndbegins[j], stopsAndbegins[j + 1] = stopsAndbegins[j + 1],
                    stopsAndbegins[j]
    return stopsAndbegins

# 聚类
def Clustering(A1, distance=0.025):
    A1.sort()
    counts = []
    for i in A1:
        count = 0
        for j in A1:
            if math.fabs(i-j) < distance:
                count += 1

        counts.append(count)
    return counts

def get_a0(id, df):
    A = get_a(id, df)
    A1 = [x[1] for x in A if x[1] < -0.2 and x[0] < getStopTime(id, df)]
    counts = Clustering(A1)
    a0_index = counts.index(max(counts))
    a0 = A1[a0_index]

    t = A[A1.index(a0)][0]
    return a0, t

# 获得汽车的停止时间与启动时间
```

```
stops = []
stopsv = []
begins = []
for id in car_ids:
    if isStop(id, df) == 0:
        V = get_v(id, df)
        A = get_a(id, df)
        A1 = [x[1] for x in A]
        a0, t_a0 = get_a0(id, df)
        sequence = list(range(t_a0, getStopTime(id, df)+1))

        # 有效时间
        df_car = df[df['vehicle_id'] == id]
        filter_df = clearData(df_car)
        nearTimes = list(filter_df['time'])
        usefulTime = list(set(sequence) & set(nearTimes))

        a1 = []
        for i in A:
            if i[1] < a0 and i[0] in usefulTime:
                a1.append(i[1])

        if len(a1) == 0:
            stopa = '0.1'
        else:
            stopa = min(a1)

        if stopa == '0.1':
            stops.append(0)
        else:
            stops.append(A[A1.index(stopa)][0])

        # 计算begin
        last_zero_index = -1
        V1 = [x[1] for x in V]
        for index, value in enumerate(V1):
            if value == 0:
                last_zero_index = index
        begint = V[last_zero_index][0]
        begins.append(begint)

stopsAndbegins = Guibing(stops, begins)
delete = []
for i, j in enumerate(stopsAndbegins):
    if j[0] == 0:
        delete.append(i)
stopsAndbegins = [element for index, element in enumerate(stopsAndbegins) if index not in
```

```
    delete]
nStopsAndbegins = normalization(stopsAndbegins)

C = dbscan(nStopsAndbegins, 0.025, 2)
print(C)

# 聚类图
# x = []
# y = []
# for data in nStopsAndbegins:
#     x.append(data[0])
#     y.append(data[1])
# plt.figure(figsize=(5, 5), dpi=80)
# plt.scatter(x, y, c=C, marker='o')
# plt.xlabel('stop')
# plt.ylabel('begin')
# plt.show()

Classes = set(C)
Classes = list(Classes)
if -1 in Classes:
    Classes.remove(-1)
begin_classes = []
stop_classes = []
for i in Classes:
    this_class_begin = []
    this_class_stop = []
    for k, j in enumerate(C):
        if j == i:
            this_class_begin.append(stopsAndbegins[k][1])
            this_class_stop.append(stopsAndbegins[k][0])

    # 当前车流的全灯周期
    this_begin = sum(this_class_begin) / len(this_class_begin)
    begin_classes.append(this_begin)

    # 当前车流的红灯周期
    stop_classes.append(this_class_begin[0] - this_class_stop[0])

# 整个红绿灯周期
begin_classes.sort()
T = 9999
for i in range(len(begin_classes)):
    if i == 0:
        continue
    if begin_classes[i] - begin_classes[i-1] < T:
        T = begin_classes[i] - begin_classes[i-1]
```

```

T_red = max(stop_classes)

print(T, T_red)
# plt.figure(figsize=(8, 6))
# plt.hist(stop_classes, bins=13, edgecolor='black') # bins参数指定分成的区间数
# plt.xlabel('Value')
# plt.ylabel('Frequency')
# plt.title('Histogram of Data')
# plt.grid(True)
# plt.tight_layout()
# plt.show()
```

Q2-高斯分布加噪.py

```
import pandas as pd
import numpy as np

# 读取Excel文件vv
file_path = 'C://Users\xxx\Desktop\math\data\Q4-xy\D.csv' # 替换为你的Excel文件路径
df = pd.read_csv(file_path)
car_ids = set(list(df['vehicle_id']))

# 遍历每一行
for index, row in df.iterrows():
    # 在这里进行数据处理
    # 例如，将某一列的数据乘以2
    if df.at[index, 'x'] not in [-18.21, 8.55, 18.65, 18.46, -0.09, 0.24, 25.95, -25.62,
        -19.22, 1.13, -18.72, -8.4, 18.84, 9.44, -0.98, -9.29, -6.11, -1.88, 26.33, 6.26,
        -33.02, -26.63, -7.5, -34.03, 33.81, -2.77, 33.43, 26.14, 7.65, 2.03, 3.08, -41.44,
        -2.93, 41.3, -26.13, 40.92, 2.92, 6.76, -40.43, -6.61, 48.79, -3.66, -5.22, 33.62,
        -47.84, 48.41, 5.37, -4.56, -48.85, 56.28, 5.87, -55.24, 55.9, 3.81, -56.25, 63.39,
        -5.72, 63.77, -62.65, -33.53, 4.97, -5.45, 4.71, -2.04, 71.25, 17.46, -77.46, 41.11,
        -70.06, 32.44, 54.9, 62.39, 17.84, 32.82, 40.3, -63.66, 3.18, -18.23, 2.19, -4.32, 5.6,
        4.08, 24.95, 47.41, -9.4, 69.87, 77.36, 84.85, -71.07, -78.47, -6.35, -4.82, -25.64,
        -84.87, -0.96, -92.28, -1.86, -2.75, -3.65, -5.43, 78.36, 1.01, -3.93, 25.33, 47.79,
        55.28, 62.77, -33.04, -18.32, 6.5, 17.82, 25.31, 32.8]:
        df.at[index, 'x'] = row['x'] * (1 + np.random.normal(0, 0.0025))
        df.at[index, 'y'] = row['y'] * (1 + np.random.normal(0, 0.0025))
    else:
        pass

