基于行车轨迹估计交通信号灯周期问题的研究

摘　要

本文通过使用样本车辆的轨迹行驶数据估计交通信号灯的周期。计算出不同时间的红灯时长和绿灯时长，将这些信息通过导航告知司机，为司机提供更精准的导航服务，并且研究了不同因素对于交通信号灯周期估计偏差的影响。

问题一中，在假设信号灯周期固定不变且已知所有车辆行车轨迹的前提下，本文计算出了附件 1 中五个路口交通信号灯的周期。利用速度公式计算出车辆每一时刻的速度，以 1.39m/s 作为阈值判断车辆处于行驶状态还是停车状态，从而进一步计算车辆的红灯等待时间与绿灯行驶时间。对于红灯时长计算，假设多个车辆在红灯下的等待时间成均匀分布，画出其**均匀分布图**，图像的**最大值或者拐点**（存在意外停车的情况）即为红灯时长；对于绿灯时长，根据速度确定第一辆车在绿灯前的启动时刻以及之后的车辆在信号灯前停止的时刻，二者**作差**取**最小值**即为绿灯时长。求得到的 A1-A5 路口信号灯周期具体结果在正文表 3 中展示。

问题二中，样本车辆比例、车流量、定位误差等因素均会影响信号灯周期的估计精度。对于样本车辆比例的影响分析，以附件 1 中的所有车辆作为总体数据，选取了不同的样本车辆比例，之后使用**蒙特卡罗法**进行**随机抽样**，从总体数据中抽取选定比例的样本车辆，重复模型一的计算方法得到样本数据的信号灯周期，并进行误差计算。结果发现，当样本车辆比例逐渐减少时，信号灯周期的估计结果误差逐渐增大，说明**增大样本车辆比例会提高模型估计精度**；对于车流量的影响分析，使用 **K-means 聚类**方法将单位时间内车辆出现次数即车流量分为两类：高车流量与低车流量，两组数据重新使用模型一的方法计算信号灯周期并计算误差。高车流量的误差明显低于低车流量的误差。这说明，**增大车流量会提高模型估计精度**；对于定位误差的影响分析，在原数据的基础上添加**不同标准差的高斯噪声**，分别计算所造成的误差。结果证明，随着标准差的增大，信号灯周期的估计偏差增大。因此，**定位误差越大，模型的估计精度就越低**。当计算附件 2 中不同路口的信号灯周期时，为减弱定位误差对估计结果的影响，使用**二次贝塞尔曲线**进行拟合平滑，之后再次使用模型一中的方法求得信号灯周期。

问题三中，首先利用车辆启动时间列方程组解得 C1-C6 路口红绿灯总周期，并根据模型一、二中的方法求得 C1-C6 路口的绿灯车辆行驶时间。为识别信号灯周期切换时刻，采用**滑动窗口法**得出绿灯周期的时间序列，确定阈值识别时间序列时间序列中的转折点，作为绿灯周期转换时刻。最后根据红绿灯总周期和绿灯周期求出红灯周期，具体结果在正文表 10 中展示。

问题四中，首先将车辆按照起始坐标进行 K-means 聚类，确定车辆来向。之后通过对车辆从驶入路口到驶出路口时刻转向角的识别，确定其转向。将根据来向和转型分类后的数据分别按照模型一、二、三的思路重复操作，分别计算得出东西和南北方向信号灯的周期。结果发现，在连续两个小时内，该路信号灯周期发生了**一次切换**，具体切换时刻以及切换前后的不同方向的信号灯周期在正文表 11 展示。

关键词：均匀分布图　K-means 聚类　高斯噪声　二次贝塞尔曲线　滑动窗口法　蒙特卡罗法随机抽样

目录

一、 问题重述

1.1 问题背景

科学技术的发展使得人们的出行更加便利，驾驶人也越来越依赖出行导航。为了节省出行时间，人们往往会选择交通信号灯少的路段或者希望根据红绿灯周期来提前调整行驶速度以规避红灯等待的时间。因此，电子地图公司想要掌握城市中所有交通信号灯的红绿灯变换周期以为司机提供更精准的导航服务。

但由于技术或者车流量过少等原因，许多信号灯并未联网，公司无法直接从交通管理部门那里取得完整的数据。若每个路口都安排人员手动记录信号灯的周期信息，则会出现耗资较大等情况，并不现实。所以，公司决定利用广大客户的行驶轨迹数据来估算交通信号灯的变换周期。通过估算出来的周期数据来为驾驶人提供精确的导航服务。

1.2 问题要求

问题一：在假设交通信号灯的变换周期是固定的，并且已经掌握了所有车辆的行驶轨迹数据的前提下，通过建立数学模型估算每个信号灯的红绿灯变换周期。利用附件 1 中所提供的 5 个不同路口在一个方向连续 1 小时内车辆的行驶轨迹数据计算出这些路口对应方向上的信号灯周期，并按照规定的格式填写到表 1 中。

问题二：实际上，该公司的产品仅被部分用户所使用，因此只能获取到部分样本车辆的行车轨迹数据。此外，这些轨迹数据可能存在定位误差。在评估上述模型的估计精度时，综合考虑样本车辆的比例、车流量的多少以及定位误差的大小等因素。利用附件 2 中提供的另外 5 个不相关路口各自一个方向连续 1 小时内的样本车辆轨迹数据来求出这些路口相应方向的信号灯周期，并按照相同的格式要求填写到表 2 中。

问题三：如果交通信号灯的周期存在变动的可能性，建立数学模型迅速发现这种变化，并确定变化后的新周期。附件 3 中提供了六个不同路口各自一个方向连续两小时内的样本车辆轨迹数据。判断这些路口在指定方向上的信号灯周期在这段时间内是否有所变动，并努力找出周期切换的具体时刻，以及新旧周期的参数。按照规定的格式将这些信息填入表 3 中，并明确指出识别周期变化所需的时间和条件。

问题四：附件 4 详细记录了某路口连续两小时内所有方向样本车辆的行驶轨迹数据。基于这些数据，尝试识别并确定这个路口信号灯的周期，以便进一步了解交通流量和信号灯的运行规律。

二、 问题分析

2.1 问题一的分析

在假定信号灯周期固定且已知所有车辆的行车轨迹的前提下，利用 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，求出这些路口相应方向的信号灯周期。多个样本车辆在目标红绿灯前的停车时长的分布概率处于均匀分布。基于该均匀分布的理论，所有样本车辆的最大停车时长则为红灯时长。但考虑到实际道路情况，车辆中可能会存在意外停车，而这种意外停车的样本车辆的停车时长则会较长，并且会脱离上文中提到的均匀分布规律而存在。将样本车辆按照停车时长进行

排序后，确定按照排序结果进行排列的样本车辆的停车时长的分布信息，将该分布信息中遵循均匀分布规律的停车时长表示为趋于线性分布的点，这些线性分布的点可以被拟合为一条斜线。此外，表示意外停车车辆的停车时长的点在分布信息中会表现出会脱离上述斜线，从而使得该斜线上产生一个拐点，并且在该拐点处向上突变，该拐点可以称之为分布信息中的突变拐点。找到该突变拐点，即可确定该目标红绿灯的真正红灯时长。

为确定绿灯周期，将附件中由静止开始启动的第一辆车的所处时间与之后由行驶到静止的第一辆车的所处时间做差值。但考虑到样本中部分驾驶员的驾驶行为并不规范以及由于各种意外产生的刹车行为，因此极少数识别出的时间段并不是绿灯周期，此时将忽略不计。同时考虑到由行驶到静止的第一辆车可能是在绿灯熄灭红灯亮起的之后一段时间出现，这样会造成计算的绿灯周期偏大，因此，取所有计算结果（即上述差值）的最小值作为绿灯周期。

2.2 问题二的分析

由于并不是所有用户都使用该公司的产品，获取的数据只是部分车辆的轨迹数据。考虑到样本车辆比例、车流量、定位误差等因素，问题一中的模型所获得的信号灯周期可能会有偏差。要讨论这些因素对于模型估计精度的影响，可以通过变动不同的样本车辆比例、车流量、定位数据，计算最终计算周期与模型一中计算结果的偏差，以此来体现以上因素对模型估计精度的影响大小。具体思路如下图所示：

