基于轨迹数据的交通信号灯周期估计研究

摘要

交通信号灯是非常重要的交通设施，在保障交通安全和顺畅通行方面起到至关重要的作用[1]。交通信号灯配时方案通常包括信号周期，绿信比以及各相位时长灯部分，其中准确计算信号周期是至关重要的一个环节[2]。本文根据问题所给的行车轨迹数据，针对“周期正常变化”、“存在样本量过少和定位误差问题”、“考虑周期变化”和“特殊复杂的交通路口”等四种问题情境，建立数学模型，分别求解出相应的交通信号灯周期。

针对问题 1，对交通信号灯路口系统进行抽象，建立几何模型并经 **PCA** 降维为一维**位置坐标**。**基于统计**分析方法对第一坐标主成分分析求解交通信号灯路口位置（如：**A1(11.4，4.8)**, **A2(-11.83，-3.64)**）；再对车辆随时间变化的行驶轨迹建立**时间序列**分析模型，求解交通信号灯最小周期（如：T_{A1}**=105,** T_{A2}**=88,** T_{A3}**=105**）；最后对路口 0-1 判别矩阵和车辆运动状态矩阵进行**叠置**，运用**趋近**思想求解绿灯结束、红灯开始的时刻，进而求得最小周期内红绿灯的持续时间，A1-A5 路口信号灯的周期计算结果如表2所示。

针对问题 2，针对获取的行车轨迹数据**样本量过少**（样本比例小或车流量少）和**定位误差**的实际情境，本文分析了样本量和数据质量对信号灯周期估计精度的影响。通过对不同车流量和样本比例的轨迹数据进行多种评价和对照，检查和验证了模型在不同条件下的灵敏性和**稳健性**，B1-B5 路口信号灯的周期计算结果如表3所示。

针对问题 3，实际交通信号灯周期可能会因特殊事件或管理需求而调整，本文以**累积频次直方图**和**周期分布图**为核心，联合对照分析红绿灯持续时长在各周期中的变化。C1-C6 路口信号灯的周期及其变化计算如表4所示。

针对问题 4，最后，针对特殊复杂的实际交通路口情况，本文先使用**判别公式**将各轨迹数据分类到 12 条路线上，该问题即转化为 12 个路口的信号灯周期求解问题。对每一条路线进行模型求解分析，得出**正常情况**下的红绿灯**合理周期**如表5所示。

关键字：主成分分析　时间序列　周期叠置趋近模型

一、 问题重述

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。

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、 模型假设

1、没有违章行为，即无闯红灯现象。

2、路口信号灯只有红灯和绿灯两种，不考虑黄灯。

3、在路口等红灯的车辆在红灯转变为绿灯的时刻立即启动，驶过路口。

三、 符号说明

表 1　符号说明

变量名称	符号含义	单位
t	车辆行驶时间点	s
x	车辆行驶路径的 x 坐标	m
y	车辆行驶路径的 y 坐标	m
C	第一坐标主成分矩阵	-
$c_{i,t}$	第 i 辆车在第 t 秒时的坐标经过 PCA 变换后的第一坐标主成分	-
F	车辆运动状态判别矩阵	-
$f_{i,t}$	第 i 辆车在第 t-1 秒至 t 秒内的运动状态	-
DMC	车辆的坐标主成分与运动状态的联合矩阵	-
c_0	交通信号灯路口的第一坐标主成分	-
T	交通信号灯的最小周期	s
FF	车辆运动状态的叠置矩阵	-
CC	车辆第一坐标主成分的叠置矩阵	-
q	绝对时间映射到叠置周期中的相对时刻点	s
q_0	交通信号灯最小周期内绿灯结束红灯开始的时刻点	s
T_{green}	交通信号灯最小周期内绿灯持续时长	s
T_{red}	交通信号灯最小周期红灯持续时长	s

四、 问题分析

4.1 问题一分析

针对问题 1，本文对交通信号灯周期系统建立**时间序列分析**模型和**周期叠置趋近**模型。将所有车辆的行车轨迹抽象为二维平面上的散点，再**基于统计**的方法求解交通信号灯的路口点位；接着对车辆离开路口的**时间序列**分布进行分析，求得交通信号灯的最小

周期；最后，对路口红绿灯的周期进行**叠置**，求解红灯开始的时刻，再基于路口红绿灯的周期对所有车辆的行动轨迹进行叠置，求解绿灯结束的时刻，最后最两个时刻进行**趋近**分析，求解到绿灯结束、红灯开始的时刻，即求解到红灯和绿灯的周期。流程图如下：

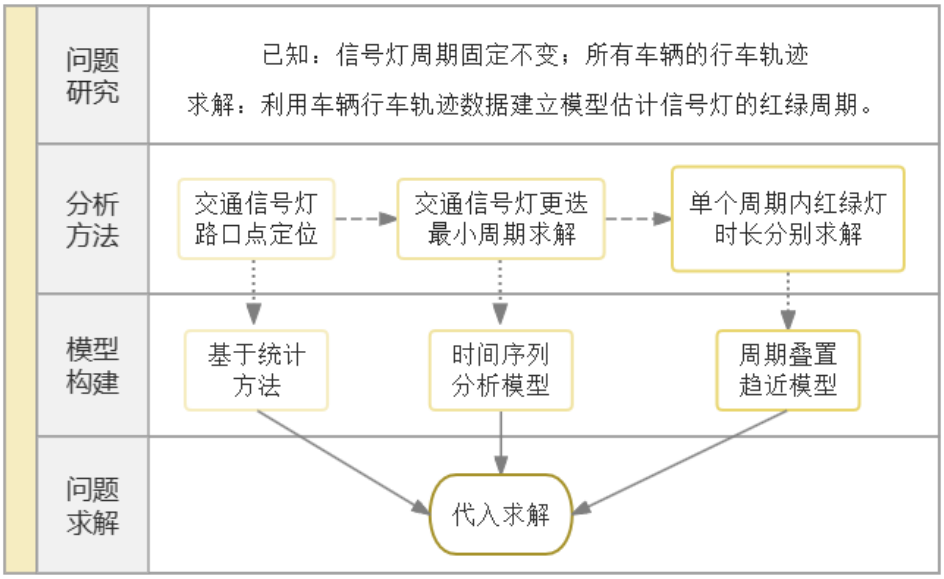

图 1　问题一流程图

4.2 问题二分析

针对问题二，本文根据数据集”B1-B5” 各自的实际特点，判断各数据是否存在样本车辆比例、车流量、定位误差等方面的特定问题，然后对各数据套用上述模型，对其特定问题进行比较分析，判断各因素对上述模型估计精度的影响。对于样本比例 η 和车流量 Q，本文认为：

$$n = \eta \cdot Q,$$

其中 n 为样本量。因此不论是样本比例过低还是车流量过少，影响均为样本量过少，从而使得估计不准确；但样本比例高和车流量大则影响不同，直观上样本比例高会使得估计准确率更高，而车流量过大则可能会导致堵塞情况的出现，可能会对精度造成暂无法估计的影响。

对模型估计精度的评定，通过绘制累计频次图、周期分布图，及计算其相关的统计量来确定。

4.3 问题三分析

问题三设定，交通信号灯的周期可能发生变化，本文同样先进行交通信号灯路口点定位，定位完成后尝试解求交通信号灯更迭的最小周期，检查该周期是否就已经发生了

变化。

然后需要判断每个周期内部的红绿灯时长是否会发生变化。尝试建立一个**红绿灯时长**对**周期序数**的函数，分析在该函数中，红绿灯时长是否随周期序数的变化而变化，若是，则需要对周期序列进行处理，寻找到这个转变周期，进而映射到准确时间点上。

4.4 问题四分析

针对问题四，本文在车辆轨迹图的基础上将路口分为四个方位，示意图如下。

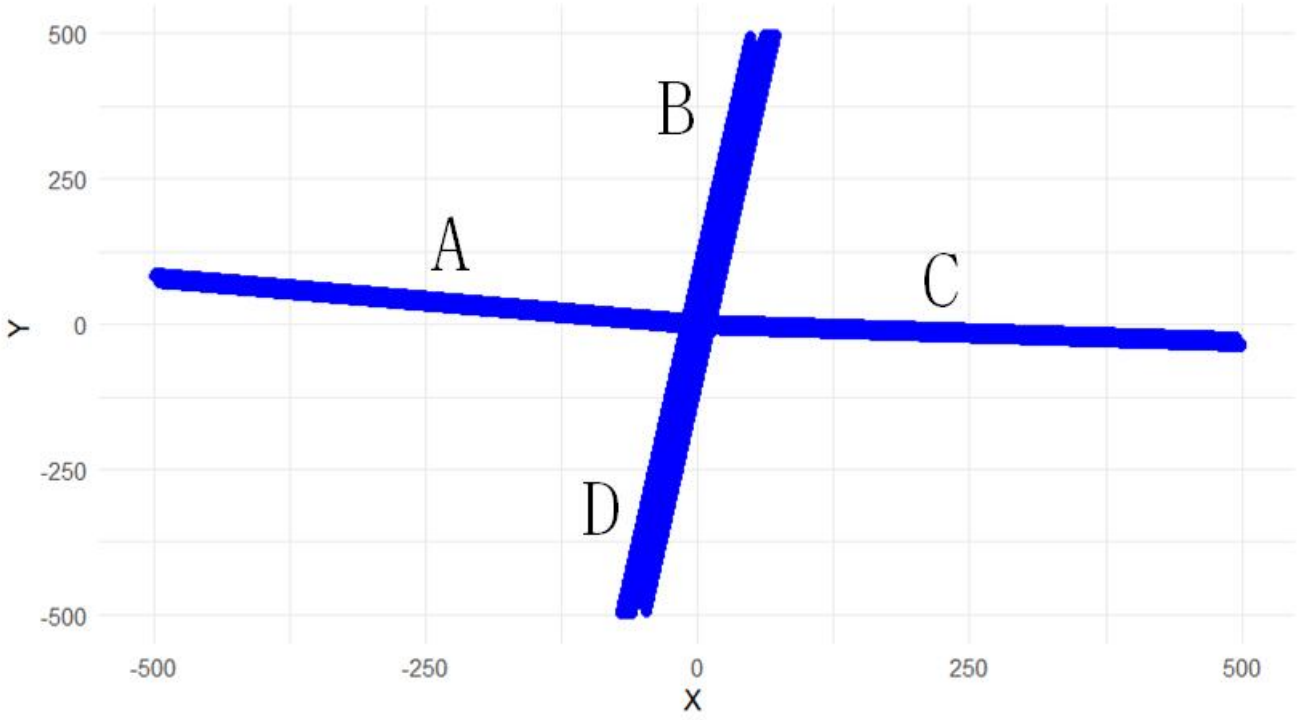

图 2　路口点位示意图

根据路口点位示意图，构建车辆转向判别函数 $M(x_1, y_1, x_2, y_2)$，

$$M(x_1,y_1,x_2,y_2)=\begin{cases}AB, & x_1<-250 \quad y_2>250,\\ AC, & x_1<-250 \quad x_2>250,\\ AD, & x_1<-250 \quad y_2<-250,\\ BA, & y_1>250 \quad x_2<-250,\\ BC, & y_1>250 \quad x_2>250,\\ BD, & y_1>250 \quad y_2<-250,\\ CA, & x_1>250 \quad x_2<-250,\\ CB, & x_1>250 \quad y_2>250,\\ CD, & x_1>250 \quad y_2<-250,\\ DA, & y_1<-250 \quad x_2<-250,\\ DB, & y_1<-250 \quad y_2>250,\\ DC, & y_1<-250 \quad x_2>250.\end{cases} \tag{1}$$

其中 (x_1,y_1) 为车辆起始点坐标，(x_2,y_2) 为车辆最终点坐标。

通过车辆转向判别函数 $M(x,y)$ 将路口数据集 D 划分为 12 个转向数据集，分别记为 $AB,AC,AD,BA,BC,BD,CA,CB,CD,DA,DB,DC$, 其中 AB 为图2中由 A 经过路口到 B 的数据，AC 为由 A 经过路口到 C 的数据……。然后对各转向数据集进行数据检查，套用上述模型，从而求得该路口所有方向的信号灯周期。

五、 模型构建

5.1 交通信号灯路口点定位

本文获得的初始数据是行车轨迹散点，由散点的时空变化可归纳出若干停车点（包括路口点），本节的核心问题则是——如何在这些停车点中，精确检索出交通信号灯路口点。由于等待红灯时，路口附近的各车辆可视为只进不出的队列，所以各车辆在路口停靠的总时间一定大于在其他地方停靠的总时间。因此本节的核心解决思路为——使用统计方法找到**车辆停靠总时间最长**的位置，该位置即为交通信号灯路口点。

为减少数据的冗余和便于对数据进行统计分析，在对所有车辆轨迹统计分析前，先使用 PCA 算法对车辆轨迹数据进行数据预处理，PCA（Principal Component Analysis）是一种常用的降维技术，用于发现数据中的主要模式和结构，其原理涉及对数据的协方差矩阵进行特征值分解。假设我们有一个包含 n 个样本和 p 个特征的数据集，可以将其表示为一个 $n\times p$ 的矩阵 X，其中每行代表一个样本，每列代表一个特征。PCA 的目标是

找到数据中的主要方向，即变量之间最大方差的方向，并将数据投影到这些方向上，以实现降维。

I 计算协方差矩阵：PCA 通过计算数据的协方差矩阵 $\sum$ 来确定主成分。协方差矩阵 $\sum$ 的元素 $\sigma_{i,j}$ 表示特征 i 和特征 j 之间的协方差。

$$\sum = \frac{1}{n} X^T X,$$

其中，X^T 是 X 的转置。

II 特征值分解：对协方差矩阵进行特征值分解。设 $\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_p A$ 是协方差矩阵 $\sum$ 的特征值，$\nu_1, \nu_2, \nu_3, \ldots, \nu_p$ 是对应的特征向量。特征向量 ν_i 表示数据在新的主成分方向 i 上的投影。

$$\sum \nu_i = \lambda_i \nu_i, \quad i = 1, 2, 3, \ldots, p$$

III 选择主成分及投影：根据特征值的大小顺序，选择最大的 k 个特征值及其对应的特征向量作为主成分, 这些特征向量构成了主成分矩阵 P。将原始数据 X 投影到所选的主成分上，得到降维后的数据。

$$Z = XP,$$

其中，Z 是降维后的数据矩阵, 即最终所求数据。

本题中轨迹数据为 2 个特征的数据集，将车辆行驶轨迹所抽象得到的二维点坐标经过 PCA 算法处理得到一维坐标系，如图3所示, 令变换函数为

$$\begin{cases} f_{PAC}(x, y) = c \\ f'_{PAC}(c) = (x, y) \end{cases}$$

其中，(x, y) 为轨迹点的二维坐标，c 为 (x, y) 经过 PCA 变换后的第一坐标主成分。

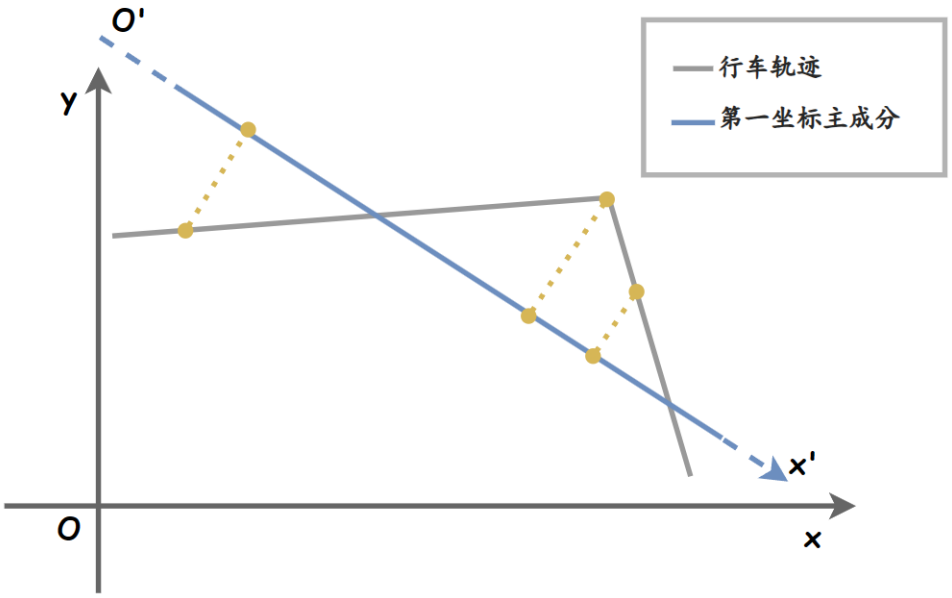

图 3　PCA 投影变换示意图

C 是一个记录车辆位置信息的矩阵，大小为 $m \times n$，$c_{i,t}$ 表示第 i 行第 t 列的元素，即为第 i 辆车在第 t 秒时的坐标主成分；F 是一个 0-1 判别矩阵，大小为 $m \times n$，$f_{i,t}$ 表示第 i 行第 t 列的元素，即为第 i 辆车在第 $t-1$ 秒至第 t 秒的运动状态。当 $f_{i,t}=1$，表示第 i 辆车在第 $t-1$ 秒至第 t 秒处于静止状态；当 $f_{i,t}=0$，表示第 i 辆车在第 $t-1$ 秒至第 t 秒处于运动状态，其表达式如下：

$$f_{i,t}=\begin{cases}1 & c_{i,t}=c_{i,t-1}\\ 0 & c_{i,t}\neq c_{i,t-1}\end{cases} 0<i\leq m, 0<t\leq n \quad and \quad i,t=1,2,3... \tag{2}$$