    # 将处理后的数据写回原Excel文件
df.to_csv(file_path, index=False)
```

Q2-综合因素.py 处理 A 数据

```
1
2  from utils import *
3  from mpl_toolkits.mplot3d import Axes3D
4
5
6  # 按顺序把stops和begins合并
7  def Guibing(stops, begins):
8      stopsAndbegins = []
9      for i in range(len(stops)):
10         stopsAndbegins.append((stops[i], begins[i]))
11     n = len(stopsAndbegins)
12     for i in range(n):
13         for j in range(0, n - i - 1):
14             if stopsAndbegins[j][0] > stopsAndbegins[j + 1][0]:
15                 stopsAndbegins[j], stopsAndbegins[j + 1] = stopsAndbegins[j + 1],
                       stopsAndbegins[j]
16     return stopsAndbegins
17
18 # 聚类
19 def Clustering(A1, distance=0.025):
20     A1.sort()
21     counts = []
22     for i in A1:
23         count = 0
24         for j in A1:
25             if math.fabs(i-j) < distance:
26                 count += 1
27
28         counts.append(count)
29     return counts
30
31 def get_a0(id, df):
32     A = get_a(id, df)
33     A1 = [x[1] for x in A if x[1] < -0.2 and x[0] < getStopTime(id, df)]
34     counts = Clustering(A1)
35     a0_index = counts.index(max(counts))
36     a0 = A1[a0_index]
37
38     t = A[A1.index(a0)][0]
39     return a0, t
40
41 end_result1 = []
42 end_result2 = []
43 for P in np.arange(0.3, 1, 0.025):
44     results1 = []
45     results2 = []
```

```
    for item in range(20):
        file = 'A4'
        df = pd.read_csv('C://Users//xxx//Desktop//math//data//Q2-xy//' + file + '.csv')
        car_ids = set(list(df['vehicle_id']))
        car_ids = list(car_ids)
        num = int(len(car_ids) * P)
        car_ids = random.sample(car_ids, num)
        if file == 'A1':
            lowCar = [46, 158, 162, 197, 284, 418, 446, 464, 486, 506, 620, 640, 668, 682, 898,
                905, 956, 993, 1027,
                    1037, 1067, 1149]
            midCar = [99, 101, 102, 125, 123, 134, 225, 226, 236, 347, 359, 367, 766, 776, 775,
                829, 840, 839, 869, 874,
                    880]
            highCar = [539, 546, 550, 552]
        elif file == 'A2':
            lowCar = [11, 100, 104, 129, 191, 220, 223, 463, 470, 522, 538, 564, 586, 623, 713,
                749, 751, 781, 809, 865,
                    961, 1015, 1057, 1073, 1111, 1121, 1144]
            midCar = []
            highCar = []
            if 472 in car_ids:
                car_ids.remove(472)
        elif file == 'A3':
            lowCar = [53, 64, 154, 156, 190, 325, 328, 388, 435, 447, 482, 565, 578, 603, 662,
                701, 702, 762, 765, 805,
                    903,
                    919, 982, 1140, 1150]
            midCar = [537, 538, 548, 631, 633, 643, 853, 860, 861, 1078, 1093, 1096]
            highCar = [287, 289, 293, 295]
        elif file == 'A4':
            lowCar = [20, 119, 211, 301, 340, 369, 423, 482, 484, 521, 532, 556, 561, 604, 620,
                638, 692, 797, 860, 862,
                    950, 1072, 1097, 1103, 1137, 1162, 1165]
            midCar = [579, 582, 591, 970, 971, 983, 1038, 1041, 1046]
            highCar = [232, 240, 249, 254, 448, 450, 453, 468]
        elif file == 'A5':
            lowCar = [84, 87, 113, 123, 175, 266, 428, 429, 449, 452, 478, 603, 601, 708, 758,
                785, 796, 848, 908, 942,
                    955,
                    974, 972, 998, 1061, 1066, 1096, 1105, 1157, 1167]
            midCar = [1118, 1124, 1135]
            highCar = []

        # 获得汽车的停止时间与启动时间
        stops = []
        stopsv = []
```

```
    begins = []
    for id in car_ids:
        if isStop(id, df) == 0:
            V = get_v(id, df)
            A = get_a(id, df)
            A1 = [x[1] for x in A]
            a0, t_a0 = get_a0(id, df)
            sequence = list(range(t_a0, getStopTime(id, df) + 1))

            # 有效时间
            df_car = df[df['vehicle_id'] == id]
            if id in lowCar:
                filter_df = Q2_clearData(df_car, 75)
            elif id in midCar:
                filter_df = Q2_clearData(df_car, 100)
            elif id in highCar:
                filter_df = Q2_clearData(df_car, 125)
            else:
                continue
            nearTimes = list(filter_df['time'])
            usefulTime = list(set(sequence) & set(nearTimes))

            a1 = []
            for i in A:
                if i[1] < a0 and i[0] in usefulTime:
                    a1.append(i[1])

            if len(a1) == 0:
                stopa = '0.1'
            else:
                stopa = min(a1)

            if stopa == '0.1':
                stops.append(0)
            else:
                stops.append(A[A1.index(stopa)][0])