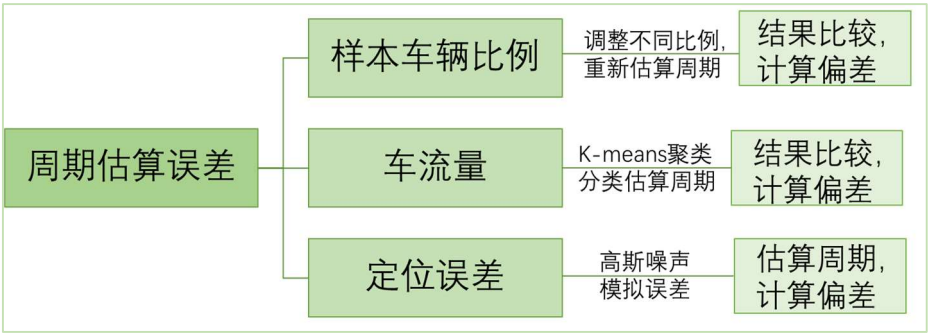

图 1：问题二思维导图

2.3 问题三的分析

由于现实生活中交通信号灯可能通过调节红绿灯时长来改善交通状况，因此信号灯的周期可能会变化。为尽快识别这种变化，通过使用模型一、二中的方法计算出附件 3 中不同路口的车辆等待红灯的时长以及连续经过绿灯的时长。之后通过一定方法识别周期发生明显变化的转折点，从而确定周期转换的时刻。最终在周期转换的前后，依旧重复模型二的计算方法分别求出周期转换前后的红灯时长与绿灯时长。

2.4 问题四的分析

附件 4 是某一路口连续两个小时内各个方向的样本车辆轨迹数据。由于样本车辆向不同方向行驶，要求得该路口指示不同方向的信号灯周期，首先要将样本车辆行驶轨迹数据按照行进方向进行分类。分类之后根据不同轨迹数据分别求出不同方向的红绿灯时长。

三、 模型假设

假设 1：在理想情况下，样本车辆中所有停车时长都是等红绿灯的时长，则基于该均匀分布的理论可以得知，所有样本车辆的最大停车时长则为红灯时长；

假设 2：在理想状况下，司机等待时间大于红灯时间的情况源于出现意外停车，例如一些车辆并不是因为等红灯停车，而是路边停车、发生交通事故或者其他原因导致在红绿灯前停车。而这种意外停车的样本车辆的停车时长则会较长，并且会脱离上文中提到的均匀分布规律而存在；

假设 3：多个样本车辆在目标红绿灯前的停车时长的分布概率处于均匀分布。

四、 符号说明

符号	说明	单位
v	车辆行驶速度	m/s
x	车辆位置横坐标	m
y	车辆位置纵坐标	m
u	信号灯周期偏差	-
T	附件 1 中整体数据所计算出来的信号灯周期	s
T'	附件 1 中选取一定比例的数据所计算出来的信号灯周期	s
Z^+	任意正整数	-

五、 模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 模型的建立

为计算 5 个不相关路口相应方向的红灯周期，在获取了所有车辆的行车轨迹数据后，首先要识别出数据中车辆何时停止以及停车时长。根据现行交通规则，将速度低于 1.39m/s（5 公里每小时）的车辆判定为处于停车状态。利用车辆坐标变化与时间变化作比计算车辆行驶速度，计算公式如下：

$$v = \frac{\sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2}}{t_n - t_{n-1}} \tag{1}$$

这里，t_{n-1}表示上一时间点，t_n表示下一时间点，(x_{n-1},y_{n-1})表示上一时间点车辆的位置，(x_n,y_n)表示下一时间点车辆的位置。

停车时长即为速度低于 1.39m/s 的持续时间。在确定停车时长之后，可以基于车辆在该目标红绿灯前的停车时长由小及大对车辆进行排序，之后基于排序结果对每个车辆赋予排序序号。

在忽略目标红绿灯所在道路以及红绿灯整个周期的影响后，每个车辆驶入红绿灯前并由于红灯而停车的时间 t1 与目标红绿灯的相位关系相互独立，所以多个车辆在目标红绿灯前的停车时长的分布概率处于均匀分布。

因此，在理想情况下，如果车辆中所有停车时长都是等红绿灯的时长，则基于该均匀分布的理论可以得知，所有车辆的最大停车时长则为红灯时长。然而，实际

道路情况是，样本车辆中可能会存在意外停车，例如一些车辆并不是因为等红灯停车，而是路边停车、发生交通事故或者其他原因导致在红绿灯前停车。而这种意外停车的样本车辆的停车时长则会较长，并且会脱离上文中提到的均匀分布规律而存在。因此，为了挖掘出目标红绿灯的真正红灯时长，可以先找出车辆的停车时长的分布信息，进而再基于分布信息以及车辆等待红灯时长的均匀分布规律挖掘出目标红绿灯的红灯时长。

在将车辆按照停车时长进行排序后，确定按照排序结果进行排列的车辆的停车时长的分布信息，该分布信息中遵循均匀分布规律的停车时长可以表示为趋于线性分布的点，该些线性分布的点可以被拟合为一条斜线，该斜线为从排序在最前的第一辆样本车辆的停车时长开始斜向上的线。

此外，该分布信息中还存在一些不遵循均匀分布规律的停车时长，也即意外停车的车辆的停车时长，表示这些停车时长的点在分布信息中会表现出会脱离上述斜线，从而使得该斜线上产生一个拐点，并且在该拐点处向上突变，该拐点可以称之为分布信息中的突变拐点。

基于上述分析可以知道，在确定按照排序结果排列的样本车辆的停车时长的分布信息之后，只要从该分布信息中找出不再遵循均匀分布规律的突变拐点，即可确定该目标红绿灯的真正红灯时长；也即可以基于该突变拐点对应的停车时长确定目标红绿灯的红灯时长。需要说明的是，上述突变拐点可以为依然靠近上文中提到的斜线上的点，也即突变拐点可以理解为分布信息中依然遵循均匀分布规律的最后一个或者最后一组点。

对于绿灯周期，将附件中由静止开始启动的第一辆车的所处时间与之后由行驶到静止的第一辆车的所处时间做差值，也即在一段时期内车辆连续通过的时间。考虑到由行驶到静止的第一辆车可能是在绿灯熄灭红灯亮起的之后一段时间出现，这样会造成计算的绿灯周期偏大，因此，取所有计算结果（即上述差值）的最小值作为绿灯周期。具体公式如下：

$$T_{green} = \mathrm{Min}(startTime_i - startTime_{i-1}) \tag{2}$$

其中 i=1，2...，n，$startTime_i$是第 i 辆车辆的启动时间，$startTime_{i-1}$是前一车辆的启动时间。

为精确判断红绿灯前车辆的启动时间，本文对红绿灯缓冲域进行识别。车辆在红灯前停止会导致其数据量增加，这里用坐标众数确定缓冲域中心，结合数据实际情况，以 10m 为半径确定红绿灯缓冲域，以在缓冲域中的启动和停止的车辆计算上文中所说的启动时间和停止时间。

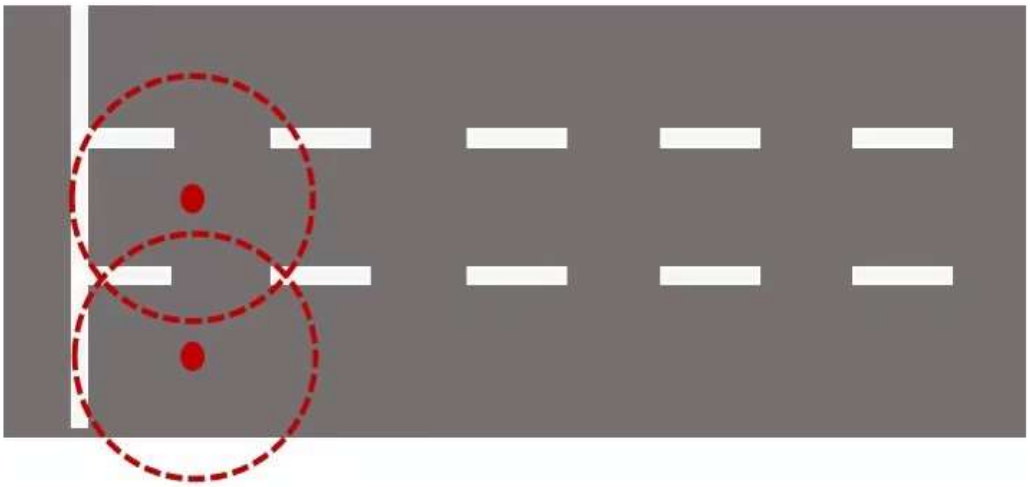

图 2：域点识别示意图

5.1.2 模型的求解

利用 Python 对上述进行求解，求得的车辆速度部分结果如下表所示：

表 1 附件 1 中车辆速度部分求解结果

time	Vehicle_id	x	y	v
80	18	11.43	1.6	0.13
81	18	11.41	1.6	0.02
82	18	11.4	1.6	0.01
83	18	11.4	1.6	0
84	18	11.4	1.6	0
85	18	11.4	1.6	0
86	18	11.4	1.6	0
87	18	11.4	1.6	0
88	18	11.4	1.6	0
89	18	11.4	1.6	0
90	18	11.4	1.6	0
91	18	11.4	1.6	0
92	18	11.4	1.6	0
93	18	11.4	1.6	0
94	18	11.4	1.6	0
95	18	11.4	1.6	0
96	18	11.4	1.6	0
97	18	11.4	1.6	0
98	18	11.4	1.6	0
99	18	11.4	1.6	0
100	18	11.4	1.6	0
101	18	11.4	1.6	0
102	18	11.4	1.6	0
103	18	11.4	1.6	0
104	18	11.4	1.6	0
105	18	9.57	1.6	1.83