DMC 是一个车辆的坐标主成分与运动状态的联合矩阵，大小为 $m \times n$，$dmc_{i,t}$ 表示第 i 行第 t 列的元素。当第 i 车在第 $t-1$ 秒至第 t 秒内静止在 $c_{i,t}$ 点，$dmc_{i,t}=1$；否则，$dmc_{i,t}=0$。DMC 与 C 和 F 的关系为:

$$DMC = C \cdot F, \tag{3}$$

其中，“$\cdot$” 为两个矩阵的对应元素相乘。

DMC' 表示将 DMC 展开成一维的向量，在该一维向量中，所有值 dmc'_k 非 0 且唯一，其表达式为:

$$dmc'_k = dmc_{i,t}, k=1,2,3...and \quad dmc'_k \neq 0 \quad and \quad each \quad is \quad unique.$$

$Time(c_k)$ 表示所有车辆在 c_k 处停留的总时长。即为对每个唯一的非 0 的 $dmc_{i,t}$ 进行求和，表达下如下：

$$beginequation]Time(c_k)=\sum_{i=1}^{m}\sum_{t=1}^{n}[dmc_{i,t}=c_k].$$

当总时长 $Time(c_k)$ 最大时，其所对应的 c_k 即为所求的交通信号灯路口点位 c_0。其中，$argmax$ 函数表示为找出使给定函数取得最大值的变量的值。

$$c_0 = argmax(Time(c_k)) \quad k = 1, 2, 3.... \tag{4}$$

5.2 交通信号灯最小总周期求解

在交通信号灯路口点得以定位之后，即可解求交通信号灯更迭的最小总周期。对于在路口停靠的车辆，其开始停靠的时刻可能不是红灯开始的时刻，但**结束停靠**的时刻一定是**红灯结束的时刻**。因此对于在路口停靠的各车辆，求出所有结束停靠的时刻，即可组成一个时间序列——用于表示各周期红灯结束的时刻，由该时间序列的间隔分布可归纳出交通信号灯更迭的最小总周期。

假设第 $i-1$ 辆汽车在路口 c_0 处等候红灯结束，绿灯开始时刻为 t_1，第 i 辆汽车在路口 c_0 处等候红灯结束，绿灯开始时刻为 t_2，那么交通信号灯的最小周期为 $T = t_2 - t_1$。

F' 表示大小为 $i \times t$ 的路口 0-1 判别矩阵，大小为 $m \times n$，$f'i,t$ 表示第 i 行第 t 列的元素，即为位于交通信号灯路口处的第 i 辆车在第 $t-1$ 秒至第 t 秒的运动状态。当 $f'_{i,t} = 1$，表示位于交通信号灯路口处的第 i 辆车在第 $t-1$ 秒至第 t 秒的静止状态；当 $f'_{i,t} = 0$，表示位于交通信号灯路口处的第 i 辆车在第 $t-1$ 秒至第 t 秒的运动状态，其表达式如下：

$$f'_{i,t} = \begin{cases} 1 & c_{i,t} = c_{i,t-1} = c_0 \\ 0 & others \end{cases} \quad 0 < i \leq m, 0 < t \leq n \quad and \quad i, t = 1, 2, 3.... \tag{5}$$

考虑到有些车辆没有经停，在路口直接通行。对所有车辆的路口 0-1 判别矩阵进行轨迹分析，提取出所有在路口经停的车辆，首先定义一个集合 I 用来表示包含至少一个 1 的所有行的索引；然后，对于集合 I 中的每一行 i，我们定义函数 $ST(i)$ 来找出这一行中最后一个 1 出现的列数 t，公式如(6) 所示：

$$beginequation]I = \{i \mid \exists t, f'_{i,t} = 1\} \quad 0 < i \leq m \quad and \quad i = 1, 2, 3...$$

$$ST(i) = \max\{t \mid f'_{i,t} = 1, 0 < t \leq n \quad and \quad t = 1, 2, 3...\} \quad i \in I \tag{6}$$

接下来将 $ST(i)$ 进行重新排序，并且去除重复元素，保留独特值，具体公式如下；对应 T 中，每相邻两个元素相减即可求得交通信号灯的最小周期 T。

$$\begin{cases} ST(i)' = \{st_k \mid st_k \in ST(i)'\}, \\ MT = sort(ST(i)'), \\ T = MT_i - MT_{i-1} \quad i \geq 1. \end{cases} \tag{7}$$

5.3 红灯与绿灯的最小周期分别求解

先明确叠置周期的定义，即从红灯结束至下一个红灯结束的时间间隔，将原始的绝对时间抽象为相对时间段。假设将所有交通信号灯的周期叠置，对于红灯开始的近似时刻和绿灯结束的近似时刻，运用趋近模型从一个叠置的周期的两个端点分别求得，最后对求解到的两个时刻取均值，等到周期内的红灯开始、绿灯结束的准确时刻。

首先，在确定交通信号灯路口的情况下，路口 0-1 判别矩阵 F' 的数学模型为(5)，对其每一行基于最小周期 T 为单位进行分割，再对所有分割后的独立周期进行叠置，如图4所示，得到运动状态叠置矩阵 FF，其模型如下：

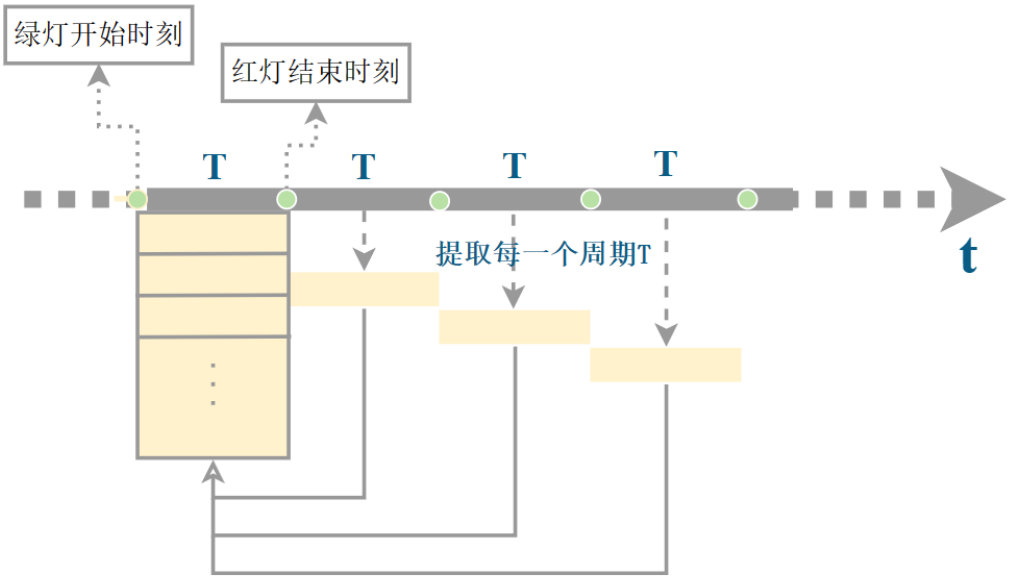

图 4　叠置分析示意图

$$\begin{cases} f'_{i,t} = \begin{cases} 1 & c_{i,t} = c_{i,t-1} = c_0 \\ 0 & others \end{cases} & 0 < i \le m, 0 < t \le n \quad and \quad i,t = 1,2,3..., \\ FF = (ff_{p,q})_{p=1,q=1}^{\frac{mn}{t},T}, \\ q = t \mod T \quad 0 \le q < T. \end{cases} \tag{8}$$

其中，q 表示为将所有时刻映射到叠置周期中的时刻点。

接下来对 FF 矩阵中的每一行元素进行统计分析，当 $ff_{p,q} = 1$ 时，求解出对应的最小 q 可证实为红灯开始的最临近时间，为其建立数学模型如下：

$$f(p) = \min\{q \mid ff_{p,q} = 1,\ 1 \le q \le T\}. \tag{9}$$

其中，$f(p)$ 表示为每一行 p 返回该行中 $ff_{p,q} = 1$ 的最小的 q 值。然后，对记录车辆位置信息的矩阵 C 也进行周期叠置，原理与对路口 0-1 判别矩阵 F' 叠置相同，不再赘述，

得到车辆位置信息叠置矩阵 CC，数学模型如下：

$$\begin{cases} CC = (cc_{p,q})_{p=1,q=1}^{\frac{mn}{t},T}, \\ q = t \mod T \quad 0 \le q < T. \end{cases} \tag{10}$$

对车辆位置信息叠置矩阵 CC 的每一行进行统计分析，定义一个函数 $g(p)$，表示为车辆运动至路口 c_0 处的相邻两秒内的运动状态，可间接判断出绿灯是否还，如果 $g(p) < 0$，即可证实此刻是绿灯状态。为确定绿灯结束的最邻近时刻，即需要寻找到每一行中，当 $g(p) < 0$ 时，最大的 q 值，建立数学模型如下：

$$\begin{cases} g(p) = (cc_{p,q-1} - c_0)(cc_{p,q} - c_0), \\ f(p) = \max\{q \mid g(p) < 0, 1 \le q \le T\}. \end{cases}$$

最后，利用趋近方法，如图5所示，将 $f(p)$ 中最小元素与 $g(p)$ 中最大的元素进行平均求和得到 q_0，即为绿灯结束、红灯开始的时刻，建立数学模型如下：

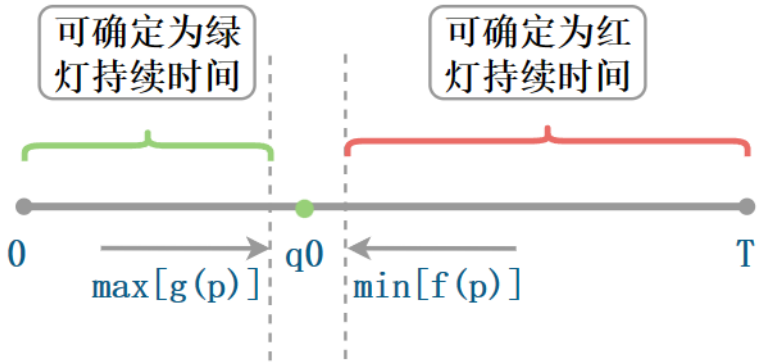

图 5　趋近分析示意图

$$q_0 = \frac{\min f(p) + \max g(p)}{2}. \tag{11}$$

因此，在交通信号灯的最小周期内，可求解到绿灯持续时间 T_{green}，即为 q_0；红灯持续时间 T_{red} 为总周期 T 减去绿灯持续时间 T_{green}，具体数学模型如下：

$$\begin{cases} T_{green} = q_0, \\ T_{red} = T - T_{green} \end{cases} \tag{12}$$

六、 问题求解

6.1 问题一求解

问题一求解步骤分为以下四个步骤：

Step1：将行车轨迹的二维坐标经过 PCA 降维处理。

在问题一求解前，先分析每一车辆位移与时间关系，其中 A1 得到 X 变化图如下。从图中可以目视师判别出 A1 路口点坐标 x 大致在 11m。

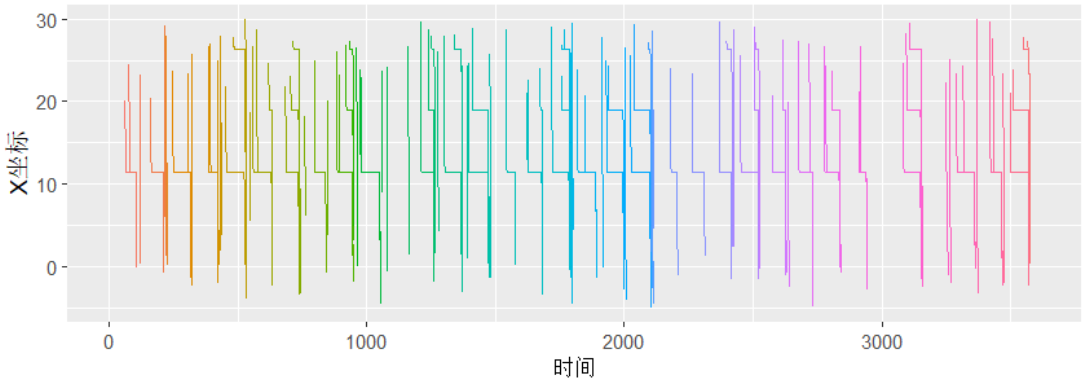

图 6　A1-X 时间变化图

接下来对行车轨迹数据使用 PCA 主成分分析进行数据预处理降维。若行车轨迹为一条直线，即道路不存在转弯，为直行路段，其降维得到的轨迹散点-第一主成分直线图为一条与原始路线相平行的直线，如图7所示；若行车轨迹为一条折线，即道路存在转弯点，为方向变更道路，其降维得到的轨迹散点-第一主成分直线图为一条与原始路线相交的直线，且与原始折线的两个夹角均为锐角，如图8所示。