            # 计算begin
            last_zero_index = -1
            V1 = [x[1] for x in V]
            for index, value in enumerate(V1):
                if value == 0:
                    last_zero_index = index
            begint = V[last_zero_index][0]
            begins.append(begint)

    stopsAndbegins = Guibing(stops, begins)
```

```
        delete = []
        for i, j in enumerate(stopsAndbegins):
            if j[0] == 0:
                delete.append(i)
        stopsAndbegins = [element for index, element in enumerate(stopsAndbegins) if index not
            in delete]
        nStopsAndbegins = normalization(stopsAndbegins)

        C = dbscan(nStopsAndbegins, 0.025, 1)

        Classes = set(C)
        Classes = list(Classes)
        if -1 in Classes:
            Classes.remove(-1)
        begin_classes = []
        stop_classes = []
        for i in Classes:
            this_class_begin = []
            this_class_stop = []
            for k, j in enumerate(C):
                if j == i:
                    this_class_begin.append(stopsAndbegins[k][1])
                    this_class_stop.append(stopsAndbegins[k][0])

            # 当前车流的全灯周期
            this_begin = sum(this_class_begin) / len(this_class_begin)
            begin_classes.append(this_begin)

            # 当前车流的红灯周期
            stop_classes.append(this_class_begin[0] - this_class_stop[0])

        # 整个红绿灯周期
        begin_classes.sort()
        T = 9999
        for i in range(len(begin_classes)):
            if i == 0:
                continue
            if begin_classes[i] - begin_classes[i - 1] < T:
                T = begin_classes[i] - begin_classes[i - 1]

        T_red = max(stop_classes)
        results1.append(T)
        results2.append(T_red)

    end_result1.append(sum(results1) / len(results1))
    end_result2.append(sum(results2) / len(results2))

```

```
for i in end_result1:
    print(i)

for i in end_result2:
    print(i)
```

Q2-综合因素 D2.py 处理 B 数据

```

from utils import *
from mpl_toolkits.mplot3d import Axes3D


# 按顺序把stops和begins合并
def Guibing(stops, begins):
    stopsAndbegins = []
    for i in range(len(stops)):
        stopsAndbegins.append((stops[i], begins[i]))
    n = len(stopsAndbegins)
    for i in range(n):
        for j in range(0, n - i - 1):
            if stopsAndbegins[j][0] > stopsAndbegins[j + 1][0]:
                stopsAndbegins[j], stopsAndbegins[j + 1] = stopsAndbegins[j + 1],
                    stopsAndbegins[j]
    return stopsAndbegins

# 聚类
def Clustering(A1, distance=0.025):
    A1.sort()
    counts = []
    for i in A1:
        count = 0
        for j in A1:
            if math.fabs(i-j) < distance:
                count += 1

        counts.append(count)
    return counts

def get_a0(id, df):
    A = get_a(id, df)
    A1 = [x[1] for x in A if x[1] < -0.2 and x[0] < getStopTime(id, df)]
    counts = Clustering(A1)
    a0_index = counts.index(max(counts))
    a0 = A1[a0_index]

    t = A[A1.index(a0)][0]
```

```
    return a0, t

results1 = []
results2 = []
for item in range(20):
    file = 'B3'
    df = pd.read_csv('C://Users//xxx//Desktop//math//data//Q2-xy//' + file + '.csv')
    car_ids = set(list(df['vehicle_id']))
    car_ids = list(car_ids)
    num = int(len(car_ids) * 0.575)
    car_ids = random.sample(car_ids, num)
    # if file == 'B1':
    #     lowCar = [16, 20, 45, 93, 139, 170, 330, 554, 657, 759, 847, 1015, 1051, 1068, 1118]
    #     midCar = [238, 245, 246, 717, 721, 731]
    #     highCar = []
    # elif file == 'B2':
    #     lowCar = [54, 122, 168, 204, 344, 376, 385, 434, 436, 468, 549, 774, 811, 812, 851,
        887, 928, 972, 1014, 1050,
    #             1063, 1147]
    #     midCar = [72, 85, 93, 501, 499, 505]
    #     highCar = [283, 291, 297, 307]
    #     if 472 in car_ids:
    #         car_ids.remove(472)
    # elif file == 'B3':
    #     lowCar = [230, 235, 718, 839, 881]
    #     midCar = []
    #     highCar = []
    # elif file == 'B4':
    #     lowCar = [138, 299, 337, 374, 404, 459, 496, 520, 677, 689, 744, 779, 819, 851, 882,
        951, 1040, 1098, 1130,
    #             1135]
    #     midCar = []
    #     highCar = []
    # elif file == 'B5':
    #     lowCar = [55, 108, 140, 195, 273, 311, 390, 455, 461, 589, 664, 697, 707, 727, 751,
        797, 824, 907, 916, 942,
    #             947, 981, 994, 1073, 1153, 1154]
    #     midCar = []
    #     highCar = []

    # 获得汽车的停止时间与启动时间
    stops = []
    stopsv = []
    begins = []
    for id in car_ids:
        if isStop(id, df) == 0:
            V = get_v(id, df)
```

```
            A = get_a(id, df)
            A1 = [x[1] for x in A]
            a0, t_a0 = get_a0(id, df)
            sequence = list(range(t_a0, getStopTime(id, df) + 1))

            # 有效时间
            df_car = df[df['vehicle_id'] == id]
            # if id in lowCar:
            #     filter_df = Q2_clearData(df_car, 75)
            # elif id in midCar:
            #     filter_df = Q2_clearData(df_car, 100)
            # elif id in highCar:
            #     filter_df = Q2_clearData(df_car, 125)
            # else:
            #     continue
            filter_df = Q2_clearData(df_car, 100)
            nearTimes = list(filter_df['time'])
            usefulTime = list(set(sequence) & set(nearTimes))

            a1 = []
            for i in A:
                if i[1] < a0 and i[0] in usefulTime:
                    a1.append(i[1])

            if len(a1) == 0:
                stopa = '0.1'
            else:
                stopa = min(a1)

            if stopa == '0.1':
                stops.append(0)
            else:
                stops.append(A[A1.index(stopa)][0])