根据表中数据，求得每辆车车速低于 1.39m/s 的持续时间，最终按照持续时间排序画图，所得结果如下所示：

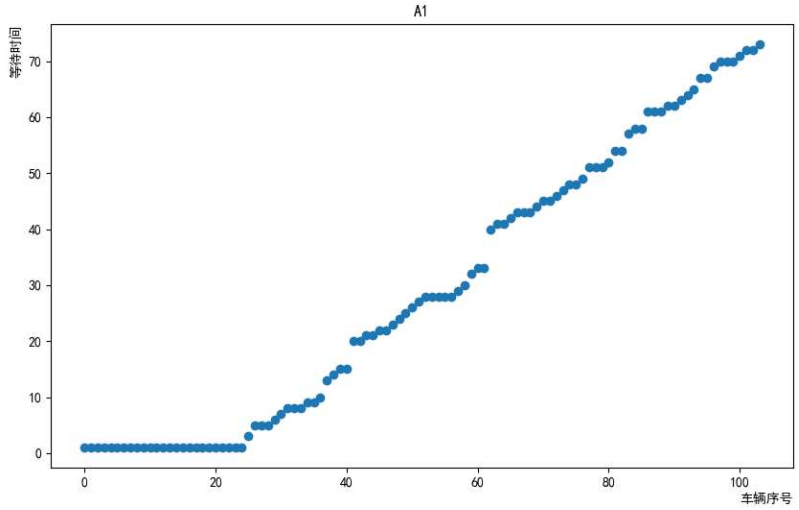

图 3：A1 路口车辆等待时间分布图

从图 1 中可以看出，A1 路口的等待时间基本服从线性的均匀分布，但并没有拐点出现，这是因为附件一数据是一个小时以内的车辆轨迹数据，时间相对较短，并没有意外情况发生，所有车辆均正常停车。考虑到部分车辆可能在红灯亮起之后的时间点到达信号灯前等待，等待时间并不是完整的红灯时长。因此，A1 路口的红灯时长即可视为均匀分布的等待时间的最大值 73s。

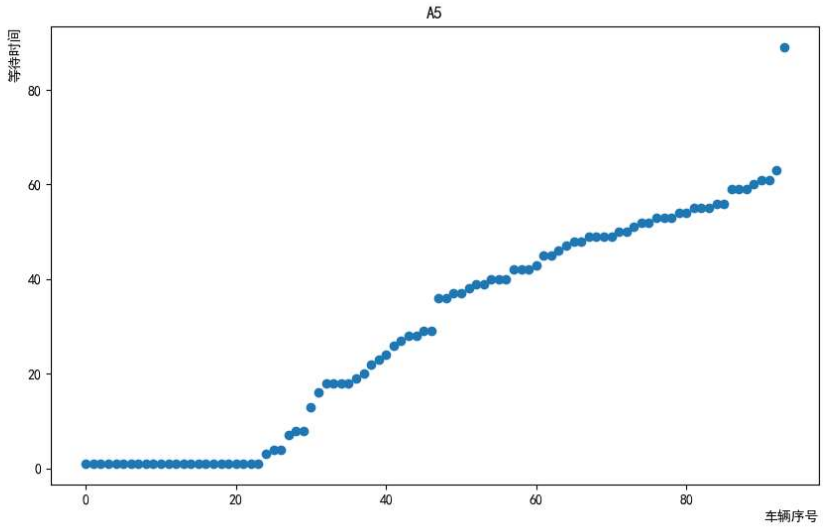

图 4：A5 路口车辆等待时间均匀分布图

从图 2 中看到，A5 路口除正常的均匀分布的等待时间之外，还有意外停车的情况，与预想情况一致，取等待时间的突变拐点作为红灯时长，最终结果为 64s。

由于求解结果过多，A2、A3、A4 三个路口的车辆等待时间均匀分布图以及五个路口的车辆等待时间具体数据均在附录中展示。

在根据所求车辆的速度识别出第一辆车启动时刻与后面的第一辆车停止的时刻，

做差值之后取最小值即为绿灯周期时长。需要注意的一点是，在附件 1 时间截止的最后，部分绿灯行驶时间可能被截断。为排除这种情况的干扰，将最后的绿灯行驶时清除。表 2 为 Python 运行部分结果示例：

表 2：A1 路口绿灯时长统计

绿灯时长	启动时刻	启动车辆 ID	停止时刻	停止车辆 ID
33	1994	821	2027	862
34	1364	549	1398	590
34	944	364	978	407
35	524	180	559	226
36	1784	731	1820	773
36	419	150	455	180
37	3464	1466	3501	1506
39	3254	1396	3293	1414

最终根据以上结果求得的五个路口的红绿灯周期如下表所示：

表 3：A1-A5 路口信号灯周期

路口	A1	A2	A3	A4	A5
红灯时长（秒）	73	57	81	70	63
绿灯时长（秒）	33	34	25	22	26

5.2 问题二模型的建立与求解

5.2.1 模型的建立

考虑样本车辆比例、车流量、定位误差等因素，问题一中的模型所获得的信号灯周期可能会有偏差。要讨论这些因素对于模型估计精度的影响，可以通过变动不同的样本车辆比例、车流量、定位数据，计算最终计算周期与模型一中计算结果的偏差，以此来体现以上因素对模型估计精度的影响大小。

对于探究样本车辆比例对于模型估计精度的影响，将附件 1 中的车辆数据作为总体数据，采用蒙特卡罗法从附件 1 中抽取不同数量比例的车辆，蒙特卡罗方法的基本原理是利用随机数来模拟样本车辆，其核心思想是利用随机性来解决确定性问题。之后用有蒙特卡罗法抽样调查附件 1 中不同比例（从 10%开始，以 10%作为间距，直到抽样比例达到 90%）的车辆重复模型一的计算方法求得不同路口信号灯的周期，通过与模型一中计算结果进行对比，计算出不同样本车辆比例下的估计偏差。估计偏差的计算公式如下：

$$u = \left| \frac{T' - T}{T} \right| * 100\% \tag{3}$$

对于车流量的不同，将附件 1 中 5 个路口分别按照时间进行切割，每个路口分成 5 组，计算出该组车流量的大小。车流量指的是单位时间内所通过车的数量。计算出车流量之后，采用 K-means 聚类的方法，将五组数据分为两类，分别为高车流量和低车流量。K-means 聚类方法的步骤如下：

Step 1: 根据设定的簇的数目 2，随机选择 2 个数据点作为初始的簇中心;

Step 2: 对于其他所有的数据点，计算它们与这 2 个初始簇中心的距离，然后将它们归类到与之最近的簇中心所属的簇;

Step 3: 对于每个簇，计算其中所有数据点的均值，并将这个均值作为新的簇中心;

Step 4: 这个过程会不断重复，直到达到某个停止条件，比如簇中心不再发生变化，或者迭代次数达到预设的最大值。

之后分别将 K-means 聚类后的不同车流量的数据带入模型一中计算得出其信号灯的周期，通过与总车流量计算得出的信号灯周期进行对比，来得出车流量大小对于信号灯周期估计精度的影响。

当讨论定位误差对于估计结果的影响时，在附件 1 原有的轨迹数据中加入高斯噪声形成位置偏差。基本原理是通过对原有数据加入均值为 0、方差为σ_i^2的高斯噪音形成定位偏差，再将具有不同程度定位偏差的数据重新利用模型一的方法求出新的红绿灯时长，与原来的红绿灯时长做对比，计算出估计偏差。

在求解附件 2 中五个路口相应方向的信号灯周期时，考虑到其所提供的轨迹数据可能存在定位误差，为尽可能排除这种因素的影响，本文采用二次贝塞尔曲线进行拟合平滑。二次贝塞尔曲线由三个点定义，分别是起始点（P0）、控制点（P1），和结束点（P2）。曲线从起始点出发，经过控制点，最终到达结束点。曲线的形状由控制点的位置决定，控制点引导了曲线的弯曲和走势。以此去除定位存在的噪声，平滑曲线来使定位结果更加准确。减少定位误差对轨迹数据的影响之后，采用模型一的方法计算出 B1-B5 五个路口的红绿灯时长。