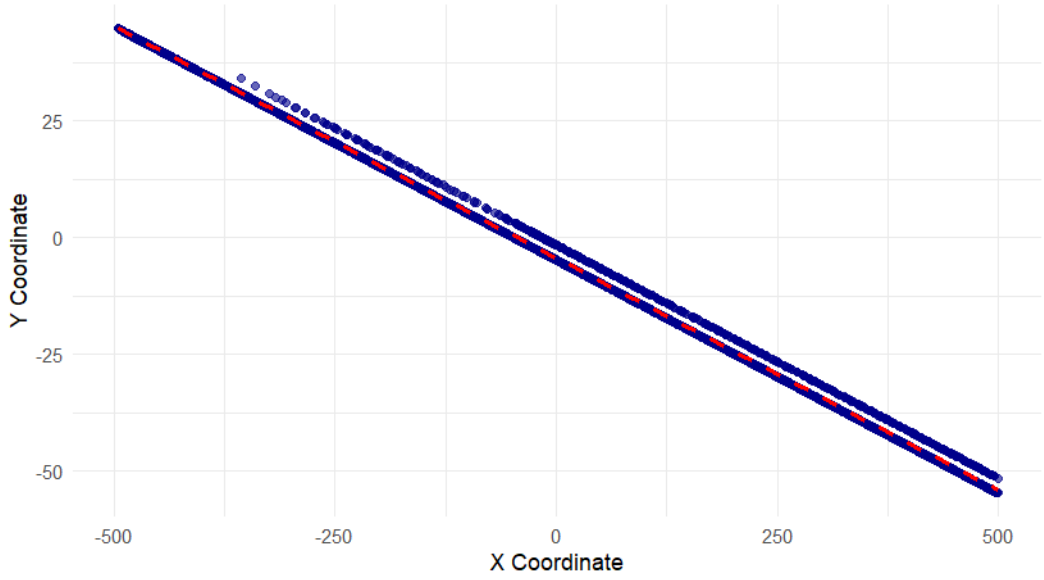

图 7　A2-PCA 直线图

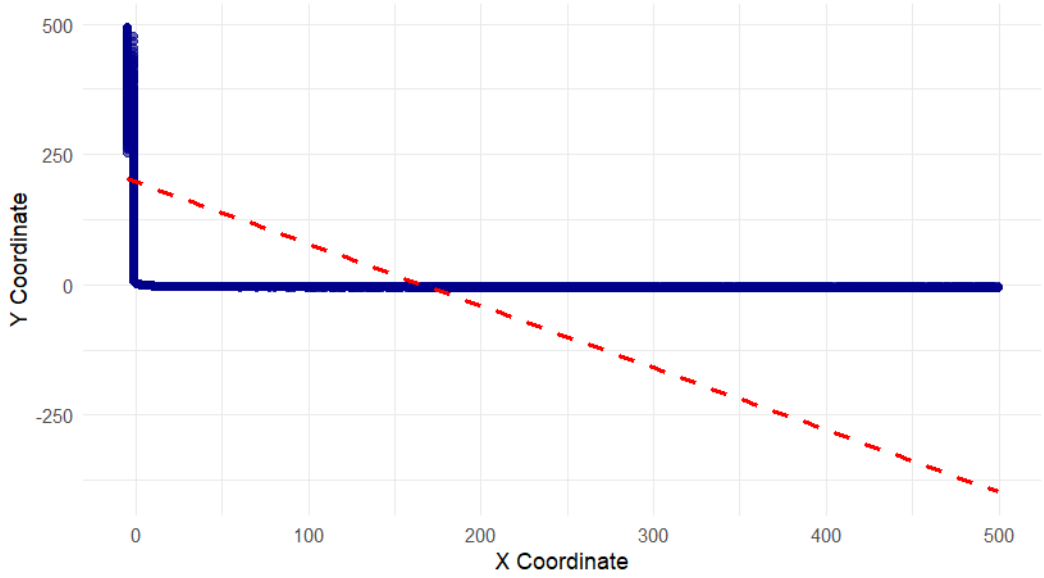

图 8　A3-PCA 直线图

Step2：交通信号灯路口点定位。

在运用程序求解过程中，其基本思想：若一车辆在第 i 秒内和在第 $i-1$ 秒内的坐标主成分相同，则判定为车辆处于等红灯静止状态；遍历所有车辆，将其第 i 秒内和在第 $i-1$ 秒内的坐标主成分进行对比，将所有车辆处于静止状态下最大位置信息统计分析，即计算车辆停止位置的时间频数，选择时间频数最大的位置作为路口位置，其中 A1,A4 路口停靠点时间直方图如下。

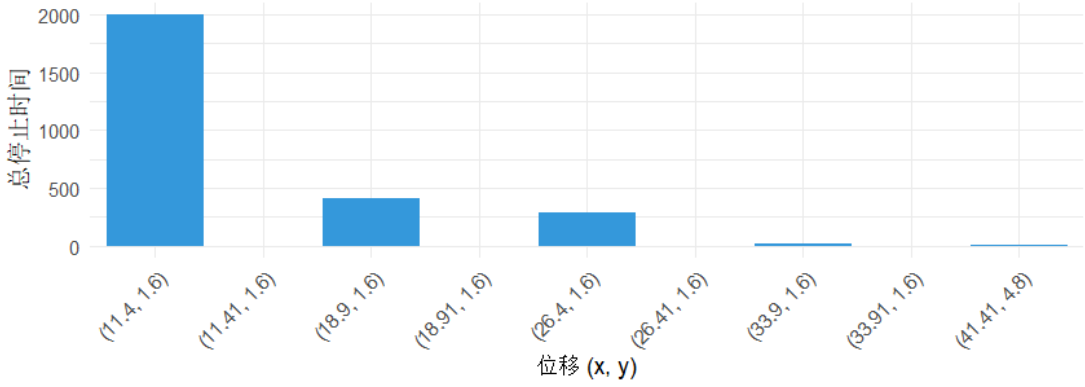

图 9　A1 停靠点时间分布图

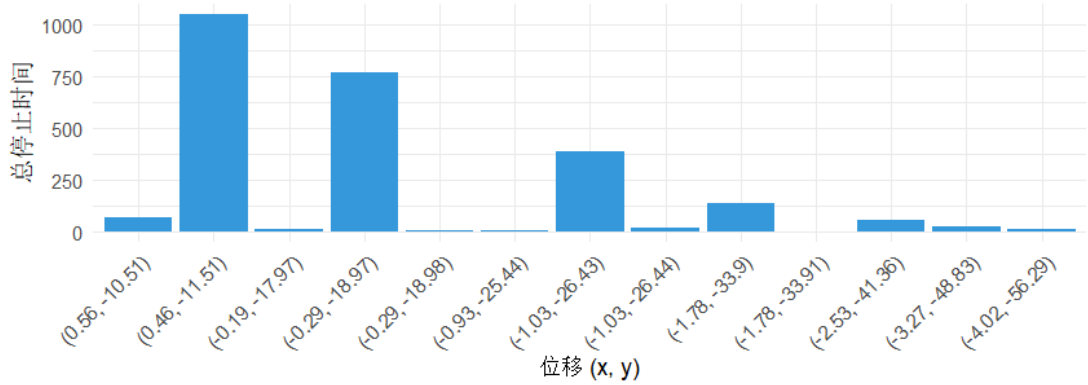

图 10　A4 停靠点时间分布图

根据统计结果，认为 A1-A5 路口位置分别为（11.4，4.8），(-11.83，-3.64），(-1.6，11.4），(0.46，-11.51），(3.64，-11.83）。(若路口有两个车道，本文只对展示一个车道。)

Step3：求解交通信号灯总周期。

在本步骤中，先对所有车辆按照序号和时间排序，对一行相邻两个时间点的行车轨迹点进行对比，若其保持不变且等于交通信号灯路口点位，则提取出每一行的最大的时间点；接下来对所有时间点进行排序，相邻两个时间之差即为对应的交通信号灯周期。但是，考虑到每个每两个时间点之差可能不相同，那么统计数据时间间隔，认为时间间隔统计频数最大的为总周期，其中得到的 A1 和 A4 的时间间隔统计图如下。

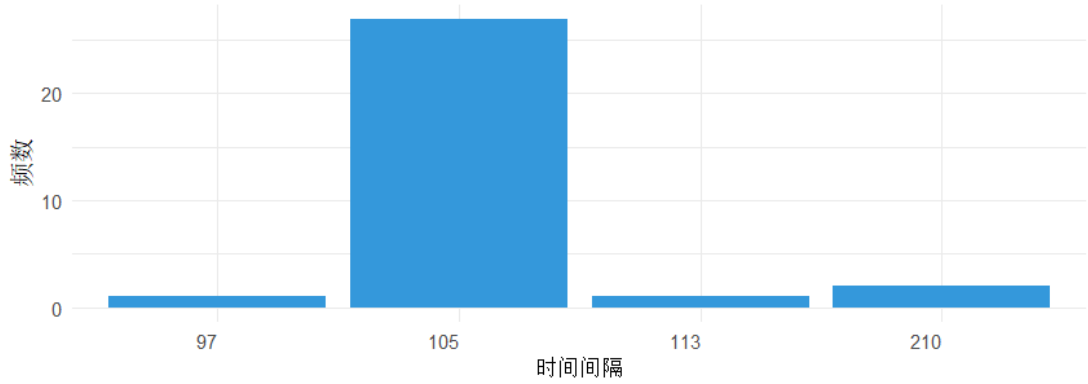

图 11　A1 时间间隔分布图

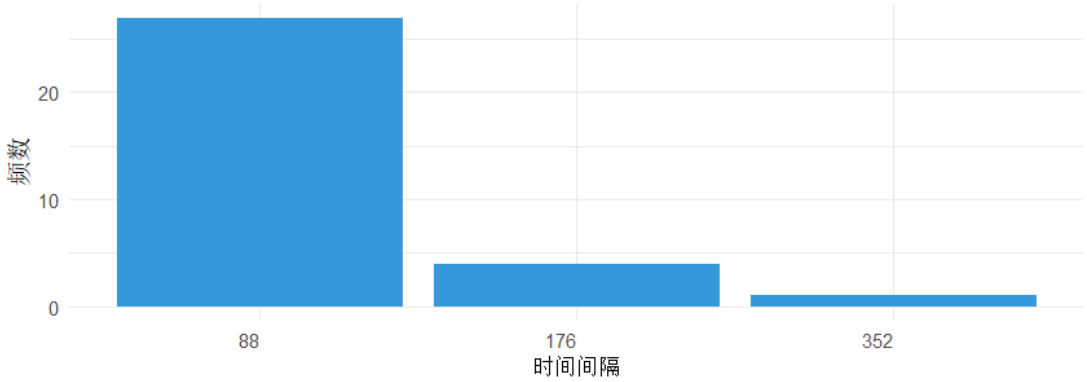

图 12　A4 时间间隔分布图

根据统计结果，认为 A1-A5 总周期分别为 105 秒，88 秒，105 秒，88 秒，88 秒。

Step4：求解交通信号灯周期内红绿灯的持续时长

对每一车辆的行车轨迹按照周期时间进行叠置，其中周期开始时间为红灯结束绿灯开始时刻。对所有车辆叠置后的行车轨迹点与路口点位进行对比分析：若相邻两个时间点的行车轨迹包含路口点位，即可证明此刻一定是绿灯，取出每一行中满足此绿灯条件的最大的时间点，即为绿灯结束的最邻近时间；若在相邻两个时间点的行车轨迹点与路口点位重合，即可证明此刻处遇红灯阶段，取出每一行中满足此红灯条件的最小时间点，即为红灯开始的最邻近时间点。

对于两个运用趋近方法求得的时间点与总周期时间段进行对比，可以明显计算出中间有一段剩余时间；将这段时间平均分给红灯和绿灯时间段中进行修正，即可得到准确的总周期内的绿灯结束红灯开始的时间点，进而求解出交通信号灯周期内红灯和绿灯的持续时长。结果如下表2。

表 2　路口 A1-A5 各自一个方向信号灯周期识别结果

路口	A1	A2	A3	A4	A5
红灯时长（秒）	74	56	79	66	61
绿灯时长（秒）	31	32	26	22	27

6.2 问题二求解

绘制出 B1-B5 的停靠点时间分布图，该图像中显示的众数表示路口所在的位置。除此之外，该图像还可展示一定的定位误差信息。如图13所示，车辆停靠点除了大部分较为突出的矩形之外（如众数-11.4，-1.6），还包括与这些矩形非常相近的坐标点（如-11.41，-1.6）。观察知，几乎所有路口的车辆轨迹记录均包含这种较小的偶然误差，但由于该误

差体量非常小且出现的频率非常低，它们几乎不会影响到模型中关于红灯和绿灯的判断。因此该种定位误差并不会影响模型的估计精度。

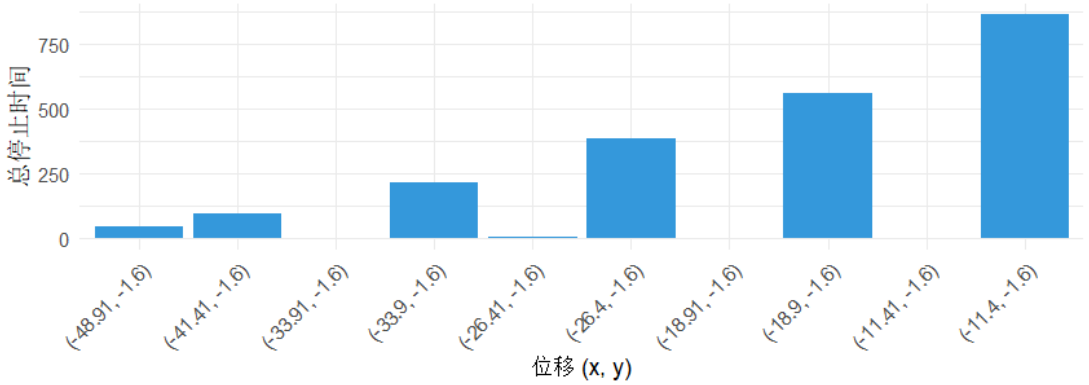

图 13　B1 停靠点时间分布图

绘制出所有车辆在路口停靠后，再次出发的时点（即绿灯开始的时点），如图14所示，这些时点数据较为完整，可以通过该时间序列的分布统计出红绿灯更迭一次的总周期。将时间间隔统计成直方图，如图15所示，可以直接确定占绝对优势的众数 105 即为红绿灯更迭一次的周期。

图 14　B1 已确定的绿灯开始时点

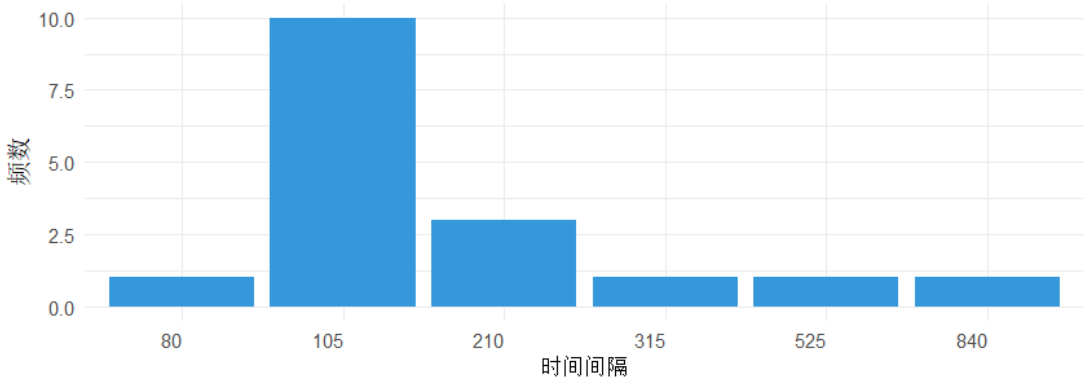

图 15　B1 时间间隔统计图

接下来对该思想进行佐证，已知 B1 路口存在定位误差，计算出 B1 路口每一时刻红绿灯判断的累积频次，绘制出直方图，如图16所示。

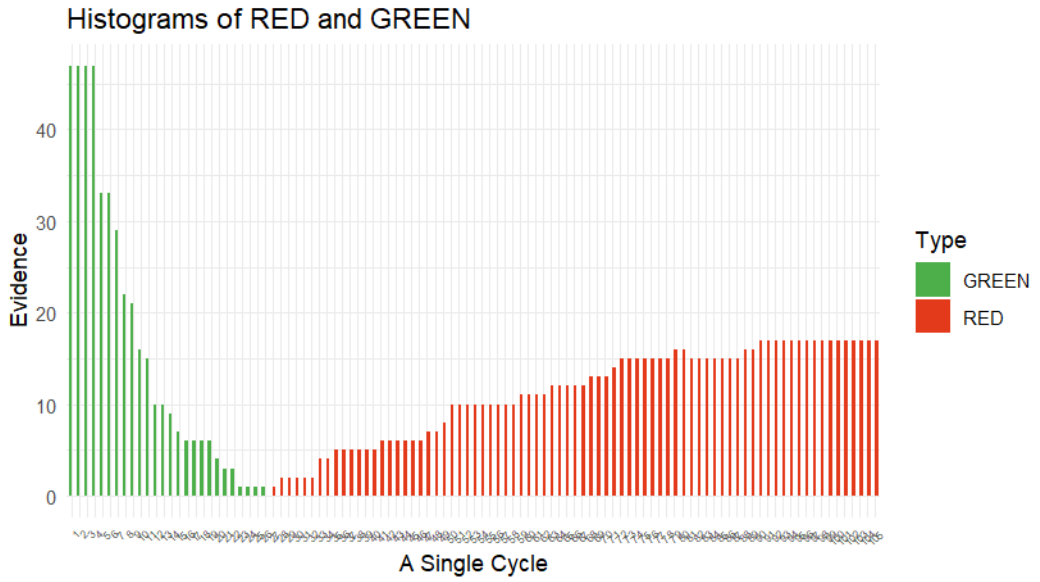

图 16　B1 累积频次直方图

其中图的横坐标表示任意时刻相对于本个红绿灯循环开始时刻的相对时刻，绿色矩形上的点表示该相对时刻及以后的时间通过路口的总车数，红色矩形上的点表示该相对时刻在等红灯的总车数，因此绿色或红色的值越大，表示该相对时刻为绿灯或红灯的把握就越大（类似于概率分布图的思想）。此时我们找到**谷底最小值**对应的相对时刻，该相对时刻可认为是由绿灯切换到红灯的时刻。