            # 计算begin
            last_zero_index = -1
            V1 = [x[1] for x in V]
            for index, value in enumerate(V1):
                if value == 0:
                    last_zero_index = index
            begint = V[last_zero_index][0]
            begins.append(begint)

    stopsAndbegins = Guibing(stops, begins)
    delete = []
    for i, j in enumerate(stopsAndbegins):
        if j[0] == 0:
```

```
            delete.append(i)
    stopsAndbegins = [element for index, element in enumerate(stopsAndbegins) if index not in
        delete]
    nStopsAndbegins = normalization(stopsAndbegins)

    C = dbscan(nStopsAndbegins, 0.025, 1)

    Classes = set(C)
    Classes = list(Classes)
    if -1 in Classes:
        Classes.remove(-1)
    begin_classes = []
    stop_classes = []
    for i in Classes:
        this_class_begin = []
        this_class_stop = []
        for k, j in enumerate(C):
            if j == i:
                this_class_begin.append(stopsAndbegins[k][1])
                this_class_stop.append(stopsAndbegins[k][0])

        # 当前车流的全灯周期
        this_begin = sum(this_class_begin) / len(this_class_begin)
        begin_classes.append(this_begin)

        # 当前车流的红灯周期
        stop_classes.append(this_class_begin[0] - this_class_stop[0])

    # 整个红绿灯周期
    begin_classes.sort()
    T = 9999
    for i in range(len(begin_classes)):
        if i == 0:
            continue
        if begin_classes[i] - begin_classes[i - 1] < T:
            T = begin_classes[i] - begin_classes[i - 1]

    T_red = max(stop_classes)
    results1.append(T)
    results2.append(T_red)

print(sum(results1) / len(results1))
print(sum(results2) / len(results2))
```

1.3 问题三源代码

```python
import matplotlib.pyplot as plt

from utils import *
from mpl_toolkits.mplot3d import Axes3D


# 按顺序把stops和begins合并
def Guibing(stops, begins):
    stopsAndbegins = []
    for i in range(len(stops)):
        stopsAndbegins.append((stops[i], begins[i]))
    n = len(stopsAndbegins)
    for i in range(n):
        for j in range(0, n - i - 1):
            if stopsAndbegins[j][0] > stopsAndbegins[j + 1][0]:
                stopsAndbegins[j], stopsAndbegins[j + 1] = stopsAndbegins[j + 1],
                    stopsAndbegins[j]
    return stopsAndbegins

# 聚类
def Clustering(A1, distance=0.025):
    A1.sort()
    counts = []
    for i in A1:
        count = 0
        for j in A1:
            if math.fabs(i-j) < distance:
                count += 1

        counts.append(count)
    return counts

def get_a0(id, df):
    A = get_a(id, df)
    A1 = [x[1] for x in A if x[1] < -0.2 and x[0] < getStopTime(id, df)]
    counts = Clustering(A1)
    a0_index = counts.index(max(counts))
    a0 = A1[a0_index]

    t = A[A1.index(a0)][0]
    return a0, t

results1 = []
results2 = []

for item in range(1):
```

```
    file = 'C6'
    df = pd.read_csv('C://Users//xxx//Desktop//math//data//Q3-xy//' + file + '.csv')
    car_ids = set(list(df['vehicle_id']))
    car_ids = list(car_ids)
    num = int(len(car_ids) * 1)
    car_ids = random.sample(car_ids, num)
    # if file == 'A1':
    #     lowCar = [46, 158, 162, 197, 284, 418, 446, 464, 486, 506, 620, 640, 668, 682, 898,
        905, 956, 993, 1027,
    #               1037, 1067, 1149]
    #     midCar = [99, 101, 102, 125, 123, 134, 225, 226, 236, 347, 359, 367, 766, 776, 775,
        829, 840, 839, 869, 874,
    #               880]
    #     highCar = [539, 546, 550, 552]
    # elif file == 'A2':
    #     lowCar = [11, 100, 104, 129, 191, 220, 223, 463, 470, 522, 538, 564, 586, 623, 713,
        749, 751, 781, 809, 865,
    #               961, 1015, 1057, 1073, 1111, 1121, 1144]
    #     midCar = []
    #     highCar = []
    #     if 472 in car_ids:
    #         car_ids.remove(472)
    # elif file == 'A3':
    #     lowCar = [53, 64, 154, 156, 190, 325, 328, 388, 435, 447, 482, 565, 578, 603, 662,
        701, 702, 762, 765, 805,
    #               903,
    #               919, 982, 1140, 1150]
    #     midCar = [537, 538, 548, 631, 633, 643, 853, 860, 861, 1078, 1093, 1096]
    #     highCar = [287, 289, 293, 295]
    # elif file == 'A4':
    #     lowCar = [20, 119, 211, 301, 340, 369, 423, 482, 484, 521, 532, 556, 561, 604, 620,
        638, 692, 797, 860, 862,
    #               950, 1072, 1097, 1103, 1137, 1162, 1165]
    #     midCar = [579, 582, 591, 970, 971, 983, 1038, 1041, 1046]
    #     highCar = [232, 240, 249, 254, 448, 450, 453, 468]
    # elif file == 'A5':
    #     lowCar = [84, 87, 113, 123, 175, 266, 428, 429, 449, 452, 478, 603, 601, 708, 758,
        785, 796, 848, 908, 942,
    #               955,
    #               974, 972, 998, 1061, 1066, 1096, 1105, 1157, 1167]
    #     midCar = [1118, 1124, 1135]
    #     highCar = []