5.2.2 模型的求解

通过调整不同的样本车辆比例所计算出来的估计偏差部分结果如下表所示：

表 4：A1 路口不同样本比例红灯时长估算偏差

样本车辆比例	总数据红灯时长	样本数据红灯时长	估计偏差
0.1	73	66.609	8.755%
0.2	73	68.562	6.079%
0.3	73	69.532	4.751%
0.4	73	70.1625	3.887%
0.5	73	70.591	3.300%
0.6	73	70.907	2.867%
0.7	73	71.15485714	2.528%
0.8	73	71.36025	2.246%
0.9	73	71.53066667	2.013%

表 5：A1 路口不同样本比例绿灯时长估算偏差

样本车辆比例	总数据绿灯时长	样本数据绿灯时长	估计偏差
0.5	33	34.79	5.424%
0.6	33	34.08	3.273%
0.7	33	33.652	1.976%
0.8	33	33.378	1.145%
0.9	33	33.167	0.506%

其他四个路口的估计偏差在附录中展示。从表 4、5 中可以看出，无论是红灯周

期还是绿灯周期，随着样本车辆比例的提高，估计偏差都会逐渐减小。这说明样本车辆比例提高即将公司产品更广泛的推广吸引更多用户有利于提高信号灯周期估计精度。

在探究不同车流量对于估计偏差的影响时，首先使用 K-means 聚类方法将车流量分为了高车流量和低车流量两大类，不同车流量下红绿灯时长估计偏差结果如表 6 所示：

表 6：不同车流量的红绿灯时长估计偏差

车流量	信号灯	A1	A2	A3	A4	A5
高	绿	0.000%	0.000%	0.000%	40.909%	0.000%
	红	0.000%	1.786%	0.000%	2.778%	1.563%
低	绿	12.121%	8.824%	56.000%	22.727%	15.385%
	红	5.479%	10.714%	4.938%	0.000%	0.000%

从表中可以看出，高车流量下的估计偏差明显小于低车流量的估计偏差。因此，在其他条件不变的情况下，车流量减小会降低模型的估计精度。

加入具有不同标准差的高斯噪音之后，求得新的红绿灯时长，利用公式（2）计算定位误差所引起的估计偏差。下图展示了估计偏差随高斯分布的标准差的变化情况。

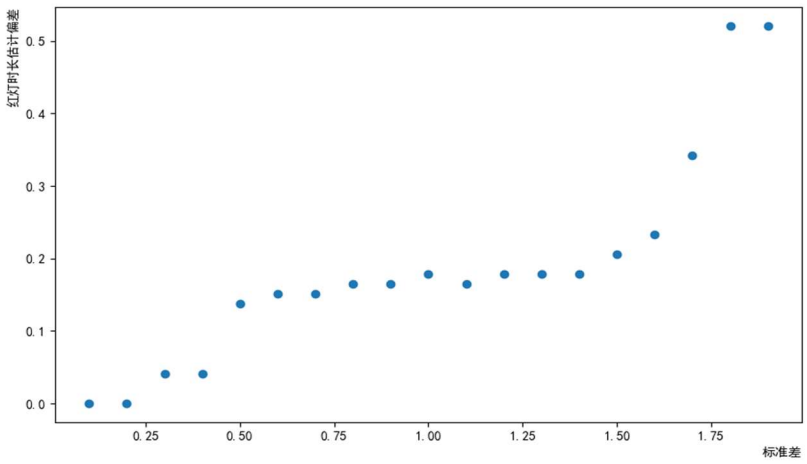

图 5：A1 路口红灯时长估计偏差变化情况

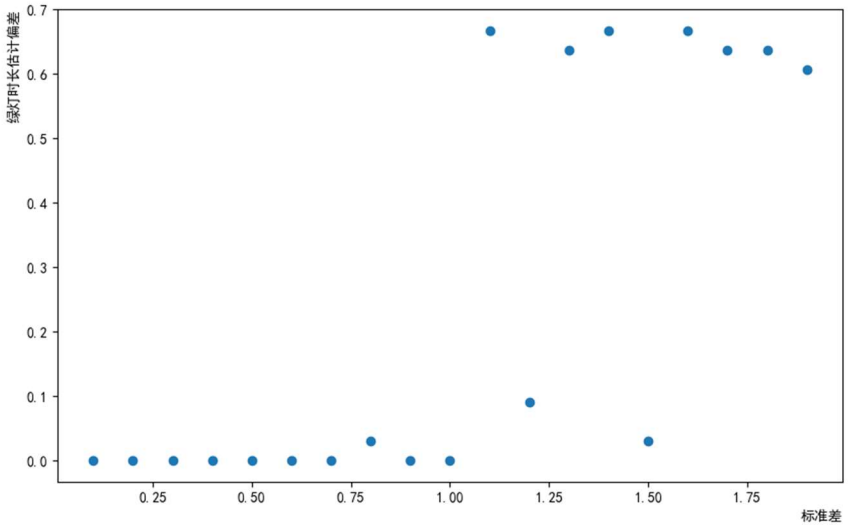

图 6：A1 路口绿灯时长估计偏差变化情况

从图 4 中可以看出随着高斯噪声标准差的逐渐变大，红灯时长估计偏差逐渐变大。由于高斯噪声标准差越大，车辆的定位误差的越大，因此得出结论定位误差增大会造成红灯时长估计偏差增大。图 5 展现了 A1 路口绿灯时长估计偏差的变化情况，当标准差保持较低水平时，绿灯时长的估计偏差很小，几乎趋于零；而当标准差大于 1 时，绿灯时长的估计偏差会急剧上涨，偏差大多数高于 60%。因此定位误差增大也会使绿灯的估计偏差增大，从而降低模型的估计精度。

将附件 2 中的轨迹数据进行二次贝塞尔曲线平滑处理之后减小定位误差之后计算出的信号灯周期结果如下表所示：

表 7：B1-B5 路口信号灯周期

路口	B1	B2	B3	B4	B5
红灯时长（秒）	78	83	55	78	89
绿灯时长（秒）	28	34	38	37	31

5.3 问题三模型的建立与求解

5.3.1 模型的建立

根据模型假设，当车辆起步速度从零开始高于 1.39m/s 时本文认为交通信号灯由红灯变为绿灯；当后车速度开始低于 1.39m/s 时本文认为交通信号灯由绿灯变为红灯。而由于车辆在红灯前停止的时间点随机的，在绿灯时的启动时间是准确的，因而用相邻的车辆启动时间作为红绿灯总周期。具体计算公式如下：

$$\begin{cases} T \times Z_1 = startTime_2 - startTime_1 \\ T \times Z_2 = startTime_3 - startTime_2 \\ \cdots\cdots \\ T \times Z_n = startTime_{n+1} - startTime_n \end{cases} \tag{4}$$

其中，T是一个红绿灯周期时长，Z 为任意非负整数，startTime是绿灯开始时间。

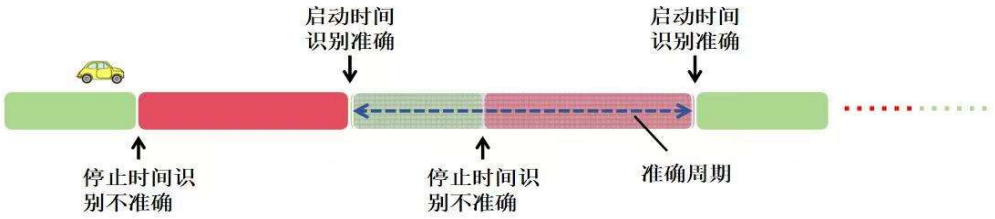

图 7：信号灯总周期识别示意图

为检测周期变化，采用滑动窗口法来识别周期变化的时刻。滑动窗口法的基本原理是利用两个指针，形成一个可移动的窗口，在这个窗口内执行特定的操作。

这个窗口就像一块“滑动板”，在数据序列上从左到右滑动。通常，有一个左指针和一个右指针，它们分别表示窗口的起始位置和结束位置。随着右指针向下移动，窗口会扩大，可能包含新的元素；同时，根据需要，左指针也可能向下移动，使窗口缩小，排除不再需要的元素。在这个过程中，寻找窗口内的元素的最小值。这些操作的结果可能随着窗口的移动而改变。将操作的结果绘制成散点图，设置 5 秒作为绿灯时长转变的阈值，当结果突变大于等于 5 秒时，可以认定在该点绿灯周期发生了转变。最终使用附件 3 中的轨迹数据计算车辆速度，利用模型一同样的方法识别车辆的启动时间（速度从低于 1.39m/s 增加到高于 1.39m/s 时即为启动时间）和绿灯行驶时间，找到与绿灯周期转变点所对应的时刻即可确定绿灯时长的转换时刻。具体步骤如下：