由该图像结果可以看出，对于路口 B1，即使有偶然误差的存在，其累积频次直方图表现得依旧很平滑，谷底最小值也很明显，也就是说模型估计精度较好。

对 B1 路口信号灯的每个周期，计算出可确定的红绿灯持续时长，可绘制成周期分布叠置柱状图图，如图17，可见即使存在偶然**误差**，绿灯和红灯趋近的结果也非常好，

不可确定为红绿灯的时长仅为 1 秒（最窄的灰色部分）。

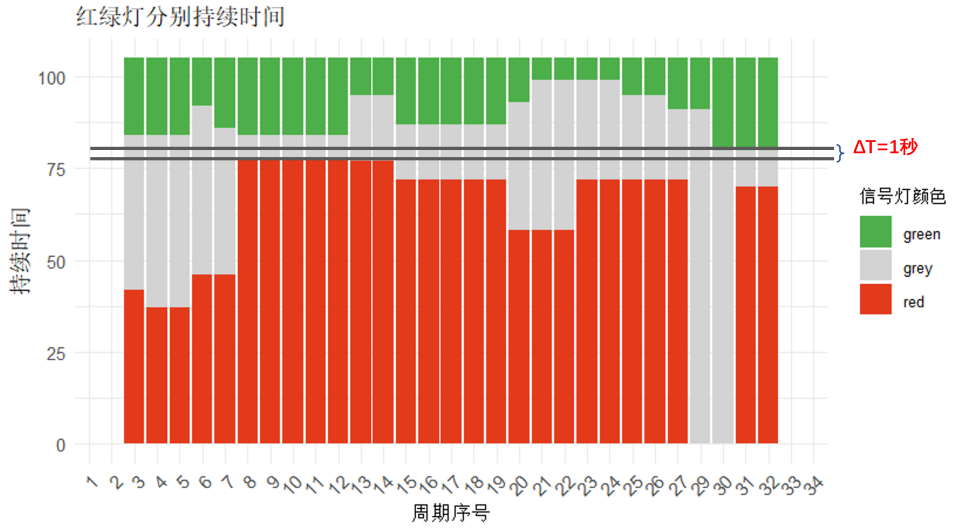

图 17　B1 周期分布图

对于样本比例因素和车流量因素进行考虑。首先对路口 B1-B5 的数据进行初步的分析和定位，整理得知路口 B1-B5 在一小时内的样本量分别为 73 辆、80 辆、21 辆、49 辆、47 辆，可认为相对而言，路口 B1 和 B2 的车流量（或样本比例）较大，路口 B3 的车流量（或样本比例）非常小，B4 和 B5 较为中等。下面考虑样本量（样本比例和车流量）的影响，由于 B3 路口的样本量非常小，此处即以 B3 路口为例，绘制出 B3 的停靠点时间分布图，如图18所示。可见不论样本量如何，求解路口点的坐标步骤并不受影响。

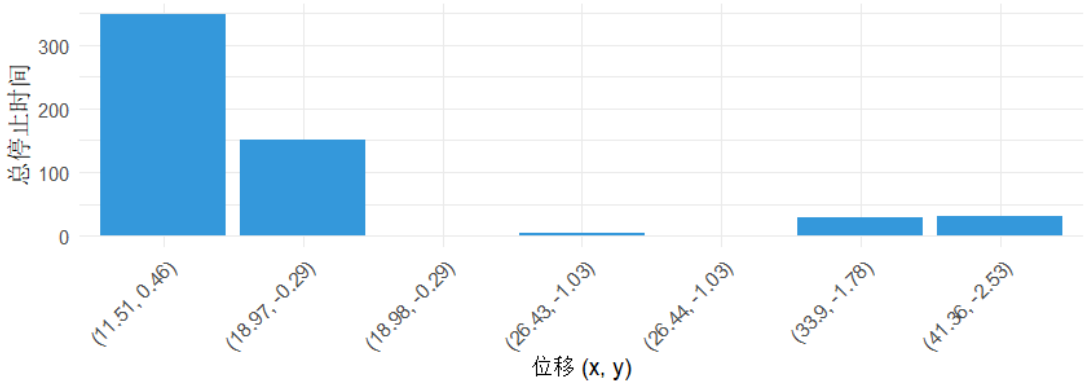

图 18　B3 停靠点时间分布图

绘制出所有车辆在路口停靠后，再次出发的时点（即绿灯开始的时点），如图19所示，可见该时间序列非常稀疏，可能难以归纳出一个合理的总周期。进一步绘制出直方图，如图20所示，其时间间隔的统计数量全为 1，无法在统计意义上寻找到总周期。此时只能凭借经验和观察，认为图中时间间隔的最大公因数 88 为总周期时长。可见样本

量少（样本比例小或车流量小）这一因素在本步骤中影响非常大，很可能导致计算出的红绿灯更迭周期不可靠。

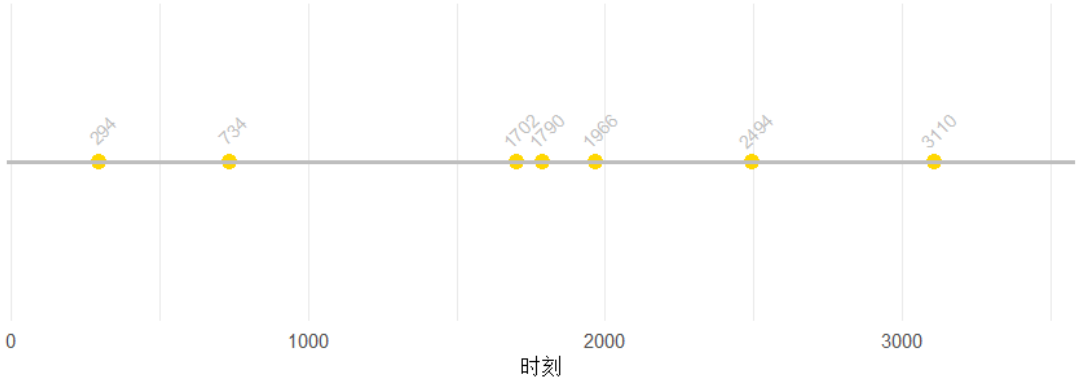

图 19　B3 已确定的绿灯开始时点

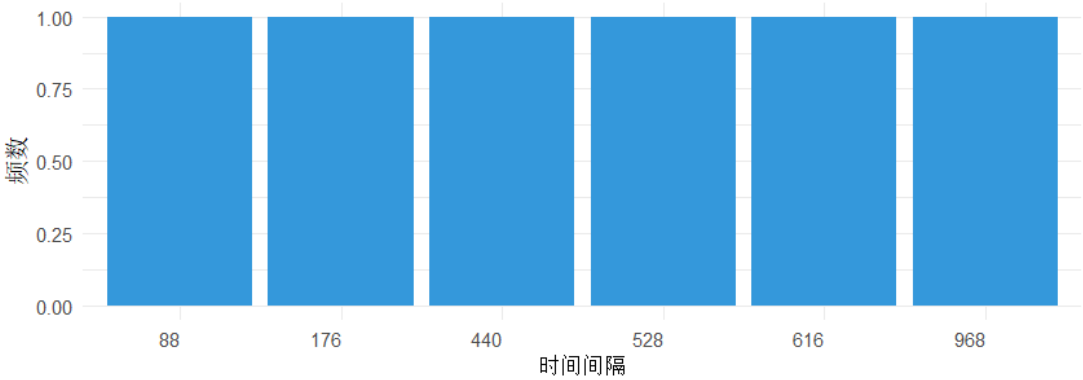

图 20　B3 时间间隔统计图

另外，计算出 B3 路口每一时刻红绿灯判断的累积频次，绘制出直方图，如图21所示。在该图像中也可以看出，样本量少（样本比例小或车流量小）这一因素对最终红绿灯切换时点的判断也影响较大——该图像的谷底最小值并不为一个值，而是一个区间，且该直方图的形状并不平滑，呈现出明显的锯齿状。因此，对于样本量非常少的 B3 路口，估计其红绿灯周期时得出的结果并不准确、且不可靠。

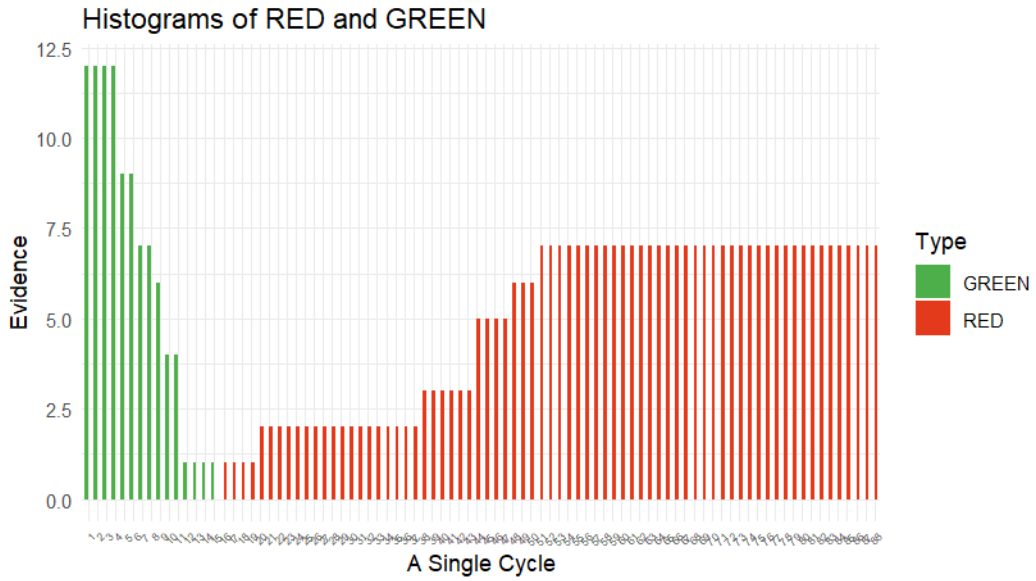

图 21　B3 累积频次直方图

之后，同理将数据代入到以上模型中，求解出信号灯周期如表3所示。

表 3　路口 B1-B5 各自一个方向信号灯周期识别结果

路口	B1	B2	B3	B4	B5
红灯时长（秒）	78	84	72	80	97
绿灯时长（秒）	27	32	16	25	19

其中路口 B1-B5 的红绿灯更迭最小总周期分别为 105 秒、116 秒、88 秒、105 秒、116 秒。

总而言之，对于本套数据（B1-B5）而言，**定位误差**这一因素对模型估计精度**几乎没有影响**，但并不排除实际情况下出现较大系统误差偏移的情况，此时若定位误差体量较大、数量不可忽略，那么就有可能降低模型估计精度。另外，**样本比例**和**车流量**这两个因素对模型估计精度**影响较大**，样本比例少或车流量小均可能导致样本量过小，继而导致总周期的计算、红绿灯持续时长的估计不再准确、可靠；反之，若样本比例大且车流量多（无论是否堵车），均会提升模型估计的准确性和可靠性。

6.3 问题三求解

问题三的求解主要围绕**累计频次直方图**和**周期分布图**进行分析。

首先套用现有模型，计算以确定的绿灯开始时点（即路口停靠一段时间车辆开始启动的时点），将该时点绘制在时间轴上，以 C6 数据为例，如22所示，将时间间隔统计成

直方图，如23所示。根据两张图可明显看出，在该段时间内，红绿灯更迭的总周期为一个恒定值 105 秒，并没有发生任何变化。对其他数据进行同等操作，统计得知 C1-C5 的红绿灯更迭的总周期分别恒定为 88 秒，88 秒，105 秒，105 秒，88 秒。

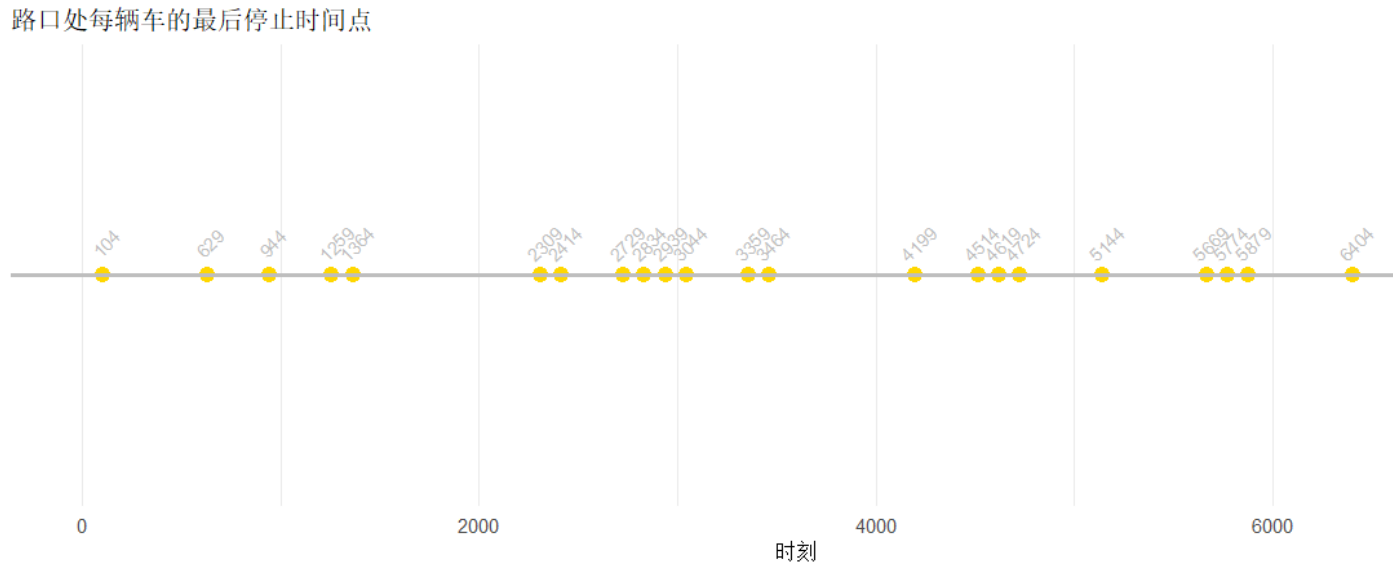

图 22　C6 已确定的绿灯开始点

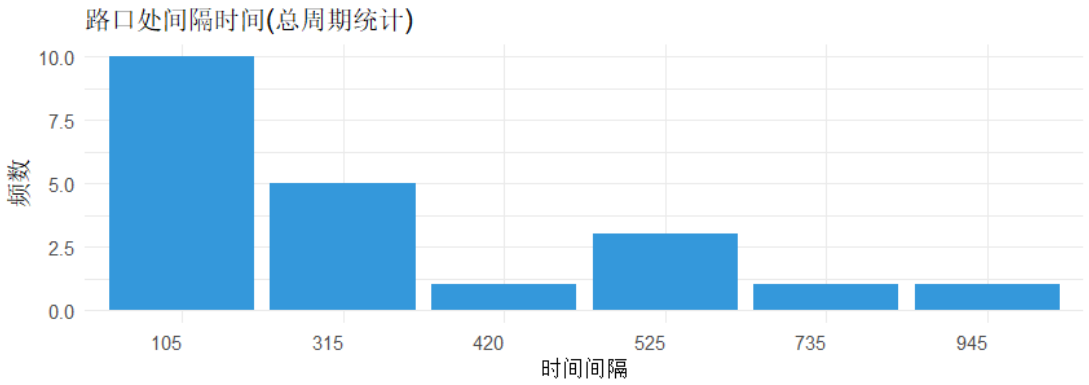

图 23　C6 时间间隔统计图

红绿灯更迭的总周期没有发生任何变化，此时即需要探索每个周期内部的红绿灯持续时长是否发生了变化。

首先绘制每组数据的累积频次直方图，若绿色区与红色区没有交叠，则可以认为该数据的红绿灯持续时长从来没有发生过变化（也可能是由于样本量不足）——即可以选择没有交叠区域的任意一个恒定时间点作为红绿灯更迭的时间，而不会与现有数据发生任何冲突；若绿色区与红色区有交叠，则可以考虑该数据的红绿灯持续时长可能发生过变化，即进入下一步的分析。