    # 获得汽车的停止时间与启动时间
    stops = []
    stopsv = []
    begins = []
```

```
for id in car_ids:
    if isStop(id, df) == 0:
        V = get_v(id, df)
        A = get_a(id, df)
        A1 = [x[1] for x in A]
        a0, t_a0 = get_a0(id, df)
        sequence = list(range(t_a0, getStopTime(id, df) + 1))

        # 有效时间
        df_car = df[df['vehicle_id'] == id]
        # if id in lowCar:
        #     filter_df = Q2_clearData(df_car, 75)
        # elif id in midCar:
        #     filter_df = Q2_clearData(df_car, 100)
        # elif id in highCar:
        #     filter_df = Q2_clearData(df_car, 125)
        # else:
        #     continue
        filter_df = Q2_clearData(df_car, 100)
        nearTimes = list(filter_df['time'])
        usefulTime = list(set(sequence) & set(nearTimes))

        a1 = []
        for i in A:
            if i[1] < a0 and i[0] in usefulTime:
                a1.append(i[1])

        if len(a1) == 0:
            stopa = '0.1'
        else:
            stopa = min(a1)

        if stopa == '0.1':
            stops.append(0)
        else:
            stops.append(A[A1.index(stopa)][0])

        # 计算begin
        last_zero_index = -1
        V1 = [x[1] for x in V]
        for index, value in enumerate(V1):
            if value == 0:
                last_zero_index = index
        begint = V[last_zero_index][0]
        begins.append(begint)

stopsAndbegins = Guibing(stops, begins)
```

```
    delete = []
    for i, j in enumerate(stopsAndbegins):
        if j[0] == 0:
            delete.append(i)
    stopsAndbegins = [element for index, element in enumerate(stopsAndbegins) if index not in
        delete]
    nStopsAndbegins = normalization(stopsAndbegins)

    C = dbscan(nStopsAndbegins, 0.01, 1)

    # x = []
    # y = []
    # for data in nStopsAndbegins:
    #     x.append(data[0])
    #     y.append(data[1])
    # plt.figure(figsize=(5, 5), dpi=80)
    # plt.scatter(x, y, c=C, marker='o')
    # plt.xlabel('stop')
    # plt.ylabel('begin')
    # plt.show()

    Classes = set(C)
    Classes = list(Classes)
    if -1 in Classes:
        Classes.remove(-1)
    begin_classes = []
    stop_classes = []
    for i in Classes:
        this_class_begin = []
        this_class_stop = []
        for k, j in enumerate(C):
            if j == i:
                this_class_begin.append(stopsAndbegins[k][1])
                this_class_stop.append(stopsAndbegins[k][0])

        # 当前车流的全灯周期
        this_begin = sum(this_class_begin) / len(this_class_begin)
        begin_classes.append(this_begin)

        # 当前车流的红灯周期
        stop_classes.append(this_class_begin[0] - this_class_stop[0])

    # 全灯周期
    begin_classes.sort()
    T = 9999
    for i in range(len(begin_classes)):
        if i == 0:
```

```
            continue
        if begin_classes[i] - begin_classes[i - 1] < T:
            T = begin_classes[i] - begin_classes[i - 1]


    T_red = max(stop_classes)
    trueBegins = [x[1] for x in stopsAndbegins]
    begins_Q3 = []
    for i in range(max(C)+1):
        indices = [index for index, value in enumerate(C) if value == i]
        elements = [trueBegins[i] for i in indices]
        begins_Q3.append(sum(elements) / len(elements))

    begins_Q3.sort()
    begins_Q3.reverse()
    T_tatal = [begins_Q3[i] - begins_Q3[i+1] for i in range(len(begins_Q3)-1)]
    T_tatal.reverse()

    # 压总周期
    for i, T in enumerate(T_tatal):
        if 150 < T < 225:
            T_tatal[i] = T/2
        elif 250 < T < 315:
            T_tatal[i] = T/3
        elif 375 < T < 415:
            T_tatal[i] = T/4
        elif 600 < T <650:
            T_tatal[i] = T /6
        elif T > 680:
            T_tatal[i] = T/7


    begins_Q3.reverse()
    stop_classes.pop(0)
    time_red_green = []

    for i in range(len(T_tatal)):
        time = begins_Q3[i]
        red = stop_classes[i]
        green = T_tatal[i] - red
        time_red_green.append((time, red))

    X = [x[0] for x in time_red_green]
    G = []
    for i in range(len(X)):
        G.append(T_tatal[i] - stop_classes[i])
    plt.figure(figsize=(5, 5), dpi=80)
```

```
    plt.plot(X, stop_classes, marker='o', c='red')
    plt.plot(X, T_tatal, marker='o', c='blue')
    plt.show()

    # X = [x[0] for x in time_red_green]
    # G = []
    # for i in range(len(X)):
    #     G.append(T_tatal[i] - stop_classes[i])
    # plt.figure(figsize=(5, 5), dpi=80)
    # plt.plot(X, stop_classes, marker='o', c='red')
    # plt.plot(X, G, marker='o', c='green')
    # plt.show()

    for i in begins:
        print(i)
```

1.4 问题四源代码

```
import pandas as pd
import matplotlib.pyplot as plt
from utils import *

df = pd.read_csv('C://Users//xxx//Desktop//math//fujian//4//D.csv')
car_ids = set(list(df['vehicle_id']))
car_ids = list(car_ids)

def inCross(x, y):
    if x*x + y*y <= 100**2:
        return True
    return False

car_ids_road = car_ids.copy()
i = 0
for id in car_ids:
    stop_x, stop_y = Q4getStopPoint(id, df)
    if inCross(stop_x, stop_y):
        car_ids_road.remove(id)


beginPoints = [(48.27, 496.27), (-496.22, 71.29), (-48.27, -496.27), (495.39, -21.7)]

X = []
Y = []
for id in car_ids:
    if Q4getBeginPoint(id, df)[0] == beginPoints[0][0]:
```