Step1：确定红绿灯总周期变化；
Step2：设定滑动窗口的大小为 10；
Step3：从时间序列的起始点开始，以窗口大小为步长，在时间序列上滑动窗口。每次移动窗口时，都会生成一个新的子序列，包含窗口内的数据点；
Step4：对于每个窗口内的数据子序列，计算其最小值作为相应时间内的绿灯时长；
Step5：按照时间顺序将最小值合并成一组序列，并绘制相应的绿灯时间散点图。
Step6：设定阈值，识别绿灯时间散点图中的转折点；
Step7：根据转折点确定周期变化点以及红绿灯周期。

在计算出红绿灯总周期之后，用总周期减去绿灯时长 即可计算出红灯时长。

5. 3. 2 模型的求解

利用 matlab 求解方程（3），得到附件三中各路口的红绿总周期如下：

表 8：C1-C5 路口红绿灯总周期

路口	C1	C2	C3	C4	C5	C6
总周期（秒）	88	88	105	105	88	105

根据滑动窗口法所取到的数据画成散点图，C1-C5 五个路口路灯时长的散点图如下所示：

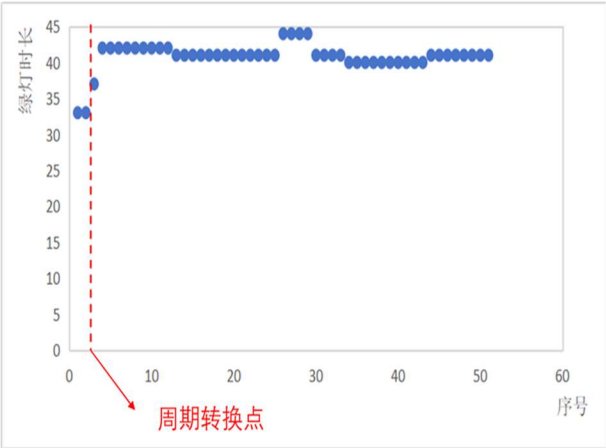

图 8：C1 绿灯时长转折点

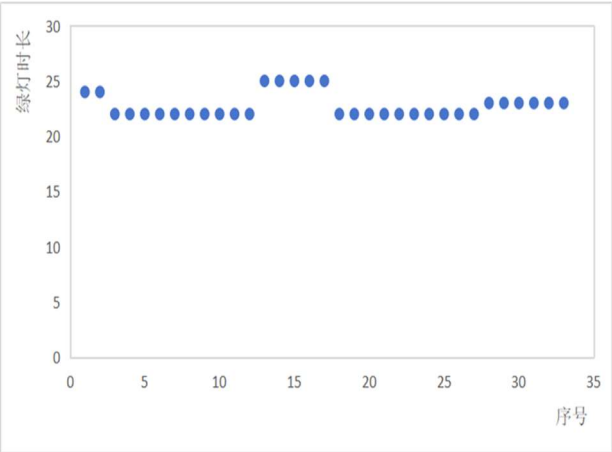

图 9：C2 绿灯时长转折点

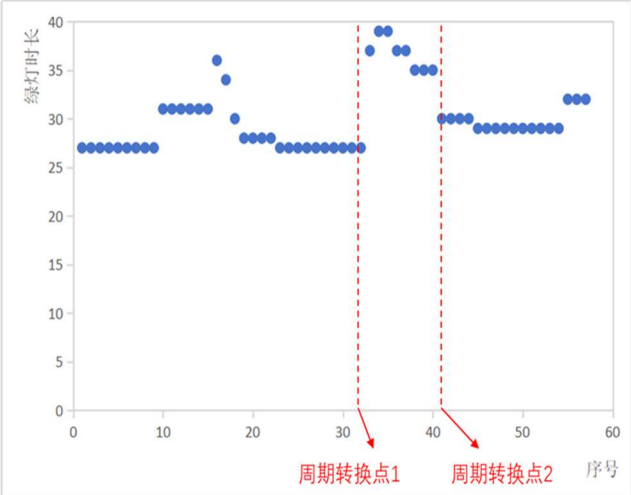

图 10：C3 绿灯时长转折点

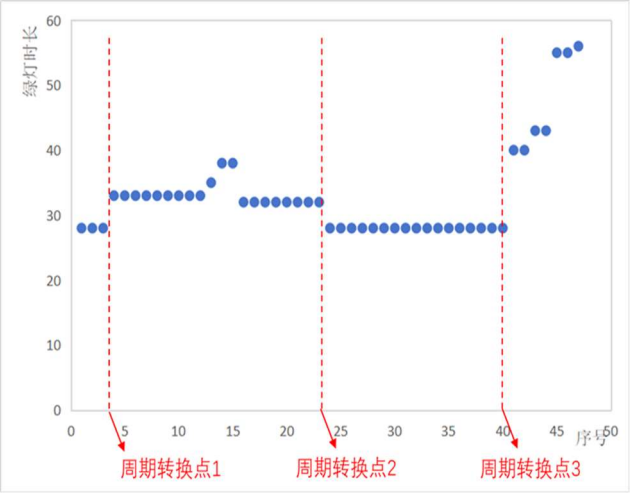

图 11：C4 绿灯时长转折点

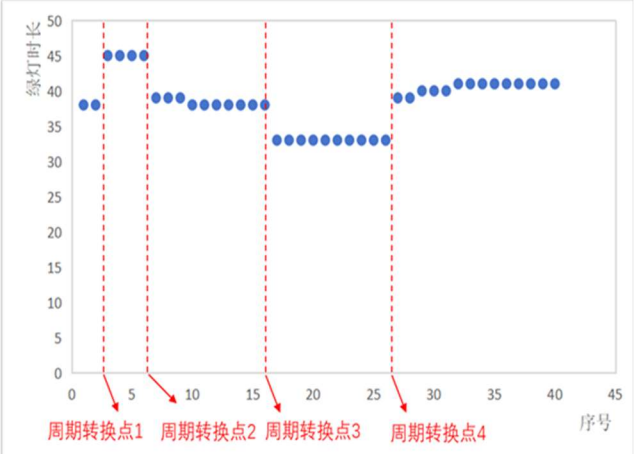

图 12：C5 绿灯时长转折点

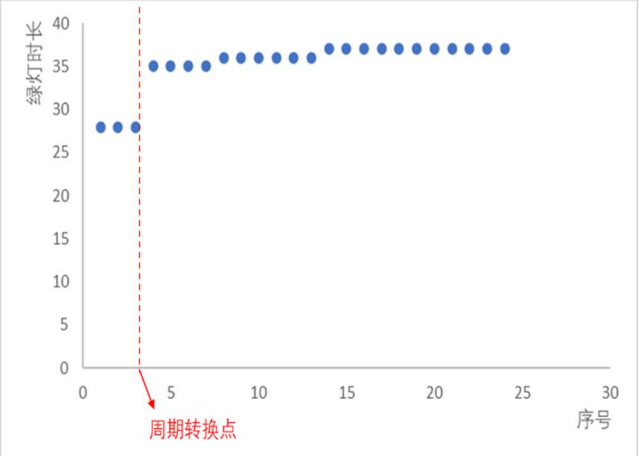

图 13：C6 绿灯时长转折点

这里我们将红绿灯周期转折阈值设为 5s，即当图中绿灯时长突变差距大于等于 5 秒且稳定时，识别为绿灯时长转折点。从文件附件 3 路口绿灯时长.rar 找到对应时刻即为绿灯时长转变时刻。由于红绿灯总周期并未改变，因此绿灯时长转变时刻即为周期切换时刻，具体结果如下表所示：

表 9：C1-C5 路口周期转换时刻

路口	C1	C2	C3	C4	C5	C6
周期切换时刻	1583s	无切换	5854s 6589s	1814s、4755s、 6436s	175s、3343s、 4751s、5191s	944s

表 10：C1-C5 路口信号灯周期

路口	C1	C2	C3	C4	C5	C6
周期 1 红灯时长（秒）	55s	66s	78s	77s	40s	70s
周期 1 绿灯时长（秒）	33s	22s	27s	28s	38s	28s
周期切换时刻	1583s	无	5854s	1814s	175s	944s
周期 2 红灯时长（秒）	48s		70s	72s	43s	56s
周期 2 绿灯时长（秒）	40s		35s	33s	45s	35s
周期切换时刻	无		6589s	4755s	3343s	无
周期 3 红灯时长（秒）			76s	77s	40s	
周期 3 绿灯时长（秒）			29s	28s	38s	
周期切换时刻			无	6436s	4751s	
周期 4 红灯时长（秒）				65s	55s	
周期 4 绿灯时长（秒）				40s	33s	
周期切换时刻				无	5191s	
周期 5 红灯时长（秒）					49s	
周期 5 绿灯时长（秒）					39s	