观察下图可知仅 C3、C5 有交叠区域，所以认为这两个路口的红绿灯持续时长可能发生了变化。

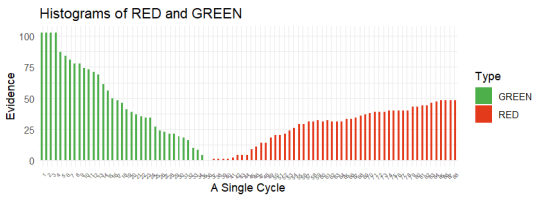

图 24　C1 累积频次直方图

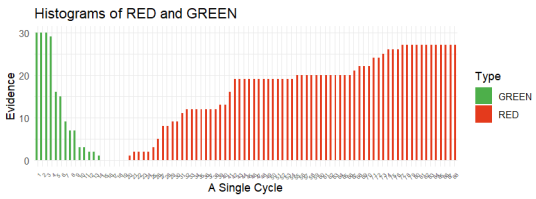

图 25　C2 累积频次直方图

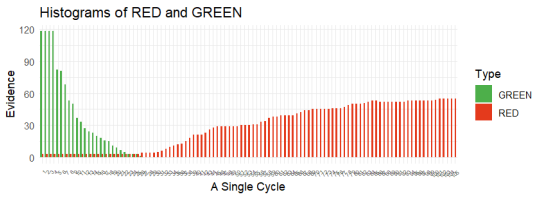

图 26　C3 累积频次直方图

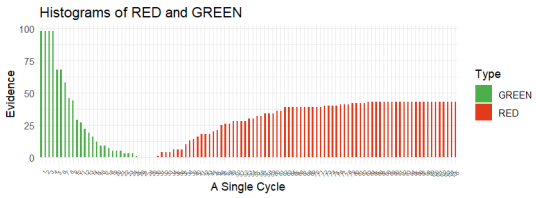

图 27　C4 累积频次直方图

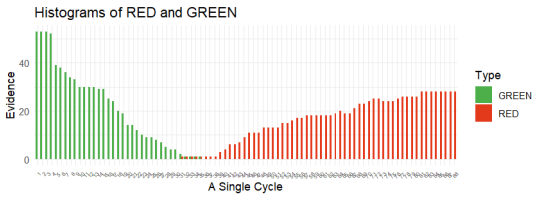

图 28　C5 累积频次直方图

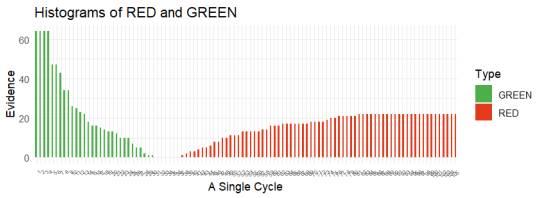

图 29　C6 累积频次直方图

绘制出 C3、C5 路口的周期分布图，如图30和图31所示。

对于图30而言，第 10 周期（921 秒）到第 13 周期（1340 秒）可确认的红灯时长和绿灯时长之和已经超过了红绿灯更迭的总周期，猜测在该时段内，该路口有特殊情况出现（如交警管制灯），使得非红灯时刻的某些车辆也必须停靠在路口位置。

对于图31而言，首先可以由图像中看出在第 58 周期（5017 秒）及以前，其红灯持续时长保持在一个较高的值，在 72 周期（6313 秒）及以后，其红灯持续时长保持在一个相对较低的值。因此需要在 58 周期到 72 周期之间，寻找到一个使红绿灯持续时长突变的时间点。使用均值确定突变的时刻，用该时刻将 C5 时段分为两部分，前后各自使用前文的周期叠置趋近模型计算红绿灯持续时长。所有分析结果如表4所示。

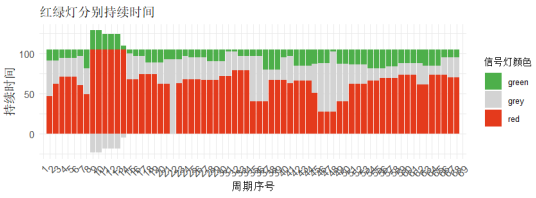

图 30　C3 周期分布图

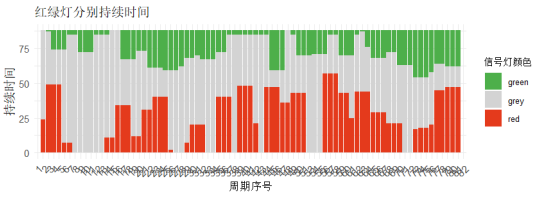

图 31　C5 周期分布图

表 4　第三问结果表格

路口	C1	C2	C3	C4	C5	C6
红灯时长（秒）	52	71	80	77	58	72
绿灯时长（秒）	36	17	25	28	30	33
周期切换时刻	-	-	921	-	5665	-
红灯时长（秒）	-	-	105	-	50	-
绿灯时长（秒）	-	-	0	-	38	-
周期切换时刻	-	-	1340	-	-	-
红灯时长（秒）	-	-	80	-	-	-
绿灯时长（秒）	-	-	25	-	-	-

6.4 问题四求解

首先将数据 D 按“vehicle_id”分组，使用公式1对每一组表示的车辆进行分类，由此可按行动轨迹将数据 D 分为 12 组，分别为 $AB, AC, AD, BA, BC, BD, CA, CB, CD, DA, DB$，其中 AB 为图2中由 A 经过路口到 B 的数据，AC 为由 A 经过路口到 C 的数据……然后对各组数据套用以上模型，可以求得该路口所有方向的信号灯周期。求出所有结果如表5所示，其中红灯时间和绿灯时间包括估计的部分（平摊未知时间）。

表 5　十字路口各路线红绿灯周期

路口	周期时间	红灯时间	绿灯时间	未知时间
AB	142	108	34	1
AC	142	104	38	0
AD	142	92	50	0
BA	142	79	63	0
BC	142	113	29	0
BD	142	96	46	0
CA	142	102	40	0
CB	142	94	48	1
CD	142	111	31	2
DA	142	115	27	3
DB	142	96	46	0
DC	142	90	52	0

以上计算仅考虑**红绿灯正常**、**合理运行**时的情况，即不考虑晚高峰、交通拥堵、交通管制、交警辅佐等特殊情况。

对计算结果进行分析，首先绘制所有路线的累积频次直方图和周期分布图，以 DB 路线为例，如图32和图33。这两张图显示，DB 路线红绿灯周期的估计非常准确——累积频次直方图较为平滑且有明显谷底，周期分布图的红灯最大值和绿灯最小值较为趋近，未知时段时长趋于 0。DB 路线的红绿灯周期估计表现较好，原因可能是由于数据量大，且在该路线上红绿灯周期正常、未发生改变。

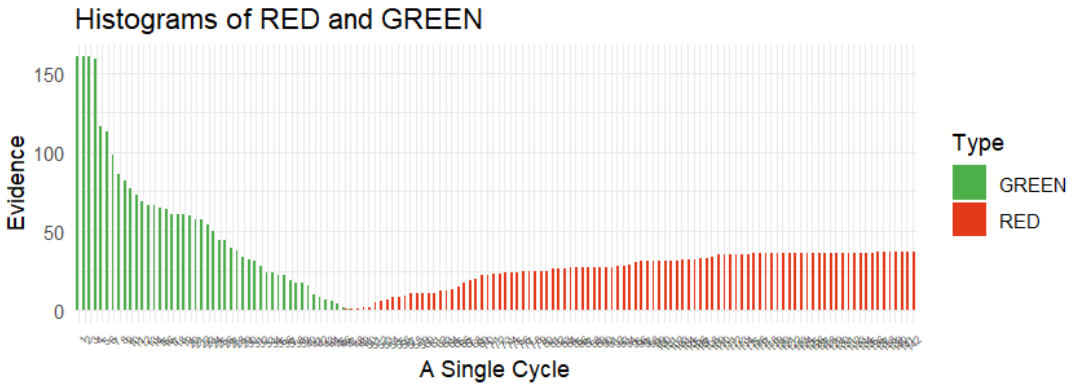

图 32　DB 累积频次直方图

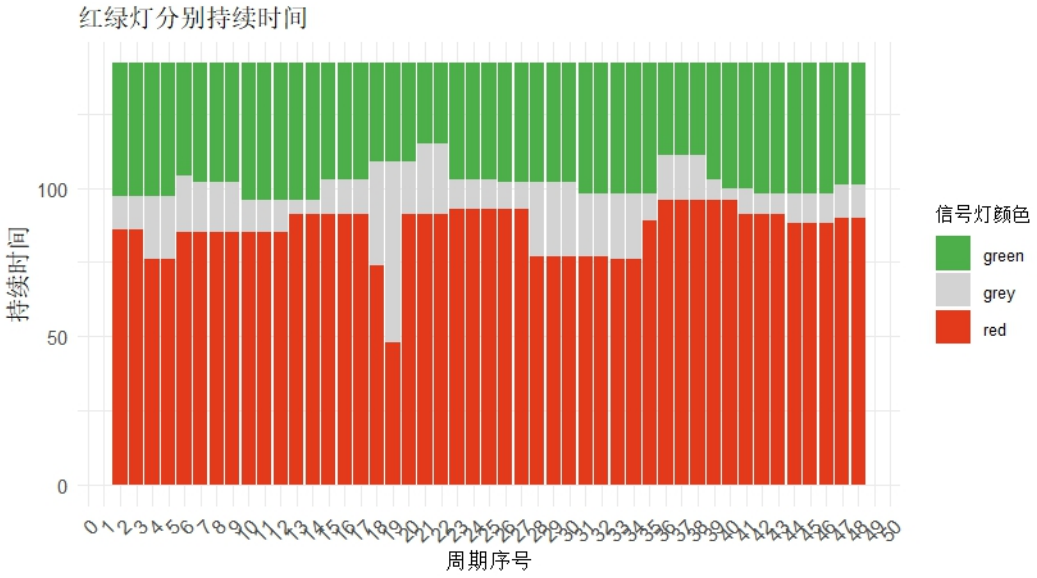

图 33　DB 周期分布图

但是除了 DB、BD 线路之外，其他线路的情况则更为复杂，以 AB 线路为例，其累积频次直方图如图34所示，其周期分布图如图35所示。

首先观察图35，可认为，在前 1/3 的时段内，该路口的红绿灯运行正常，按模型判断出的红绿灯时长总和小于总周期；但在后 2/3 的时段内，该路口的绿灯判断时长突然增大，且红绿灯时长总和已经超过了总周期，结合图34，可知此时必有大量车辆在本属于红灯的时刻穿过该路口。推测该路口可能出现了晚高峰、交通拥堵、交通管制、交警辅佐等特殊情况，且在这些情况下，红绿灯的持续时长不好量化。因此结果表格表5仅对红绿灯正常运转时的**合理且正常的周期**进行估计求解。

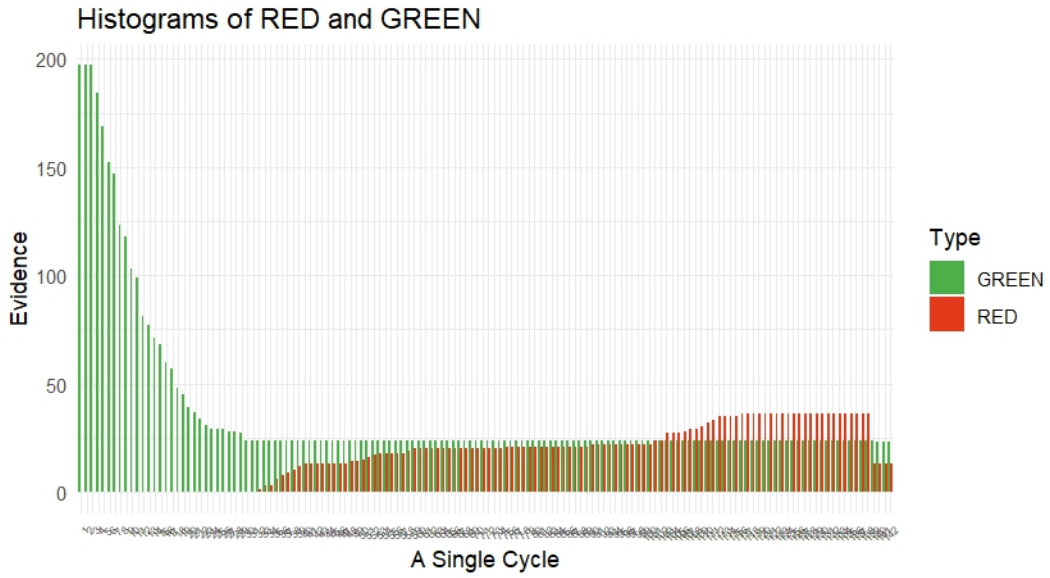

图 34　AB 累积频次直方图

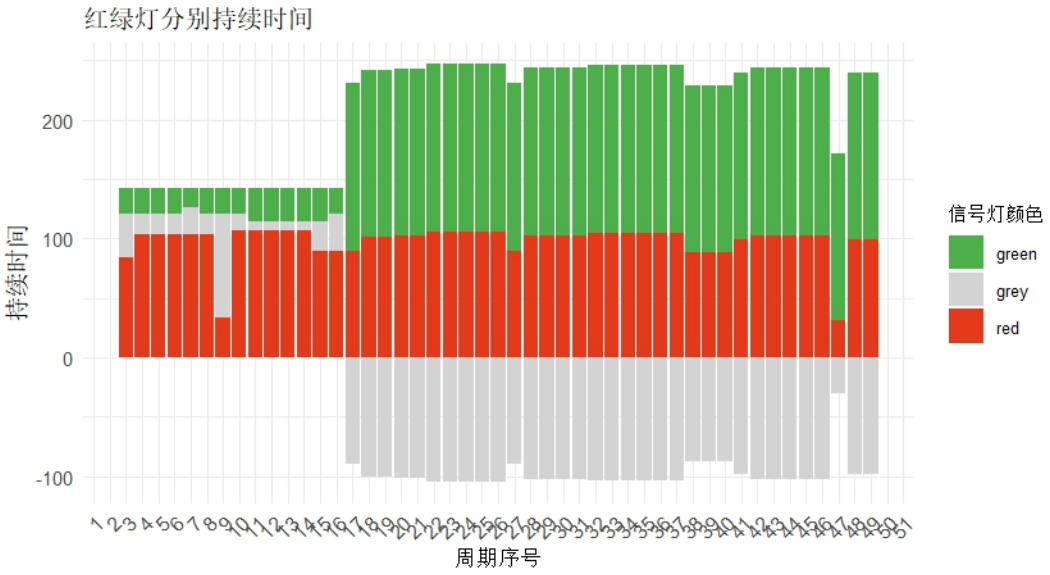

图 35　AB 周期分布图

七、 模型评价与改进

7.1 模型亮点

（1）通过几何抽象和主成分分析降维，对车辆行驶的轨迹数据进行简化和噪声处理，使得数据处理、分析和模型训练的速度更快；

（2）建立时序分析模型和周期叠置趋近模型，大大简化了问题的复杂程度，优化了模型性能，实现了更精确的预测结果。

（3）通过计算每一时刻交通信号灯判断的累计频次，将问题三寻找周期切换时刻转化为寻找谷底最小值对应的时刻 (如图16)，提升算法的稳定性和可解释性。

（4）计算交通信号灯每个周期内可确定红绿灯持续时段，可将其绘制成随周期变化的堆叠柱状图（如图17），判断上述结果的可靠性以及寻找周期变化时刻。

7.2 模型改进

（1）模型建立过程中存在一定的简化和理想化假设，未考处于交通信号灯下的车辆行驶的复杂性和测量系统的不确定性，在实际应用中需要结合具体情况进行参数修正和调整；

（2）约束条件的权重会对最终结果产生一定影响，需要进一步根据实际需求来权衡；

（3）由于数据集的有限，模型的预测能力和泛化性受到一定限制，需要提供更全面的训练基础。

7.3 模型推广

（1）自动驾驶和辅助驾驶领域：该模型可以准确判断交通信号灯的状态与类别，能够在一定程度上解决城市道路交通信号灯目标小、背景环境复杂多样造成的检测难度大的问题，对于智能汽车的行车安全十分重要；

（2）交通拥堵问题解决：该模型可以根据实时车流量的统计分析，预测出交通信号灯周期变化以及红绿灯的持续时长，对于交通拥堵的的时段和地点有一定的预测能力，能够提供弹性的交通信号灯配时优化方案，缓解交通拥堵问题。

参考文献

[1] 赵恩兴, 王超. 基于 YOLOv8 的交通信号灯识别 [J]. 人工智能与机器人研究,2023,12(3):246-254.