```
        X.append(Q4getStopPoint(id, df)[0])
        Y.append(Q4getStopPoint(id, df)[1])

X1 = df['x']
Y1 = df['y']
plt.figure(figsize=(6, 6), dpi=80)
plt.scatter(X1, Y1, marker='o', c='lightgray')
plt.scatter(X, Y, marker='o', c='purple')
plt.scatter(beginPoints[0][0], beginPoints[0][1], marker='o', c='red', s=[150])
plt.show()

# for id in car_ids_road:
#     if Q4getBeginPoint(id, df)[0] == beginPoints[3][0] and Q4getStopPoint(id, df)[1] > 200:
#         print(id)
```

Q4-所有车辆起点.py

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

df = pd.read_csv('C://Users//xxx//Desktop//math//fujian//4//D.csv')
car_ids = set(list(df['vehicle_id']))



X = []
Y = []
X1 = []
Y1 = []
for id in car_ids:
    df_filtered = df[df['vehicle_id'] == id]
    X.append(df_filtered.iloc[0]['x'])
    Y.append(df_filtered.iloc[0]['y'])
    X1.append(df_filtered.iloc[-1]['x'])
    Y1.append(df_filtered.iloc[-1]['y'])

X = df['x']
Y = df['y']
beginPoints = [(48.27, 496.27), (-496.22, 71.29), (-48.27, -496.27), (495.39, -21.7)]
bg0 = [x[0] for x in beginPoints]
bg1 = [x[1] for x in beginPoints]

plt.figure(figsize=(6, 6), dpi=80)
plt.scatter(X, Y, marker='o', c='lightblue')
plt.scatter(X1, Y1, marker='o', c='purple')
plt.scatter(bg0, bg1, marker='o', c='red')
plt.show()
```