5.4 问题四模型的建立与求解

5.4.1 模型的建立

由于样本车辆向不同方向行驶，而指示不同方向的信号灯周期往往不同，因此要将样本车辆行驶轨迹数据按照行进方向进行分类。要确定样本车辆的行进方向，先要将车辆的起始位置所处的道路确定下来。根据对样本数据的散点图绘制可以判断该路口是一个十字路口，对应东、西、南、北四条道路。确定起始位置位于哪一条道路可以使用 K-means 聚类方法，将起始位置分为四类，聚类步骤如下：

Step 1：根据设定的簇的数目 4，随机选择 4 个起始位置坐标作为初始的簇中心;

Step 2：对于其他所有的起始位置坐标，计算它们与这 4 个初始簇中心的距离，然后将它们归类到与之最近的簇中心所属的簇；

Step 3：对于每个簇,计算其中所有位置坐标的均值，并将这个均值作为新的簇中心；

Step 4：这个过程会不断重复，直到簇中心不再发生变化。

分类之后根据不同轨迹数据中坐标变化求得车辆的方向向量，进而根据方向向量确定位于不同道路的车辆行驶方向，由于相同道路上车辆经过红绿灯时可能会出现直行、左转和右转三种情况，但现实生活中右转车辆不需要看交通信号灯，对于确定交通信号灯的周期并没有帮助，因此本文自动剔除车辆右转的情况。根据车辆行驶方向和经过红绿灯时直行还是左转两个标准将所有车辆划分为 8 种情形，分别为自南向北直行、自南向北左转、自北向南直行、自北向南左转、自东向西直行、自东向西左转、自西向东直行、自西向东左转。实际情况下，相向行驶的车辆经过路口时看到的是同一个红绿灯的指示。因此对于相向行驶的车辆数据合并在一起利用模型一和模型二的方法求出其红绿灯时长。考虑到该路口的信号灯周期可能会发生变化，再次使用模型三中的滑动窗口法判断信号灯周期变化的时刻以及变化后红绿灯时长分别是多少。

5.4.2 模型的求解

K-means 聚类之后，将样本车辆的起始位置分为四类，结果如下图所示：

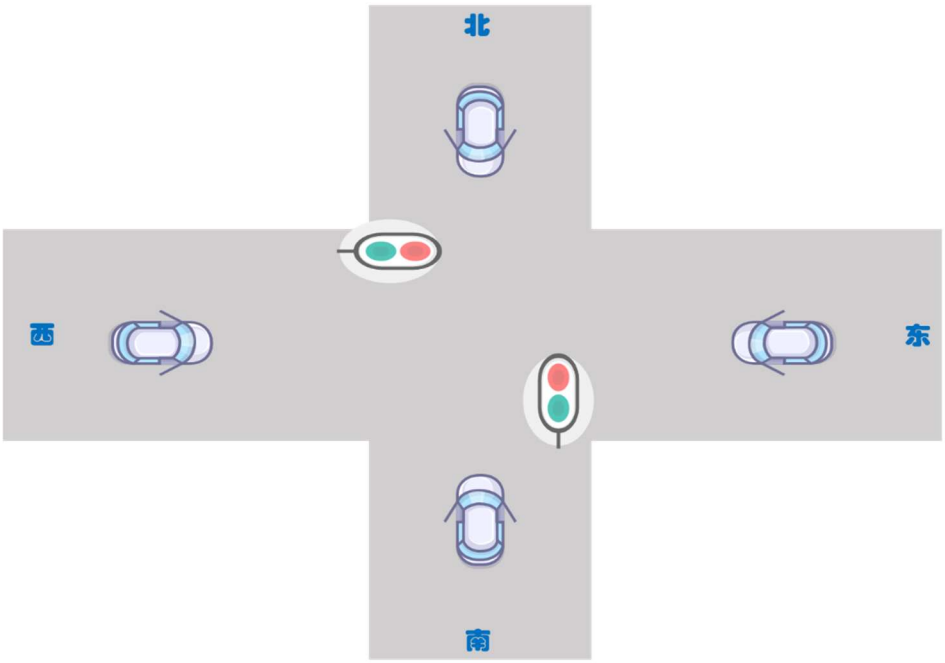

图 14：车辆起始位置示意图

将车辆进行分类之后再次合并为四种情况，分别为开始行驶方向为南北方向，之后左转和直行、开始行驶方向为东西方向，之后左转和直行。红灯时长依旧按照之前的方法画出等待时间的均匀分布图找最大值或者图像拐点。

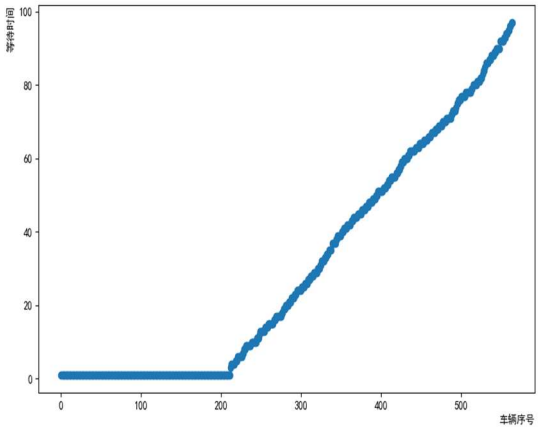

图 15：南北直行等待时间均匀分布图

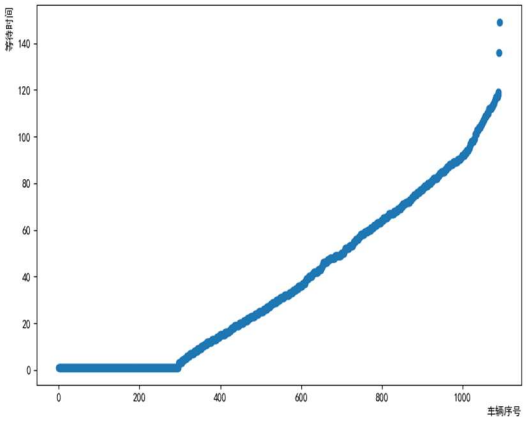

图 16：南北左转等待时间均匀分布图

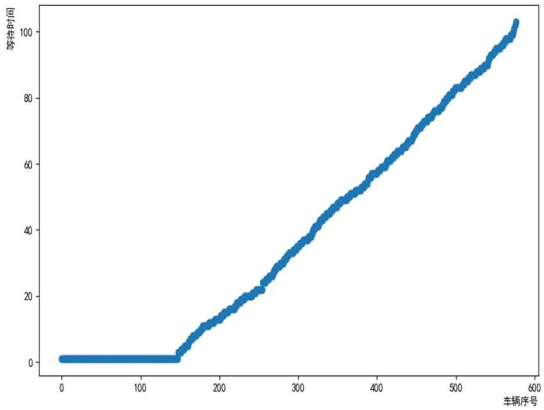

图 17：东西直行等待时间均匀分布图

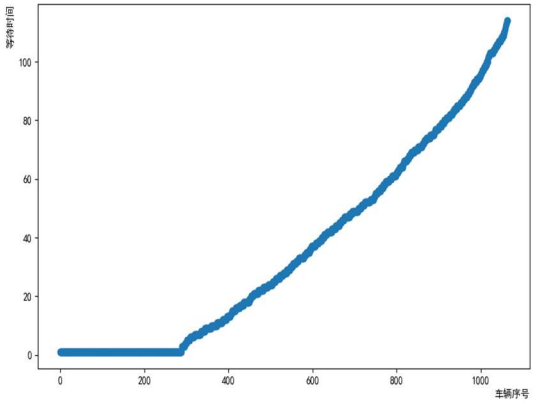

图 18：东西左转等待时间均匀分布图

依据均匀分布图上的最大值或者图像拐点可确定红灯时长，而依据所识别的绿灯行驶时间的最小值即可确定绿灯时长，进而确定出该路口不同方向信号灯的周期。同时，经过滑动窗口法判断，该路口在连续两个小时内信号灯周期的确发生了变化，切换时刻以及切换前后的红绿灯时长计算结果如下表所示：

表 11：附件 4 路口信号灯周期

	南北向直行	南北向左转	东西向直行	东西向左转
红灯时长 1（秒）	97	119	103	114
绿灯时长 1（秒）	46	25	40	29
周期切换时刻	3407s	3452s	4611s	4653s
红灯时长 2（秒）	91	112	98	107
绿灯时长 2（秒）	51	30	44	35

六、 模型的评价、改进与推广

6.1 模型的优点

1. 本报告中计算红灯时长和绿灯时长的模型耦合性很小，采用不同的方法分别计算红灯时长和绿灯时长，二者相互关联和影响的程度较低，较不容易出现

联合性错误；

2. 采用滑动窗口法来识别周期变化时刻的一个主要优点是它只需要在窗口内的元素上进行操作，而不是在整个数据序列上进行，因此可以提高效率。总的来说，滑动窗口法是一种非常灵活且有效的算法，广泛应用于各种需要处理子序列或子数组的问题中。
3. 模型稳定性较高，经过稳定性分析和误差分析后，发现当车辆数量较少时，依旧可以保证周期估计的准确度