[2] 吴庆哲, 周俊. 基于模糊控制的多路口路段交通信号灯信号周期的研究 [J]. 工业控制计算机,2019,32(6):81-82+85.

[3] 王正武, 谢静怡, 王杰, 邢璐, 朱全军. 基于轨迹数据的非机动车左转信号灯安全效应评估 [J]. 长沙理工大学学报（自然科学版）,2024,21(1):164-173.

附录 A　问题一求解程序

```r
# Load necessary libraries
library(ggplot2)
library(dplyr)
library(stats) # 加载stats包
library(prettyR)

# 自定义函数来获取第m个众数
get_mode <- function(data_frame, m=1) {
  # 检查数据框是否只有一列
  if (ncol(data_frame) != 1) {
    stop("Data frame must have exactly one column.")
  }

  # 提取数据列
  data_vector <- data_frame[[1]]

  # 计算众数
  mode_values <- sort(table(data_vector), decreasing = TRUE)

  # 如果存在至少m个众数，则返回第m个众数
  if (length(mode_values) >= m) {
    return(as.numeric(names(mode_values)[m]))
  } else {
    warning("The data does not have m modes.")
    return(NA)
  }
}

# Function to process initial plots and the location of Cross
Analysis_Cross <- function(data, flag=FALSE) {
  if(flag==TRUE){
    # Plot vehicle X coordinates over time with thicker lines
    plot_vehicle_x <- ggplot(data, aes(x=time, y=PC1, color=factor(vehicle_id))) +
      geom_line(size=0.5) +
      labs(x = "时间", y = "X坐标") +
      theme(legend.position = "none") +
      ylim(-5, 30)
    print(plot_vehicle_x)
  }

  data <- data %>%
    arrange(vehicle_id, time) %>%
    group_by(vehicle_id) %>%
    mutate(stopped = (PC1 == lag(PC1)))
```

```
  # Calculate and summarize stop times
  stop_times <- data %>%
    arrange(vehicle_id, time) %>%
    group_by(vehicle_id) %>%
    filter(stopped == TRUE) %>%
    group_by(vehicle_id, PC1, x, y) %>%
    summarise(stop_time = n(), .groups = 'drop') %>%
    group_by(PC1, x, y) %>%
    summarise(total_time = sum(stop_time), .groups = 'drop')

  # 生成PC1对应的(x, y)标签
  stop_times$labels <- paste("(", stop_times$x, ", ", stop_times$y, ")", sep="")

  # Plot the distribution of displacement vs total stop time using PC1 as the x-axis and (x, y)
      as labels
  plot_displacement_vs_time <- ggplot(stop_times, aes(x=factor(PC1), y=total_time)) +
    geom_bar(stat="identity", fill="#3498db") + # Draw the bars
    labs(x = "位移 (x, y)", y = "总停止时间") +
    theme_minimal() +
    theme(axis.text.x = element_text(angle=45, hjust=1, vjust=1)) + # Set text alignment for
        better readability
    scale_x_discrete(breaks = stop_times$PC1, labels = stop_times$labels) # Set custom labels
        for x-axis

  print(plot_displacement_vs_time)

  # Identify the row with the maximum total_time
  max_stop_time <- stop_times %>%
    filter(total_time == max(total_time))

  return(list(data, max_stop_time))
}
# Function to process final plots and Cycle
Analysis_Cycle <- function(data, Threshold_x, Threshold_y) {
  # Get last stops at {Threshold_x} meters and calculate time intervals
  last_stops <- data %>%
    filter(x == Threshold_x & y == Threshold_y & stopped == TRUE) %>%
    group_by(vehicle_id) %>%
    filter(time == max(time)) %>%
    ungroup() %>%
    distinct(time) %>%
    arrange(time)

  # Plot last stop times on a time axis
  time_point_plot <- ggplot(last_stops, aes(x=time, y=0)) +
    geom_point(color="gold", size=3) +
```

```
  geom_hline(yintercept=0, color="grey", linetype="solid", size=1) +
  geom_text(aes(label = format(time, digits = 4)), angle = 45, vjust = -1, hjust = -0.1,
      color = "grey", size=3) +
  labs(x = "时刻") +
  xlim(range(data$time)) +
  theme_minimal() +
  theme(axis.text.y = element_blank(), axis.ticks.y = element_blank(), axis.title.y =
      element_blank(),
       panel.background = element_blank(), panel.grid.major.y = element_blank(),
           panel.grid.minor.y = element_blank())
print(time_point_plot)

# Calculate time intervals
time_intervals <- diff(last_stops$time)
interval_data <- data.frame(interval = time_intervals)

# Plot the distribution of time intervals
time_interval_plot <- ggplot(interval_data, aes(x=factor(interval), y=..count..)) +
  geom_bar(stat="count", fill="#3498db") +
  labs(x = "时间间隔", y = "频数") +
  theme_minimal() +
  theme(axis.text.x = element_text(hjust = 1))
print(time_interval_plot)

interval <- get_mode(interval_data)
# 判断条件并重新计算众数（如果需要）
if (interval > 150) {
  interval <- get_mode(interval_data, 2)
}
time_values <- last_stops$time
# 初始化结果向量，从第一个时间点减去间隔开始
result_data <- c(time_values[1] - interval, time_values[1])

# 遍历数据，逐一判断间隔并插入值
for (i in 2:length(time_values)) {
  # 计算相邻数据之间的间隔
  gap <- time_values[i] - time_values[i - 1]

  # 如果间隔大于给定的interval，就逐步插入值
  if (gap > 1.1 * interval) {
    seq_values <- seq(time_values[i - 1] + interval, time_values[i] - 1, by = interval)
    result_data <- c(result_data, seq_values, time_values[i])
  } else {
    result_data <- c(result_data, time_values[i])
  }
}
return(result_data)
```

```r
}
# 定义函数进行PCA计算并绘制散点图和PCA计算的直线
pca_fun <- function(data) {
  pca_result <- princomp(data[, c("x", "y")])
  pc1_coef <- pca_result$loadings[, 1]
  # 提取第一主成分（一维数据）
  pc1 <- pca_result$scores[, 1]
  # 创建新的数据框保存一维数据
  pca_data <- data.frame(PC1 = pc1)
  trajectory_data_with_pca <- cbind(data, PC1 = pc1)
  return(trajectory_data_with_pca)
}
setwd("C:\\Users\\86135\\Desktop\\华中杯\\B题：使用行车轨迹估计交通信号灯周期问题\\附件\\附件1")

# Assuming the data is read and named as A1-A5
A1 <- read.csv("A1.csv") %>% pca_fun()
A2 <- read.csv("A2.csv") %>% pca_fun()
A3 <- read.csv("A3.csv") %>% pca_fun()
A4 <- read.csv("A4.csv") %>% pca_fun()
A5 <- read.csv("A5.csv") %>% pca_fun()

A1_stopped <- Analysis_Cross(A1,TRUE)
result_A1 <- Analysis_Cycle(A1_stopped[[1]], A1_stopped[[2]]$x, A1_stopped[[2]]$y)

A2_stopped <- Analysis_Cross(A2,TRUE)
result_A2 <- Analysis_Cycle(A2_stopped[[1]], A2_stopped[[2]]$x, A2_stopped[[2]]$y)

A3_stopped <- Analysis_Cross(A3,TRUE)
result_A3 <- Analysis_Cycle(A3_stopped[[1]], A3_stopped[[2]]$x, A3_stopped[[2]]$y)

A4_stopped <- Analysis_Cross(A4,TRUE)
result_A4 <- Analysis_Cycle(A4_stopped[[1]], A4_stopped[[2]]$x, A4_stopped[[2]]$y)

A5_stopped <- Analysis_Cross(A5,TRUE)
result_A5 <- Analysis_Cycle(A5_stopped[[1]], A5_stopped[[2]]$x, A5_stopped[[2]]$y)
```