Q4-综合因素.py

```
from utils import *



# 按顺序把stops和begins合并
def Guibing(stops, begins):
    stopsAndbegins = []
    for i in range(len(stops)):
        stopsAndbegins.append((stops[i], begins[i]))
    n = len(stopsAndbegins)
    for i in range(n):
        for j in range(0, n - i - 1):
            if stopsAndbegins[j][0] > stopsAndbegins[j + 1][0]:
                stopsAndbegins[j], stopsAndbegins[j + 1] = stopsAndbegins[j + 1],
                    stopsAndbegins[j]
    return stopsAndbegins

# 聚类
def Clustering(A1, distance=0.025):
    A1.sort()
    counts = []
    for i in A1:
        count = 0
        for j in A1:
            if math.fabs(i-j) < distance:
                count += 1

        counts.append(count)
    return counts

def get_a0(id, df):
    A = get_a(id, df)
    A1 = [x[1] for x in A if x[1] < -0.2 and x[0] < getStopTime(id, df)]
    counts = Clustering(A1)
    a0_index = counts.index(max(counts))
    a0 = A1[a0_index]

    t = A[A1.index(a0)][0]
    return a0, t

results1 = []
results2 = []
```

```python
for item in range(1):
    file = 'D'
    df = pd.read_csv('C://Users//xxx//Desktop//math//data//Q4-xy//' + file + '.csv')
    dfcar = pd.read_csv('C://Users\xxx\Desktop\math\data\Q4-车辆分类.csv')
    car_ids = set(list(dfcar['12']))
    car_ids = list(car_ids)
    num = int(len(car_ids) * 1)
    car_ids = random.sample(car_ids, num)
    # if file == 'A1':
    #     lowCar = [46, 158, 162, 197, 284, 418, 446, 464, 486, 506, 620, 640, 668, 682, 898, 905, 956, 993, 1027,
    #               1037, 1067, 1149]
    #     midCar = [99, 101, 102, 125, 123, 134, 225, 226, 236, 347, 359, 367, 766, 776, 775, 829, 840, 839, 869, 874,
    #               880]
    #     highCar = [539, 546, 550, 552]
    # elif file == 'A2':
    #     lowCar = [11, 100, 104, 129, 191, 220, 223, 463, 470, 522, 538, 564, 586, 623, 713, 749, 751, 781, 809, 865,
    #               961, 1015, 1057, 1073, 1111, 1121, 1144]
    #     midCar = []
    #     highCar = []
    #     if 472 in car_ids:
    #         car_ids.remove(472)
    # elif file == 'A3':
    #     lowCar = [53, 64, 154, 156, 190, 325, 328, 388, 435, 447, 482, 565, 578, 603, 662, 701, 702, 762, 765, 805,
    #               903,
    #               919, 982, 1140, 1150]
    #     midCar = [537, 538, 548, 631, 633, 643, 853, 860, 861, 1078, 1093, 1096]
    #     highCar = [287, 289, 293, 295]
    # elif file == 'A4':
    #     lowCar = [20, 119, 211, 301, 340, 369, 423, 482, 484, 521, 532, 556, 561, 604, 620, 638, 692, 797, 860, 862,
    #               950, 1072, 1097, 1103, 1137, 1162, 1165]
    #     midCar = [579, 582, 591, 970, 971, 983, 1038, 1041, 1046]
    #     highCar = [232, 240, 249, 254, 448, 450, 453, 468]
    # elif file == 'A5':
    #     lowCar = [84, 87, 113, 123, 175, 266, 428, 429, 449, 452, 478, 603, 601, 708, 758, 785, 796, 848, 908, 942,
    #               955,
    #               974, 972, 998, 1061, 1066, 1096, 1105, 1157, 1167]
    #     midCar = [1118, 1124, 1135]
    #     highCar = []

    # 获得汽车的停止时间与启动时间
    stops = []
```

```
    stopsv = []
    begins = []
    for id in car_ids:
        if isStop(id, df) == 0:
            V = get_v(id, df)
            A = get_a(id, df)
            A1 = [x[1] for x in A]
            a0, t_a0 = get_a0(id, df)
            sequence = list(range(t_a0, getStopTime(id, df) + 1))

            # 有效时间
            df_car = df[df['vehicle_id'] == id]
            # if id in lowCar:
            #     filter_df = Q2_clearData(df_car, 75)
            # elif id in midCar:
            #     filter_df = Q2_clearData(df_car, 100)
            # elif id in highCar:
            #     filter_df = Q2_clearData(df_car, 125)
            # else:
            #     continue
            filter_df = Q2_clearData(df_car, 100)
            nearTimes = list(filter_df['time'])
            usefulTime = list(set(sequence) & set(nearTimes))

            a1 = []
            for i in A:
                if i[1] < a0 and i[0] in usefulTime:
                    a1.append(i[1])

            if len(a1) == 0:
                stopa = '0.1'
            else:
                stopa = min(a1)

            if stopa == '0.1':
                stops.append(0)
            else:
                stops.append(A[A1.index(stopa)][0])

            # 计算begin
            last_zero_index = -1
            V1 = [x[1] for x in V]
            for index, value in enumerate(V1):
                if value == 0:
                    last_zero_index = index
            begint = V[last_zero_index][0]
            begins.append(begint)
```

```
131
132     stopsAndbegins = Guibing(stops, begins)
133     delete = []
134     for i, j in enumerate(stopsAndbegins):
135         if j[0] == 0:
136             delete.append(i)
137     stopsAndbegins = [element for index, element in enumerate(stopsAndbegins) if index not in
            delete]
138     nStopsAndbegins = normalization(stopsAndbegins)
139
140     C = dbscan(nStopsAndbegins, 0.025, 2)
141
142     x = []
143     y = []
144     for data in nStopsAndbegins:
145         x.append(data[0])
146         y.append(data[1])
147     plt.figure(figsize=(5, 5), dpi=80)
148     plt.scatter(x, y, c=C, marker='o')
149     plt.xlabel('stop')
150     plt.ylabel('begin')
151     plt.show()
152
153     Classes = set(C)
154     Classes = list(Classes)
155     if -1 in Classes:
156         Classes.remove(-1)
157     begin_classes = []
158     stop_classes = []
159     for i in Classes:
160         this_class_begin = []
161         this_class_stop = []
162         for k, j in enumerate(C):
163             if j == i:
164                 this_class_begin.append(stopsAndbegins[k][1])
165                 this_class_stop.append(stopsAndbegins[k][0])
166
167         # 当前车流的全灯周期
168         this_begin = sum(this_class_begin) / len(this_class_begin)
169         begin_classes.append(this_begin)
170
171         # 当前车流的红灯周期
172         stop_classes.append(this_class_begin[0] - this_class_stop[0])
173
174     # 全灯周期
175     begin_classes.sort()
176     T = 9999
```

```
    for i in range(len(begin_classes)):
        if i == 0:
            continue
        if begin_classes[i] - begin_classes[i - 1] < T:
            T = begin_classes[i] - begin_classes[i - 1]


    T_red = max(stop_classes)
    print(T, T_red)

    results1.append(T)
    results2.append(T_red)


    # trueBegins = [x[1] for x in stopsAndbegins]
    # begins_Q3 = []
    # for i in range(max(C)+1):
    #     indices = [index for index, value in enumerate(C) if value == i]
    #     elements = [trueBegins[i] for i in indices]
    #     begins_Q3.append(sum(elements) / len(elements))
    #
    # begins_Q3.sort()
    # begins_Q3.reverse()
    # T_tatal = [begins_Q3[i] - begins_Q3[i+1] for i in range(len(begins_Q3)-1)]
    # T_tatal.reverse()
    # for i, T in enumerate(T_tatal):
    #     if T > 200:
    #         for n in range(2, 20):
    #             if T / n <= 150:
    #                 T_tatal[i] = T / n
    #                 break
    #
    # print(T_tatal)
    # begins_Q3.reverse()
    # stop_classes.pop(0)
    # time_red_green = []
    # print(stop_classes)
    # for i in range(len(T_tatal)):
    #     time = begins_Q3[i]
    #     red = stop_classes[i]
    #     green = T_tatal[i] - red
    #     time_red_green.append((time, red/green))
    #
    # X = [x[0] for x in time_red_green]
    # Y = [x[1] for x in time_red_green]
    # plt.figure(figsize=(5, 5), dpi=80)
    # plt.plot(X, Y, marker='o')
```

```
    # plt.show()
```

1.5 辅助函数

RGcorrdinate.py 计算红绿灯坐标以及车辆停止坐标

```python
from utils import *

df = pd.read_csv('C://Users//xxx//Desktop//math//fujian//1//A1.csv')
car_ids = set(list(df['vehicle_id']))

rg = []
for id in car_ids:
    df_filtered = df[df['vehicle_id'] == id]
    X = list(df_filtered['x'])
    Y = list(df_filtered['y'])
    corrd = [(x, y) for (x, y) in zip(X, Y)]
    if isStop(id, df) == 0:
        count = Counter(corrd)
        most_common_element, most_common_count = count.most_common(1)[0]
        rg.append(most_common_element)

count = Counter(rg)
# 获取按降序排列的所有键值及次数
sorted_counts = count.most_common()

# 打印结果
results = []
for element, cnt in sorted_counts:
    print(f'元素: {element}, 出现次数: {cnt}')
    results.append(element[0])
print(results)
```