6.2 模型的缺点

1. 问题四的模型中，对于单一方向的红绿灯周期计算仅考虑当前方向的车辆行车轨迹的影响，但没有考虑其余方向车辆的行车轨迹可能对计算当前方向红绿灯周期产生的影响。
2. 本文所用模型没有根据总周期时长对红绿灯时长进行调整，导致红绿灯时长之和与总周期时长具有偏差。

6.3 模型的推广

本文利用公式和均匀分布的思想分别计算了红灯和绿灯时长，且模型耦合性较低，能够保证较高的估计精度。并且适用于较大范围的车流量，当车流量较低时，也可以保证估计精度。同时在分析样本车辆比例、车流量、定位误差等不同因素对于信号灯周期估计精度的影响时，采用了定量分析与定性分析相结合的方法，多方面综合分析，结果具有较高的可靠性。为减弱定位误差对估计结果的影响，采用了二次贝塞尔曲线对车辆轨迹进行平滑拟合。其作为一种数学曲线，常用于计算机图形和数据可视化领域。它的特点是平滑、连续，能够推广于更多需要去除噪声的问题研究。

七、 参考文献

[1]https://pss-system.cponline.cnipa.gov.cn/documents/detail?prevPageTit=changgui （专利网）
[2] 陈凯,王春,代文,等.基于控制点蒙特卡罗检验的无人机地形建模精度影响因素研究[J].自然资源遥感,2023,35(03):107-115.
[3] 张云辉,谭庆昌,田原嫄.图像测量系统精度影响因素的研究[J].微计算机信息,2008,(24):271-273.
[4] 袁方.基于伪距差分 GPS 的行车轨迹线形拟合算法[J].中国水运(下半月),2015,15(09):143-144.

附录

附录 1
介绍：支撑材料的文件列表
信号灯识别结果.docx：利用附件 1、2、3 中数据求出的不同路口信号灯周期结果
附件 1 车辆速度数据.rar：利用附件 1 中数据求出的车辆在不同时刻的速度
A1-A5 等待时间均匀分布图.rar：根据所求速度来判断的 A1-A5 路口车辆等待时间排序之后的均匀分布图
A1-A5 车辆等待时间.rar：根据所求速度来判断的 A1-A5 路口车辆等待时间
附件 1 路口绿灯时长.rar：根据附件 1 数据求得的路口绿灯时长及车辆启动时刻与停止时刻
样本车辆比例造成的红灯周期误差.rar：A1-A5 五个路口分别调整不同的样本车辆比例，得到新的红灯周期与问题一求得的红灯周期作比较，利用公式求得的估计误差
样本车辆比例造成的绿灯周期误差.rar：A1-A5 五个路口分别调整不同的样本车辆比例，得到新的红灯周期与问题一求得的绿灯周期作比较，利用公式求得的估计误差
附件 3 路口绿灯时长.rar：C1-C6 六个路口在经过二次贝塞尔曲线平滑之后求得的绿灯时长及车辆启动时刻与停止时刻

附录 2
介绍：该代码是 Python 语言编写的，作用是对原始数据进行清洗，查找是否有空值

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

# 通过修改读取文件的名称进行数据清洗
fileName = 'D'
# 读取 csv 表格里面的数据
df = pd.read_csv(fileName + '.csv')
# print(df)
# df.info()
# #查看这个 dataframe 的信息
isNull = df.isnull().sum()
# 判断 df 中是否有空值
print(isNull)
# 有无空值
# 有无重复值
isRepeat = df.duplicated().sum()
print(isRepeat)

# 计算速度值
# 根据欧氏公式计算距离进而算出速度，通过差分得出
df['speed'] = (np.sqrt((df['x'] -
df.groupby('vehicle_id')['x'].shift()) ** 2 +
                       (df['y'] -
df.groupby('vehicle_id')['y'].shift()) ** 2) / (
                      df['time'] -
df.groupby('vehicle_id')['time'].shift()))
```

```
# speed_sign = np.where((df['x'] -
df.groupby('vehicle_id')['x'].shift()) > 0, -1, 1)
# df['speed'] = df['speed'] * speed_sign

# 填补每一个 id 对应的第一个速度值，因为没有办法差分得到上一个速度
missing_rows = df['speed'].isnull()
# 使用 loc 来定义上述找到的 missing_rows 然后填入经过 vehicle_id 进行分组后的下一个
speed 的
df.loc[missing_rows, 'speed'] =
df.groupby('vehicle_id')['speed'].shift(-1)
# 输出加入速度的 csv 表格到表格命名的 csv 表格
df.to_csv(fileName + 'WithSpeed.csv', index=False)

# 根据车辆的 id 进行分类获取分好组的字典得到可以查询的速度
# groupedTime = df.groupby('vehicle_id')['time'].apply(list).to_dict()
# groupedX = df.groupby('vehicle_id')['x'].apply(list).to_dict()
# groupedY = df.groupby('vehicle_id')['y'].apply(list).to_dict()
groupedSpeed = df.groupby('vehicle_id')['speed'].apply(list).to_dict()
# print(groupedX[8][19])

# 获取存在的车辆 ID 列表，注意是 unique 的，独一无二的
existing_vehicle_ids = df['vehicle_id'].unique()
print(existing_vehicle_ids)
# 根据存在的车辆 ID 过滤分组数据，不存在的 id 就抛弃掉
filtered_groupedSpeed = {vehicle_id: groupedSpeed[vehicle_id] for
vehicle_id in existing_vehicle_ids}
# 输出过滤好的分组速度，得到想要的
print(filtered_groupedSpeed)
# 绘制箱线图
plt.figure(figsize=(100, 6))  # 创建一个图形对象，并设置尺寸为 (100, 6)，单位
是英寸
sns.boxplot(data=df, x='vehicle_id', y='speed')  # 绘制箱线图，x 轴为
'vehicle_id' 列，y 轴为 'speed' 列的数据
plt.title('Boxplot of X Data by Vehicle ID')  # 设置图标题
plt.xlabel('Time')  # 设置 x 轴标签
plt.ylabel('speed Data')  # 设置 y 轴标签
plt.show()  # 显示图形
```

附录 3

介绍：该代码是 Python 语言编写的，作用是计算车辆等待时间，并且绘制车辆等待时间升序图，寻找合理的红灯时间

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

# 设置字体为中文字体
plt.rcParams['font.family'] = 'SimHei'  # 设置字体为宋体或 SimHei
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号


# 定义一个函数来判断 i 后面是否有 0，如果有的话，它就会知道当前的 speed 值是异常值
def whether0behind(speed, i, length):
    count1 = 10
    m = 1
```

```
    while m <= count1 and i + m < length:
        if speed[i + m] <= 1.39:  # 1.39 是通过网上得到的车辆停止的 speed 阈值
          return True
        m += 1
    return False


# 修改文件名，以获取特定数据集的结果
file = 'B5'

# 读取带有速度数据的 CSV 文件
filename = file + 'WithSpeed.csv'
df = pd.read_csv(filename)

# 创建一个新列，存储上一行的速度
df['previous_speed'] = df.groupby('vehicle_id')['speed'].shift(1)

# 填补每一个 id 对应的第一个速度值，因为没有办法差分得到上一个速度
missing_rows = df['previous_speed'].isnull()
# 使用 loc 来定义上述找到的 missing_rows 然后填入经过 vehicle_id 进行分组后的上一个
speed 的
df.loc[missing_rows, 'previous_speed'] =
df.groupby('vehicle_id')['previous_speed'].shift(-1)

df['future_speed'] = df.groupby('vehicle_id')['speed'].shift(-1)
# 填补每一个 id 对应的第一个速度值，因为没有办法差分得到上一个速度
missing_rows = df['future_speed'].isnull()
# 使用 loc 来定义上述找到的 missing_rows 然后填入经过 vehicle_id 进行分组后的上一个
speed 的
df.loc[missing_rows, 'future_speed'] =
df.groupby('vehicle_id')['future_speed'].shift(1)
# print(df)


df['future_future_speed'] =
df.groupby('vehicle_id')['future_speed'].shift(-1)
missing_rows = df['future_future_speed'].isnull()
df.loc[missing_rows, 'future_future_speed'] =
df.groupby('vehicle_id')['future_future_speed'].shift(1)

df.to_csv(file+'future.csv')
# 将速度数据按车辆 ID 分组并转换为字典，并得到速度
groupedSpeed = df.groupby('vehicle_id')['speed'].apply(list).to_dict()

# 获取数据集中存在的车辆 ID，最后能够得到数据里面单独的独一无二的 ID
existing_vehicle_ids = df['vehicle_id'].unique()

# 初始化等待时间字典
waitingTime = {}

# 遍历数据集中的每个车辆 ID，来计算等待的时间
for vehicle_id in existing_vehicle_ids:
    count = 0  # 初始化等待计数器
    i = 0  # 初始化速度列表索引
    # 遍历该车辆的速度列表
    if vehicle_id == 43:
        print('nih ')
    while i < len(groupedSpeed[vehicle_id]):
```

```
        # 如果当前速度小于等于 1.39，认为车辆在等待

       while (i+2 < len(groupedSpeed[vehicle_id]) and
groupedSpeed[vehicle_id][i] <= 1.39
                and groupedSpeed[vehicle_id][i+1]<=1.39 and
groupedSpeed[vehicle_id][i+2] <= 1.39):
            if count==35:
                print('66')
            count += 1  # 等待计数器加 1
            i += 1  # 更新速度列表索引
       # 如果连续等待超过 1 次并且之后还有速度大于 1.39，则继续计数
       if count >= 1 and whether0behind(groupedSpeed[vehicle_id], i,
len(groupedSpeed[vehicle_id])):
            count += 1  # 等待计数器加 1
        i += 1  # 更新速度列表索引
    # 如果等待时间大于 0，则记录到等待时间字典中
    # if count > 0:
     count+=1
     waitingTime[vehicle_id] = count

# waitingTime = {k: v for k, v in waitingTime.items() if v != 0}
# 输出等待时间字典
print(waitingTime)

# 创建 DataFrame，存储车辆等待时间数据
waitingTime_df = pd.DataFrame(waitingTime.items(),
columns=['vehicle_id', 'waiting_time'])

# 输出到 CSV 文件
waitingTime_df.to_csv(file + 'waiting_time.csv', index=False)

# 按照等待时间升序排列 DataFrame
waiting_time_df_sorted = waitingTime_df.sort_values(by='waiting_time')

# 重置索引，命名为车辆序号
waiting_time_df_sorted.reset_index(drop=True, inplace=True)

# 输出排序后的 DataFrame
print(waiting_time_df_sorted)

# 绘制散点图
plt.figure(figsize=(10, 6))  # 设置图形大小再 10,6
plt.scatter(waiting_time_df_sorted.index,
waiting_time_df_sorted['waiting_time'])  # 绘制散点图
plt.title(file)  # 设置图表标题为 A 几 2
plt.xlabel('车辆序号', loc='right')  # 设置 x 轴标签，并指定位置为右侧
plt.ylabel('等待时间', loc='top')  # 设置 y 轴标签，并指定位置为顶部
plt.grid(False)  # 关闭网格线
plt.savefig(file + '.png')  # 保存散点图为文件
plt.show()  # 显示图形
```