附录 B　问题二 & 三求解程序 (更改对应路径即可)

```r
setwd("C:\\Users\\86135\\Desktop\\华中杯\\B题：使用行车轨迹估计交通信号灯周期问题\\附件\\附件2")

library(ggplot2)
library(MASS) # 加载 MASS 包，其中包含了 PCA 函数
library(magrittr)
library(dplyr)
library(prettyR)
```

```r
library(tidyr)
library(zoo)

# 自定义函数来获取第m个众数
get_mode <- function(data_frame, m=1) {
  # 检查数据框是否只有一列
  if (ncol(data_frame) != 1) {
    stop("Data frame must have exactly one column.")
  }

  # 提取数据列
  data_vector <- data_frame[[1]]

  # 计算众数
  mode_values <- sort(table(data_vector), decreasing = TRUE)

  # 如果存在至少m个众数，则返回第m个众数
  if (length(mode_values) >= m) {
    return(as.numeric(names(mode_values)[m]))
  } else {
    warning("The data does not have m modes.")
    return(NA)
  }
}

# Function to process initial plots and the location of Cross
Analysis_TotalCycle <- function(data) {

  data <- data %>%
    arrange(vehicle_id, time) %>%
    group_by(vehicle_id) %>%
    mutate(stopped = (PC1 == lag(PC1)))

  # Calculate and summarize stop times
  stop_times <- data %>%
    arrange(vehicle_id, time) %>%
    group_by(vehicle_id) %>%
    filter(stopped == TRUE) %>%
    group_by(vehicle_id, PC1, x, y) %>%
    summarise(stop_time = n(), .groups = 'drop') %>%
    group_by(PC1, x, y) %>%
    summarise(total_time = sum(stop_time), .groups = 'drop')

  # Identify the row with the maximum total_time
  max_stop_time <- stop_times %>%
    filter(total_time == max(total_time))
```

```r
  Threshold_x <- max_stop_time$x
  Threshold_y <- max_stop_time$y

  # Get last stops at {Threshold_x} meters and calculate time intervals
  last_stops <- data %>%
    filter(x == Threshold_x & y == Threshold_y & stopped == TRUE) %>%
    group_by(vehicle_id) %>%
    filter(time == max(time)) %>%
    ungroup() %>%
    distinct(time) %>%
    arrange(time)

  # Calculate time intervals
  time_intervals <- diff(last_stops$time)
  interval_data <- data.frame(interval = time_intervals)

  interval <- get_mode(interval_data,1)
  # 判断条件并重新计算众数（如果需要）
  if (interval > 150) {
    interval <- get_mode(interval_data,2)
  }

  time_values <- last_stops$time
  # 初始化结果向量，从第一个时间点减去间隔开始
  result_data <- c(time_values[1] - interval, time_values[1])

  # 遍历数据，逐一判断间隔并插入值
  for (i in 2:length(time_values)) {
    # 计算相邻数据之间的间隔
    gap <- time_values[i] - time_values[i - 1]

    # 如果间隔大于给定的interval，就逐步插入值
    if (gap > 1.1*interval) {
      seq_values <- seq(time_values[i - 1] + interval, time_values[i] - 1, by = interval)
      result_data <- c(result_data, seq_values, time_values[i])
    } else {
      result_data <- c(result_data, time_values[i])
    }
  }

  return(result_data)
}

#定义计算路口位置函数
calculate_crossing_location <- function(data) {
  data <- data %>%
    arrange(vehicle_id, time) %>%
```

```
    group_by(vehicle_id) %>%
    mutate(stopped = (PC1 == lag(PC1)))

  # Calculate and summarize stop times
  stop_times <- data %>%
    arrange(vehicle_id, time) %>%
    group_by(vehicle_id) %>%
    filter(stopped == TRUE) %>%
    group_by(vehicle_id, PC1, x, y) %>%
    summarise(stop_time = n(), .groups = 'drop') %>%
    group_by(PC1, x, y) %>%
    summarise(total_time = sum(stop_time), .groups = 'drop')
  # 生成PC1对应的(x, y)标签
  stop_times$labels <- paste("(", stop_times$x, ", ", stop_times$y, ")", sep="")

  # Identify the row with the maximum total_time
  max_stop_time <- stop_times %>%
    filter(total_time == max(total_time))
  return(max_stop_time)
}

# 定义函数进行PCA计算并绘制散点图和PCA计算的直线
plot_pca_line <- function(data) {
  # 使用 PCA 计算数据的主成分
  pca_result <- princomp(data[, c("x", "y")])
  # 获取第一主成分的系数
  pc1_coef <- pca_result$loadings[, 1]
  # 提取第一主成分（一维数据）
  pc1 <- pca_result$scores[, 1]
  # 创建新的数据框保存一维数据
  pca_data <- data.frame(PC1 = pc1)
  trajectory_data_with_pca <- cbind(data, PC1 = pc1)
  # 计算直线方程的斜率和截距
  slope <- pc1_coef[2] / pc1_coef[1]
  intercept <- mean(data$y) - slope * mean(data$x)
  # 计算直线的两个端点，可以取 x 的最小和最大值
  min_x <- min(data$x)
  max_x <- max(data$x)
  # 计算直线的两个端点的坐标
  end_point1_y <- slope * min_x + intercept
  end_point2_y <- slope * max_x + intercept

  ## 绘制散点图和PCA计算的直线
  #plot_result <- ggplot(data, aes(x, y)) +
  #  geom_point(color = "darkblue", alpha = 0.6, size = 1.5) +
  #  geom_segment(aes(x = min_x, y = end_point1_y, xend = max_x, yend = end_point2_y),
  #           color = "red", linetype = "dashed", size = 1) +
```

```
  #  labs(x = "X Coordinate", y = "Y Coordinate") +
  #  theme_minimal() +
  #  theme(legend.position = "none") # 移除图例
  # 输出绘图
  # print(plot_result)

  return(trajectory_data_with_pca)
}

# 红灯判断 &      ##########红灯持续时间统计
find_longest_stopped_vehicle <- function(trajectory_data, PC1) {
  # 初始化停止时间最长的车辆ID、停止时间和开始停止时间
  longest_vehicle_id <- NA
  longest_stop_time <- -Inf
  start_stop_time <- Inf

  # 获取轨迹数据中的唯一车辆ID
  vehicle_ids <- unique(trajectory_data$vehicle_id)

  START_RED <- numeric(max(trajectory_data$time0)-min(trajectory_data$time0)+1)
  # 遍历每辆车
  for (vehicle_id in vehicle_ids) {
    # 选择该车辆在位置 PC1 处的数据

    # 用于有误差时的情况
    # vehicle_data <- trajectory_data[trajectory_data$vehicle_id == vehicle_id &
        abs(trajectory_data$PC1 - PC1)<0.1, ]
    vehicle_data <- trajectory_data[trajectory_data$vehicle_id == vehicle_id &
        trajectory_data$PC1 == PC1, ]

    # 如果该车辆在位置 PC1处没有停止数据，则继续下一辆车
    if (length(unique(vehicle_data$time0)) <= 2) {
      next
    }
    # 获取该车辆停止的最开始时间和停止时间
    start_time <- min(vehicle_data$time0)
    stop_time <- max(vehicle_data$time0)
    # 计算该车辆在位置 PC1处的停止时间
    stop_duration <- stop_time - start_time + 1
    # 更新最长停止时间及其对应的车辆ID和开始停止时间
    if (stop_duration > longest_stop_time) {
      longest_stop_time <- stop_duration
      longest_vehicle_id <- vehicle_id
      start_stop_time <- start_time
    }
    START_RED[(start_time):(stop_time+1)]=START_RED[(start_time):(stop_time+1)]+1
  }
```

```
  # 返回结果
  return(list(vehicle_id = longest_vehicle_id, stop_time = longest_stop_time, start_time =
      start_stop_time, START_RED = START_RED))
}

# 判断某一秒是否为绿灯
calculate_traffic_light <- function(trajectory_data, PC1, time) {
  # 默认设置信号灯为红灯
  light_status <- 0

  # 获取当前时间点和前一秒时间点的数据
  current_data <- trajectory_data[trajectory_data$time == time, ]
  previous_data <- trajectory_data[trajectory_data$time == time - 1, ]

  # 检查所有当前时间点的车辆
  for (vehicle_id in unique(current_data$vehicle_id)) {
    # 获取当前车辆和前一时刻的数据
    vehicle_current <- current_data[current_data$vehicle_id == vehicle_id, ]
    vehicle_previous <- previous_data[previous_data$vehicle_id == vehicle_id, ]

    # 检查车辆是否通过了交叉口
    if (nrow(vehicle_previous) > 0 && nrow(vehicle_current) > 0) {
      # 检查车辆是否从交叉口一侧移动到另一侧
      if ((vehicle_previous$PC1 < PC1-0.1 && vehicle_current$PC1 > PC1+0.1) ||
          (vehicle_previous$PC1 > PC1+0.1 && vehicle_current$PC1 < PC1-0.1)) {
        light_status <- 1 # 如果车辆穿越了交叉口，设定为绿灯
        break
      }
    }
  }
  return(light_status)
}

calculate_longest_Green_light <- function(trajectory_data, PC1, green_start_list,
    duration_to_check) {
  # 初始化最长绿灯持续时间
  longest_green_duration <- 0
  # 遍历所有绿灯开始时间
  for (green_start in green_start_list) {
    # 循环遍历需要检查的时间范围
    for (time in (green_start+1):(green_start + duration_to_check)) {
      # 判断当前时间点是否为绿灯
      light_status <- calculate_traffic_light(trajectory_data, PC1, time)
      # 如果当前时间点为绿灯
      if (light_status == 1) {
        # 计算绿灯持续时间
        green_duration <- time - green_start
```

```
      # 更新最长绿灯持续时间
      if (green_duration > longest_green_duration) {
        longest_green_duration <- green_duration
      }
    }
  }
 }
 return( longest_green_duration ) # 返回最长绿灯持续时间
}

# 绿灯持续时间统计
find_Green_light <- function(trajectory_data, PC1) {
 # 初始化最长绿灯持续时间
 END_GREEN <- numeric(max(trajectory_data$time0)-min(trajectory_data$time0)+1)

 for(time in (min(trajectory_data$time0)+2):(max(trajectory_data$time0)+1)){

  # 获取当前时间点和前一秒时间点的数据
  current_data <- trajectory_data[trajectory_data$time0 == time, ]
  previous_data <- trajectory_data[trajectory_data$time0 == time - 1, ]

  # 检查所有当前时间点的车辆
  for (vehicle_id in unique(current_data$vehicle_id)) {
    # 获取当前车辆和前一时刻的数据
    vehicle_current <- current_data[current_data$vehicle_id == vehicle_id, ]
    vehicle_previous <- previous_data[previous_data$vehicle_id == vehicle_id, ]

    for (cycle in unique(vehicle_current$cycle)){
      vehicle_current <- vehicle_current[vehicle_current$cycle==cycle, ]
      vehicle_previous <- vehicle_previous[vehicle_previous$cycle==cycle, ]

    # 检查车辆是否通过了交叉口
    if (nrow(vehicle_previous) > 0 && nrow(vehicle_current) > 0) {
      for (i in 1:min(nrow(vehicle_previous),nrow(vehicle_current))){
      # 检查车辆是否从交叉口一侧移动到另一侧
      if ((vehicle_previous[i,]$PC1 < PC1-0.1 && vehicle_current[i,]$PC1 > PC1+0.1) ||
          (vehicle_previous[i,]$PC1 > PC1+0.1 && vehicle_current[i,]$PC1 < PC1-0.1)) {
        END_GREEN[1:(time+1)]=END_GREEN[1:(time+1)]+1
      }
      }
    }
    }
  }
 }
 return(END_GREEN) # 返回最长绿灯持续时间
}
```

```r
# 定义函数来更新time列并添加cycle列
Stack_Cycle <- function(data, Greenlist) {
  a = Greenlist[1]
  b = Greenlist[2] - Greenlist[1]
  data$cycle <- ceiling((data$time - a - 1) / b)
  data$time0 <- (data$time - a - 1) %% b
  return(data)
}

# 贝叶斯直方图
Bayes_Discri <- function(RED_A1, GREEN_A1) {
  # 定义更好看的红色和绿色
  nice_red <- "#E43A1C" # 较亮的红色
  nice_green <- "#4DAF4A" # 较亮的绿色

  # 准备数据
  data <- data.frame(
    Index = rep(1:length(RED_A1), 2),
    Value = c(RED_A1, GREEN_A1),
    Type = rep(c("RED", "GREEN"), each=length(RED_A1))
  )

  # 绘制图形，使用自定义颜色
  p <- ggplot(data, aes(x=factor(Index), y=Value, fill=Type)) +
    geom_bar(stat="identity", position="dodge", width=0.8) +
    scale_fill_manual(values=c("RED"=nice_red, "GREEN"=nice_green)) +
    labs(x="A Single Cycle", y="Evidence", title="Histograms of RED and GREEN") +
    theme_minimal() +
    theme(axis.text.x = element_text(angle=45, vjust=1, hjust=1, size=5))

  # 输出图形
  print(p)
}

# 每个周期持续时间估计
find_RED_by_Cycle <- function(trajectory_data, PC1) {
  # 初始化停止时间最长的车辆ID、停止时间和开始停止时间
  longest_vehicle_id <- NA
  longest_stop_time <- -Inf
  start_stop_time <- Inf

  vehicle_ids <- unique(trajectory_data$vehicle_id)
  Cycles <- unique(trajectory_data$cycle)

  NUM_cycle <- c()
  TIME_red <- c()
  for (cycle in Cycles){
```

```
  longest_stop_time <- 0
 for (vehicle_id in vehicle_ids) {
  # 选择该车辆在位置 PC1 处的数据

  # 用于有误差时的情况
  vehicle_data <- trajectory_data[(trajectory_data$vehicle_id == vehicle_id) &
      (abs(trajectory_data$PC1 - PC1) < 0.00001) & (trajectory_data$cycle == cycle), ]
  # vehicle_data <- trajectory_data[trajectory_data$vehicle_id == vehicle_id &
      trajectory_data$PC1 == PC1 & trajectory_data$cycle == cycle, ]

  # 如果该车辆在位置 PC1处没有停止数据，则继续下一辆车
  if (length(unique(vehicle_data$time0)) <= 1) {
    next
  }
  # 获取该车辆停止的最开始时间和停止时间
  start_time <- min(vehicle_data$time0)
  stop_time <- max(vehicle_data$time0)
  # 计算该车辆在位置 PC1处的停止时间
  stop_duration <- stop_time - start_time + 1
  # 更新最长停止时间及其对应的车辆ID和开始停止时间
  if (stop_duration > longest_stop_time) {
    longest_stop_time <- stop_duration
    longest_vehicle_id <- vehicle_id
    start_stop_time <- start_time
  }
 }
  NUM_cycle <- c(NUM_cycle, cycle)
  TIME_red <- c(TIME_red, longest_stop_time)
 }



 data_frame <- data.frame(
  cycle = NUM_cycle, # 将NUM_cycle向量赋值给列名为'cycle'
  red = TIME_red   # 将TIME_red向量赋值给列名为'red'
 )
 # 返回结果
 return(data_frame)
}
find_GREEN_by_Cycle <- function(trajectory_data, PC1) {

 Cycles <- unique(trajectory_data$cycle)

 NUM_cycle <- c()
 TIME_green <- c()

 for(cycle_tem in Cycles){
```

```
  # 初始化最长绿灯持续时间
  longest_green_duration <- 0
    # 循环遍历需要检查的时间范围
  for(time in (min(trajectory_data$time0)):(max(trajectory_data$time0)+1)){

    # 获取当前时间点和前一秒时间点的数据
    current_data <- trajectory_data[(trajectory_data$time0 == time) & (trajectory_data$cycle ==
        cycle_tem), ]
    previous_data <- trajectory_data[(trajectory_data$time0 == time - 1) &
        (trajectory_data$cycle == cycle_tem), ]

    # 检查所有当前时间点的车辆
    for (vehicle_id in unique(current_data$vehicle_id)) {

      # 获取当前车辆和前一时刻的数据
      vehicle_current <- current_data[current_data$vehicle_id == vehicle_id, ]
      vehicle_previous <- previous_data[previous_data$vehicle_id == vehicle_id, ]
      # 检查车辆是否通过了交叉口
      if (nrow(vehicle_previous) > 0 && nrow(vehicle_current) > 0) {
          # 检查车辆是否从交叉口一侧移动到另一侧
          if ((vehicle_previous$PC1 < PC1-0.1 && vehicle_current$PC1 > PC1+0.1) ||
              (vehicle_previous$PC1 > PC1+0.1 && vehicle_current$PC1 < PC1-0.1)) {
            # 计算绿灯持续时间
            green_duration <- time - min(trajectory_data$time0)
            # 更新最长绿灯持续时间
            if (green_duration > longest_green_duration) {
              longest_green_duration <- green_duration
            }
        }
      }
    }
  }
  NUM_cycle <- c(NUM_cycle, cycle_tem)
  TIME_green <- c(TIME_green, longest_green_duration)
  }
  data_frame <- data.frame(
    cycle = NUM_cycle, # 将NUM_cycle向量赋值给列名为'cycle'
    green = TIME_green   # 将TIME_green向量赋值给列名为'green'
  )
  return(data_frame) # 返回最长绿灯持续时间
}

plot_stacked_bars <- function(RED_cycle_A1, GREEN_cycle_A1, GreenlistA1) {

  # 合并数据框
  cycle_A1 <- merge(RED_cycle_A1, GREEN_cycle_A1, by = "cycle")
```

```r
  # 使用滑动窗口计算相邻三个值的最大值
  cycle_A1$green <- rollapply(cycle_A1$green, width = 5, FUN = max, fill = NA, align = "center")
  cycle_A1$red <- rollapply(cycle_A1$red, width = 5, FUN = max, fill = NA, align = "center")

  # 计算'grey'列
  cycle_A1$grey <- GreenlistA1[2] - GreenlistA1[1] - cycle_A1$red - cycle_A1$green

  # 将数据框转换为长格式
  cycle_A1_long <- pivot_longer(cycle_A1, cols = c("green", "grey", "red"), names_to = "TYPE",
      values_to = "value")

  # 绘制堆叠柱状图
  p <- ggplot(cycle_A1_long, aes(x = factor(cycle), y = value, fill = TYPE)) +
    geom_col() + # 绘制柱状图
    scale_fill_manual(values = c("green" = "#4DAF4A", "grey" = "lightgrey", "red" = "#E43A1C"))
        +
    labs(
      x = "周期序号",
      y = "持续时间",
      title = "红绿灯分别持续时间",
      fill = "信号灯颜色"
    ) +
    theme_minimal() +
    theme(
      axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1),
      legend.title = element_text(size = 10),
      legend.text = element_text(size = 8)
    )
  # 输出图形
  print(p)
}
# 读取 CSV 文件
trajectory_data_A1 <- read.csv("B1.csv", header = TRUE)
trajectory_data_A2 <- read.csv("B2.csv", header = TRUE)
trajectory_data_A3 <- read.csv("B3.csv", header = TRUE)
trajectory_data_A4 <- read.csv("B4.csv", header = TRUE)
trajectory_data_A5 <- read.csv("B5.csv", header = TRUE)

trajectory_data_A5 <- trajectory_data_DB
trajectory_data_A4 <- trajectory_data_AB
trajectory_data_A3 <- trajectory_data_CB
trajectory_data_A2 <- trajectory_data_DA
trajectory_data_A1 <- trajectory_data_CD

#PCA降维
trajectory_data_A1_with_pca <- plot_pca_line(trajectory_data_A1)
trajectory_data_A2_with_pca <- plot_pca_line(trajectory_data_A2)
```

```r
trajectory_data_A3_with_pca <- plot_pca_line(trajectory_data_A3)
trajectory_data_A4_with_pca <- plot_pca_line(trajectory_data_A4)
trajectory_data_A5_with_pca <- plot_pca_line(trajectory_data_A5)

# 调用函数计算路口位置
crossing_location_A1 <- calculate_crossing_location(trajectory_data_A1_with_pca)
crossing_location_A2 <- calculate_crossing_location(trajectory_data_A2_with_pca)
crossing_location_A3 <- calculate_crossing_location(trajectory_data_A3_with_pca)
crossing_location_A4 <- calculate_crossing_location(trajectory_data_A4_with_pca)
crossing_location_A5 <- calculate_crossing_location(trajectory_data_A5_with_pca)

#调用计算绿灯起始时刻函数
GreenlistA1 <- Analysis_TotalCycle(trajectory_data_A1_with_pca)
GreenlistA2 <- Analysis_TotalCycle(trajectory_data_A2_with_pca)
GreenlistA3 <- Analysis_TotalCycle(trajectory_data_A3_with_pca)
GreenlistA4 <- Analysis_TotalCycle(trajectory_data_A4_with_pca)
GreenlistA5 <- Analysis_TotalCycle(trajectory_data_A5_with_pca)
PC1_A1 <- crossing_location_A1$PC1
PC1_A2 <- crossing_location_A2$PC1
PC1_A3 <- crossing_location_A3$PC1
PC1_A4 <- crossing_location_A4$PC1
PC1_A5 <- crossing_location_A5$PC1

StackA1 <- Stack_Cycle(trajectory_data_A1_with_pca, GreenlistA1)
StackA2 <- Stack_Cycle(trajectory_data_A2_with_pca, GreenlistA2)
StackA3 <- Stack_Cycle(trajectory_data_A3_with_pca, GreenlistA3)
StackA4 <- Stack_Cycle(trajectory_data_A4_with_pca, GreenlistA4)
StackA5 <- Stack_Cycle(trajectory_data_A5_with_pca, GreenlistA5)

# 调用函数找出在路口位置停止时间最长的车辆及其停止的最开始时间
resultA1 <- find_longest_stopped_vehicle(StackA1, PC1_A1)
resultA2 <- find_longest_stopped_vehicle(StackA2, PC1_A2)
resultA3 <- find_longest_stopped_vehicle(StackA3, PC1_A3)
resultA4 <- find_longest_stopped_vehicle(StackA4, PC1_A4)
resultA5 <- find_longest_stopped_vehicle(StackA5, PC1_A5)

Tgr_A1 <- GreenlistA1[2]-GreenlistA1[1]
Tgr_A2 <- GreenlistA2[2]-GreenlistA2[1]
Tgr_A3 <- GreenlistA3[2]-GreenlistA3[1]
Tgr_A4 <- GreenlistA4[2]-GreenlistA4[1]
Tgr_A5 <- GreenlistA5[2]-GreenlistA5[1]

duration_to_check_A1 <- Tgr_A1 - resultA1$stop_time
duration_to_check_A2 <- Tgr_A2 - resultA2$stop_time
duration_to_check_A3 <- Tgr_A3 - resultA3$stop_time
duration_to_check_A4 <- Tgr_A4 - resultA4$stop_time
duration_to_check_A5 <- Tgr_A5 - resultA5$stop_time
```

```
longest_green_duration_A1 <-
    calculate_longest_Green_light(trajectory_data_A1_with_pca,PC1_A1,GreenlistA1,duration_to_check_A1)
longest_green_duration_A2 <-
    calculate_longest_Green_light(trajectory_data_A2_with_pca,PC1_A2,GreenlistA2,duration_to_check_A2)
longest_green_duration_A3 <-
    calculate_longest_Green_light(trajectory_data_A3_with_pca,PC1_A3,GreenlistA3,duration_to_check_A3)
longest_green_duration_A4 <-
    calculate_longest_Green_light(trajectory_data_A4_with_pca,PC1_A4,GreenlistA4,duration_to_check_A4)
longest_green_duration_A5 <-
    calculate_longest_Green_light(trajectory_data_A5_with_pca,PC1_A5,GreenlistA5,duration_to_check_A5)

redtimeA1 <- resultA1$stop_time
redtimeA2 <- resultA2$stop_time
redtimeA3 <- resultA3$stop_time
redtimeA4 <- resultA4$stop_time
redtimeA5 <- resultA5$stop_time
# 打印结果
cat("A1路口总时间为", Tgr_A1, "秒。\n")
cat("A1路口红灯时间为", redtimeA1, "秒。\n")
cat("A1路口绿灯时间为", longest_green_duration_A1, "秒。\n")

cat("A2路口总时间为", Tgr_A2, "秒。\n")
cat("A2路口红灯时间为", redtimeA2, "秒。\n")
cat("A2路口绿灯时间为", longest_green_duration_A2, "秒。\n")

cat("A3路口总时间为", Tgr_A3, "秒。\n")
cat("A3路口红灯时间为", redtimeA3, "秒。\n")
cat("A3路口绿灯时间为", longest_green_duration_A3, "秒。\n")

cat("A4路口总时间为", Tgr_A4, "秒。\n")
cat("A4路口红灯时间为", redtimeA4, "秒。\n")
cat("A4路口绿灯时间为", longest_green_duration_A4, "秒。\n")

cat("A5路口总时间为", Tgr_A5, "秒。\n")
cat("A5路口红灯时间为", redtimeA5, "秒。\n")
cat("A5路口绿灯时间为", longest_green_duration_A5, "秒。\n")

#####################################################
RED_A1 <- resultA1$START_RED
GREEN_A1 <- find_Green_light(StackA1, PC1_A1)
Bayes_Discri(RED_A1, GREEN_A1)

RED_A2 <- resultA2$START_RED
GREEN_A2 <- find_Green_light(StackA2, PC1_A2)
Bayes_Discri(RED_A2, GREEN_A2)
```

```
RED_A3 <- resultA3$START_RED
GREEN_A3 <- find_Green_light(StackA3, PC1_A3)
Bayes_Discri(RED_A3, GREEN_A3)

RED_A4 <- resultA4$START_RED
GREEN_A4 <- find_Green_light(StackA4, PC1_A4)
Bayes_Discri(RED_A4, GREEN_A4)

RED_A5 <- resultA5$START_RED
GREEN_A5 <- find_Green_light(StackA5, PC1_A5)
Bayes_Discri(RED_A5, GREEN_A5)
##################################################


CycleTime_A1 <- GreenlistA1[2]-GreenlistA1[1]
CycleTime_A2 <- GreenlistA2[2]-GreenlistA2[1]
CycleTime_A3 <- GreenlistA3[2]-GreenlistA3[1]
CycleTime_A4 <- GreenlistA4[2]-GreenlistA4[1]
CycleTime_A5 <- GreenlistA5[2]-GreenlistA5[1]

RED_cycle_A1 <- find_RED_by_Cycle(StackA1, PC1_A1)
GREEN_cycle_A1 <- find_GREEN_by_Cycle(StackA1, PC1_A1)
plot_stacked_bars(RED_cycle_A1, GREEN_cycle_A1, GreenlistA1)

RED_cycle_A2 <- find_RED_by_Cycle(StackA2, PC1_A2)
GREEN_cycle_A2 <- find_GREEN_by_Cycle(StackA2, PC1_A2)
plot_stacked_bars(RED_cycle_A2, GREEN_cycle_A2, GreenlistA2)

RED_cycle_A3 <- find_RED_by_Cycle(StackA3, PC1_A3)
GREEN_cycle_A3 <- find_GREEN_by_Cycle(StackA3, PC1_A3)
plot_stacked_bars(RED_cycle_A3, GREEN_cycle_A3, GreenlistA3)

RED_cycle_A4 <- find_RED_by_Cycle(StackA4, PC1_A4)
GREEN_cycle_A4 <- find_GREEN_by_Cycle(StackA4, PC1_A4)
plot_stacked_bars(RED_cycle_A4, GREEN_cycle_A4, GreenlistA4)

RED_cycle_A5 <- find_RED_by_Cycle(StackA5, PC1_A5)
GREEN_cycle_A5 <- find_GREEN_by_Cycle(StackA5, PC1_A5)
plot_stacked_bars(RED_cycle_A5, GREEN_cycle_A5, GreenlistA5)
```