utils.py

```python
import pandas as pd
import matplotlib.pyplot as plt
import math
from collections import Counter
import numpy as np
import random

# 清理数据
def clearData(df):
    cross = [
        (-11.44, -1.26),
```

```
        (-2.94, 12.02),
        (33.73, -3.77),
        (4.46, -11.54),
        (-11.51, 0.22),
    ]
    df_filtered = df[np.sqrt((df['x'] - cross[4][0])**2 + (df['y'] - cross[4][1])**2) <= 100]
    return df_filtered

def Q2_clearData(df, distance):
    cross = [
        (11.54, 4.46),
        (-2.94, 12.02),
        (33.73, -3.77),
        (4.46, -11.54),
        (-11.51, 0.22),
        (-12.02, -2.94),
        (18.94, 1.04),
    ]
    df_filtered = df[np.sqrt((df['x'] - cross[5][0])**2 + (df['y'] - cross[5][1])**2) <=
        distance]
    return df_filtered

def get_a(id, df):
    V = get_v(id, df)
    length = len(V)

    A = []
    for i in range(len(V)):
        if i == 0:
            continue

        a = V[i][1] - V[i-1][1]
        A.append((V[i][0], a))

    return A

def get_v(id, df):
    df_filtered = df[df['vehicle_id'] == id]
    t1 = list(df_filtered['time'])
    t = [int(x) for x in t1]
    x = list(df_filtered['x'])
    y = list(df_filtered['y'])
    length = len(x)

    V = []

    for i in range(length):
```

```
        if i == 0:
            continue

        v = math.sqrt((x[i] - x[i-1])**2 + (y[i] - y[i-1])**2)
        V.append((t[i], v))

    return V

def get_v(id, df):
    df_filtered = df[df['vehicle_id'] == id]
    t1 = list(df_filtered['time'])
    t = [int(x) for x in t1]
    x = list(df_filtered['x'])
    y = list(df_filtered['y'])
    length = len(x)

    V = []

    for i in range(length):
        if i == 0:
            continue

        v = math.sqrt((x[i] - x[i-1])**2 + (y[i] - y[i-1])**2)
        V.append((t[i], v))

    return V

def isStop(id, df):
    V = get_v(id, df)
    list = [i[1] for i in V]
    count = Counter(list)
    if count[0.0] < 5:
        return 1
    else:
        return 0

def getStopTime(id, df):
    A = get_a(id, df)
    if isStop(id, df) == 0:
        list = [i[1] for i in A]
        count = Counter(list)
        most_common_element, most_common_count = count.most_common(1)[0]
        return A[list.index(most_common_element)][0]

def normalization(data):
    data_array = np.array(data)

```

```
    # 分别获取 x 和 y 的值
    x_values = data_array[:, 0]
    y_values = data_array[:, 1]

    # 对 x 和 y 分别进行归一化
    x_min, x_max = x_values.min(), x_values.max()
    y_min, y_max = y_values.min(), y_values.max()

    x_normalized = (x_values - x_min) / (x_max - x_min)
    y_normalized = (y_values - y_min) / (y_max - y_min)

    # 将归一化后的 x 和 y 合并回去
    normalized_data = list(zip(x_normalized, y_normalized))
    return normalized_data

def dbscan(Data, Eps, MinPts):
    def dist(t1, t2):
        dis = math.sqrt((np.power((t1[0] - t2[0]), 2) + np.power((t1[1] - t2[1]), 2)))
        # print("两点之间的距离为: "+str(dis))
        return dis

    num = len(Data) # 点的个数
    # print("点的个数: "+str(num))
    unvisited = [i for i in range(num)] # 没有访问到的点的列表
    # print(unvisited)
    visited = [] # 已经访问的点的列表
    C = [-1 for i in range(num)]
    # C为输出结果, 默认是一个长度为num的值全为-1的列表
    # 用k来标记不同的簇, k = -1表示噪声点
    k = -1
    # 如果还有没访问的点
    while len(unvisited) > 0:
        # 随机选择一个unvisited对象
        p = random.choice(unvisited)
        unvisited.remove(p)
        visited.append(p)
        # N为p的epsilon邻域中的对象的集合
        N = []
        for i in range(num):
            if (dist(Data[i], Data[p]) <= Eps): # and (i!=p):
                N.append(i)
        # 如果p的epsilon邻域中的对象数大于指定阈值, 说明p是一个核心对象
        if len(N) >= MinPts:
            k = k + 1
            # print(k)
            C[p] = k
            # 对于p的epsilon邻域中的每个对象pi
```

```
            for pi in N:
                if pi in unvisited:
                    unvisited.remove(pi)
                    visited.append(pi)
                    # 找到pi的邻域中的核心对象，将这些对象放入N中
                    # M是位于pi的邻域中的点的列表
                    M = []
                    for j in range(num):
                        if (dist(Data[j], Data[pi]) <= Eps): # and (j!=pi):
                            M.append(j)
                    if len(M) >= MinPts:
                        for t in M:
                            if t not in N:
                                N.append(t)
                # 若pi不属于任何簇，C[pi] == -1说明C中第pi个值没有改动
                if C[pi] == -1:
                    C[pi] = k
        # 如果p的epsilon邻域中的对象数小于指定阈值，说明p是一个噪声点
        else:
            C[p] = -1

    return C

def Q4getBeginPoint(id, df):
    df_filtered = df[df['vehicle_id'] == id]
    return df_filtered.iloc[0]['x'], df_filtered.iloc[0]['y']

def Q4getStopPoint(id, df):
    df_filtered = df[df['vehicle_id'] == id]
    return df_filtered.iloc[-1]['x'], df_filtered.iloc[-1]['y']
```

路口形状.py

```
from utils import *

df = pd.read_csv('C://Users//xxx//Desktop//math//fujian//3//C6.csv')

X = df['x']
Y = df['y']

plt.figure(figsize=(5, 5))
plt.scatter(X, Y, marker='o', color='lightcoral', label='vehicle')
plt.ylabel('a')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```