附录 C　问题四求解程序

```
rm(list = ls())
```

```r
setwd("C:\\Users\\86135\\Desktop\\华中杯\\B题：使用行车轨迹估计交通信号灯周期问题\\附件\\附件4")

library(ggplot2)
library(MASS) # 加载 MASS 包，其中包含了 PCA 函数
library(magrittr)
library(dplyr)

# 定义函数进行PCA计算并绘制散点图和PCA计算的直线
plot_pca_line <- function(data) {
  # 使用 PCA 计算数据的主成分
  pca_result <- princomp(data[, c("x", "y")])
  # 获取第一主成分的系数
  pc1_coef <- pca_result$loadings[, 1]
  # 提取第一主成分（一维数据）
  pc1 <- pca_result$scores[, 1]
  # 创建新的数据框保存一维数据
  pca_data <- data.frame(PC1 = pc1)
  trajectory_data_with_pca <- cbind(data, PC1 = pc1)
  # 计算直线方程的斜率和截距
  slope <- pc1_coef[2] / pc1_coef[1]
  intercept <- mean(data$y) - slope * mean(data$x)
  # 计算直线的两个端点，可以取 x 的最小和最大值
  min_x <- min(data$x)
  max_x <- max(data$x)
  # 计算直线的两个端点的坐标
  end_point1_y <- slope * min_x + intercept
  end_point2_y <- slope * max_x + intercept
  # 绘制散点图和PCA计算的直线
  plot_result <- ggplot(data, aes(x, y)) +
    geom_point(color = "darkblue", alpha = 0.6, size = 1.5) +
    geom_segment(aes(x = min_x, y = end_point1_y, xend = max_x, yend = end_point2_y),
                 color = "red", linetype = "dashed", size = 1) +
    labs(x = "X Coordinate", y = "Y Coordinate") +
    theme_minimal() +
    theme(legend.position = "none") # 移除图例
  # 输出绘图
  print(plot_result)

  return(trajectory_data_with_pca)
}

# Function to process initial plots and the location of Cross
Analysis_Cross <- function(data, flag=FALSE) {
  if(flag==TRUE){
    # Plot vehicle X coordinates over time with thicker lines
    plot_vehicle_x <- ggplot(data, aes(x=time, y=x, color=factor(vehicle_id))) +
      geom_line(size=0.5) +
```

```
    labs(x = "时间", y = "X坐标", title = "车辆X坐标随时间变化图") +
    theme(legend.position = "none") +
    ylim(-20, 30)
  print(plot_vehicle_x)
}

data <- data %>%
  arrange(vehicle_id, time) %>%
  group_by(vehicle_id) %>%
  mutate(stopped = (PC1 == lag(PC1)))

# Calculate and summarize stop times
stop_times <- data %>%
  arrange(vehicle_id, time) %>%
  group_by(vehicle_id) %>%
  filter(stopped == TRUE) %>%
  group_by(vehicle_id, PC1, x, y) %>%
  summarise(stop_time = n(), .groups = 'drop') %>%
  group_by(PC1, x, y) %>%
  summarise(total_time = sum(stop_time), .groups = 'drop')

# 生成PC1对应的(x, y)标签
stop_times$labels <- paste("(", stop_times$x, ", ", stop_times$y, ")", sep="")

# Plot the distribution of displacement vs total stop time using PC1 as the x-axis and (x, y)
    as labels
plot_displacement_vs_time <- ggplot(stop_times, aes(x=factor(PC1), y=total_time)) +
  geom_bar(stat="identity", fill="#3498db") + # Draw the bars
  labs(x = "位移 (x, y)", y = "总停止时间", title = "位移与停止时间的关系图") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle=45, hjust=1, vjust=1)) + # Set text alignment for
      better readability
  scale_x_discrete(breaks = stop_times$PC1, labels = stop_times$labels) # Set custom labels
      for x-axis

print(plot_displacement_vs_time)

# Identify the row with the maximum total_time
max_stop_time <- stop_times %>%
  filter(total_time == max(total_time))

return( max_stop_time)
}

# 定义函数找出在特定位置停止时间最长的车辆及其停时间
find_longest_stopped_vehicle <- function(trajectory_data, PC1) {
  # 初始化停止时间最长的车辆ID、停止时间和开始停止时间
```

```
  longest_vehicle_id <- NA
  longest_stop_time <- -Inf
  start_stop_time <- Inf

  # 获取轨迹数据中的唯一车辆ID
  vehicle_ids <- unique(trajectory_data$vehicle_id)

  # 遍历每辆车
  for (vehicle_id in vehicle_ids) {
    # 选择该车辆在位置 PC1 处的数据
    vehicle_data <- trajectory_data[trajectory_data$vehicle_id == vehicle_id &
        trajectory_data$PC1 == PC1, ]

    # 如果该车辆在位置 PC1处没有停止数据，则继续下一辆车
    if (nrow(vehicle_data) == 0) {
      next
    }

    # 获取该车辆停止的最开始时间和停止时间
    start_time <- min(vehicle_data$time)
    stop_time <- max(vehicle_data$time)

    # 计算该车辆在位置 PC1处的停止时间
    stop_duration <- stop_time - start_time + 1

    # 更新最长停止时间及其对应的车辆ID和开始停止时间
    if (stop_duration > longest_stop_time) {
      longest_stop_time <- stop_duration
      longest_vehicle_id <- vehicle_id
      start_stop_time <- start_time
    }
  }

  # 返回结果
  return(list(vehicle_id = longest_vehicle_id, stop_time = longest_stop_time, start_time =
      start_stop_time))
}

# Function to process final plots and Cycle
Analysis_Cycle <- function(data, Threshold_x, Threshold_y) {
  # Get last stops at {Threshold_x} meters and calculate time intervals
  last_stops <- data %>%
    filter(x == Threshold_x & y == Threshold_y & stopped == TRUE) %>%
    group_by(vehicle_id) %>%
    filter(time == max(time)) %>%
    ungroup() %>%
    distinct(time) %>%
```

```
    arrange(time)

  # Plot last stop times on a time axis
  time_point_plot <- ggplot(last_stops, aes(x=time, y=0)) +
    geom_point(color="gold", size=3) +
    geom_hline(yintercept=0, color="grey", linetype="solid", size=1) +
    geom_text(aes(label = format(time, digits = 4)), angle = 45, vjust = -1, hjust = -0.1,
        color = "grey", size=3) +
    labs(x = "时刻", title = "路口处每辆车的最后停止时间点") +
    xlim(range(data$time)) +
    theme_minimal() +
    theme(axis.text.y = element_blank(), axis.ticks.y = element_blank(), axis.title.y =
        element_blank(),
         panel.background = element_blank(), panel.grid.major.y = element_blank(),
             panel.grid.minor.y = element_blank())
  print(time_point_plot)

  # Calculate time intervals
  time_intervals <- diff(last_stops$time)
  interval_data <- data.frame(interval = time_intervals)

  # Plot the distribution of time intervals
  time_interval_plot <- ggplot(interval_data, aes(x=factor(interval), y=..count..)) +
    geom_bar(stat="count", fill="#3498db") +
    labs(x = "时间间隔", y = "频数", title = "路口处间隔时间(总周期统计)") +
    theme_minimal() +
    theme(axis.text.x = element_text(hjust = 1))
  print(time_interval_plot)
}

# 读取 CSV 文件
trajectory_data_D <- read.csv("D.csv", header = TRUE)

vehicle_data_list <- split(trajectory_data_D, trajectory_data_D$vehicle_id)

# 初始化一个空的数据框用于保存符合条件的车辆轨迹数据
trajectory_data_AB <- data.frame()
trajectory_data_AC <- data.frame()
trajectory_data_AD <- data.frame()

trajectory_data_BA <- data.frame()
trajectory_data_BC <- data.frame()
trajectory_data_BD <- data.frame()

trajectory_data_CA <- data.frame()
trajectory_data_CB <- data.frame()
trajectory_data_CD <- data.frame()
```

```
trajectory_data_DA <- data.frame()
trajectory_data_DB <- data.frame()
trajectory_data_DC <- data.frame()

# 遍历车辆数据列表
for (vehicle_id in unique(trajectory_data_D$vehicle_id)) {
  vehicle_data <- trajectory_data_D[trajectory_data_D$vehicle_id == vehicle_id, ]

  # 提取车辆的起始点和终止点
  start_point <- vehicle_data[1, c("x", "y")]
  end_point <- vehicle_data[nrow(vehicle_data), c("x", "y")]

  if (start_point$x < -250 && end_point$y > 250) {
    trajectory_data_AB <- rbind(trajectory_data_AB, vehicle_data)
  }
  if (start_point$x < -250 && end_point$x > 250) {
    trajectory_data_AC <- rbind(trajectory_data_AC, vehicle_data)
  }
  if (start_point$x < -250 && end_point$y < -250) {
    trajectory_data_AD <- rbind(trajectory_data_AD, vehicle_data)
  }


  if (start_point$y > 250 && end_point$x < -250) {
    trajectory_data_BA <- rbind(trajectory_data_BA, vehicle_data)
  }
  if (start_point$y > 250 && end_point$x > 250) {
    trajectory_data_BC <- rbind(trajectory_data_BC, vehicle_data)
  }
  if (start_point$y > 250 && end_point$y < -250) {
    trajectory_data_BD <- rbind(trajectory_data_BD, vehicle_data)
  }


  if (start_point$x > 250 && end_point$x < -250) {
    trajectory_data_CA <- rbind(trajectory_data_CA, vehicle_data)
  }
  if (start_point$x > 250 && end_point$y > 250) {
    trajectory_data_CB <- rbind(trajectory_data_CB, vehicle_data)
  }
  if (start_point$x > 250 && end_point$y < -250) {
    trajectory_data_CD <- rbind(trajectory_data_CD, vehicle_data)
  }


  if (start_point$y < -250 && end_point$x < -250) {
```

```
    trajectory_data_DA <- rbind(trajectory_data_DA, vehicle_data)
  }
  if (start_point$y < -250 && end_point$y > 250) {
    trajectory_data_DB <- rbind(trajectory_data_DB, vehicle_data)
  }
  if (start_point$y < -250 && end_point$x > 250) {
    trajectory_data_DC <- rbind(trajectory_data_DC, vehicle_data)
  }
}
```