附录 3

介绍：该代码是 Python 语言编写的，作用是计算车辆等待时间，并且绘制车辆等待时间升序图，寻找合理的红灯时间

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

```
import seaborn as sns
import matplotlib.pyplot as plt

# 设置字体为中文字体
plt.rcParams['font.family'] = 'SimHei'  # 设置字体为宋体或 SimHei
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号


# 定义一个函数来判断 i 后面是否有 0，如果有的话，它就会知道当前的 speed 值是异常值
def whether0behind(speed, i, length):
    count1 = 10
    m = 1
    while m <= count1 and i + m < length:
        if speed[i + m] <= 1.39:  # 1.39 是通过网上得到的车辆停止的 speed 阈值
          return True
        m += 1
    return False


# 修改文件名，以获取特定数据集的结果
file = 'B5'

# 读取带有速度数据的 CSV 文件
filename = file + 'WithSpeed.csv'
df = pd.read_csv(filename)

# 创建一个新列，存储上一行的速度
df['previous_speed'] = df.groupby('vehicle_id')['speed'].shift(1)

# 填补每一个 id 对应的第一个速度值，因为没有办法差分得到上一个速度
missing_rows = df['previous_speed'].isnull()
# 使用 loc 来定义上述找到的 missing_rows 然后填入经过 vehicle_id 进行分组后的上一个
speed 的
df.loc[missing_rows, 'previous_speed'] =
df.groupby('vehicle_id')['previous_speed'].shift(-1)

df['future_speed'] = df.groupby('vehicle_id')['speed'].shift(-1)
# 填补每一个 id 对应的第一个速度值，因为没有办法差分得到上一个速度
missing_rows = df['future_speed'].isnull()
# 使用 loc 来定义上述找到的 missing_rows 然后填入经过 vehicle_id 进行分组后的上一个
speed 的
df.loc[missing_rows, 'future_speed'] =
df.groupby('vehicle_id')['future_speed'].shift(1)
# print(df)


df['future_future_speed'] =
df.groupby('vehicle_id')['future_speed'].shift(-1)
missing_rows = df['future_future_speed'].isnull()
df.loc[missing_rows, 'future_future_speed'] =
df.groupby('vehicle_id')['future_future_speed'].shift(1)

df.to_csv(file+'future.csv')
# 将速度数据按车辆 ID 分组并转换为字典，并得到速度
groupedSpeed = df.groupby('vehicle_id')['speed'].apply(list).to_dict()

# 获取数据集中存在的车辆 ID，最后能够得到数据里面单独的独一无二的 ID
existing_vehicle_ids = df['vehicle_id'].unique()
```

```
# 初始化等待时间字典
waitingTime = {}

# 遍历数据集中的每个车辆 ID，来计算等待的时间
for vehicle_id in existing_vehicle_ids:
    count = 0  # 初始化等待计数器
    i = 0  # 初始化速度列表索引
    # 遍历该车辆的速度列表
    if vehicle_id == 43:
        print('nih ')
    while i < len(groupedSpeed[vehicle_id]):
        # 如果当前速度小于等于 1.39，认为车辆在等待

        while (i+2 < len(groupedSpeed[vehicle_id]) and
groupedSpeed[vehicle_id][i] <= 1.39
               and groupedSpeed[vehicle_id][i+1]<=1.39 and
groupedSpeed[vehicle_id][i+2] <= 1.39):
            if count==35:
                print('66')
            count += 1  # 等待计数器加 1
            i += 1  # 更新速度列表索引
        # 如果连续等待超过 1 次并且之后还有速度大于 1.39，则继续计数
        if count >= 1 and whether0behind(groupedSpeed[vehicle_id], i,
len(groupedSpeed[vehicle_id])):
            count += 1  # 等待计数器加 1
        i += 1  # 更新速度列表索引
    # 如果等待时间大于 0，则记录到等待时间字典中
    # if count > 0:
    count+=1
    waitingTime[vehicle_id] = count

# waitingTime = {k: v for k, v in waitingTime.items() if v != 0}
# 输出等待时间字典
print(waitingTime)

# 创建 DataFrame，存储车辆等待时间数据
waitingTime_df = pd.DataFrame(waitingTime.items(),
columns=['vehicle_id', 'waiting_time'])

# 输出到 CSV 文件
waitingTime_df.to_csv(file + 'waiting_time.csv', index=False)

# 按照等待时间升序排列 DataFrame
waiting_time_df_sorted = waitingTime_df.sort_values(by='waiting_time')

# 重置索引，命名为车辆序号
waiting_time_df_sorted.reset_index(drop=True, inplace=True)

# 输出排序后的 DataFrame
print(waiting_time_df_sorted)

# 绘制散点图
plt.figure(figsize=(10, 6))  # 设置图形大小再 10,6
plt.scatter(waiting_time_df_sorted.index,
waiting_time_df_sorted['waiting_time'])  # 绘制散点图
plt.title(file)  # 设置图表标题为 A 几 2
```

```
plt.xlabel('车辆序号', loc='right')  # 设置 x 轴标签，并指定位置为右侧
plt.ylabel('等待时间', loc='top')  # 设置 y 轴标签，并指定位置为顶部
plt.grid(False)  # 关闭网格线
plt.savefig(file + '.png')  # 保存散点图为文件
plt.show()  # 显示图形
```

附录 4

介绍：该代码是 Python 语言编写的，作用是计算第一题中等待红灯之后起步到有车停在这个路口的间隔时间，获得间隔时间序列，进而得出绿灯时间

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

# 修改文件名，以获取特定数据集的结果
file = 'A1'

# 读取带有速度数据的 CSV 文件
filename = file + 'WithSpeed.csv'
df = pd.read_csv(filename)
# 筛选出距离路口欧氏距离在 15m 以内的车，即路口的车
condition = np.sqrt(df['x'] ** 2 + df['y'] ** 2) <= 15  # 设定筛选条件
df_filtered = df[condition]  # 使用布尔索引筛选满足条件的行

# 重置筛选出来的 df 的索引
df_filtered.reset_index(drop=True, inplace=True)
# 创建一个新列，存储上一行的速度
df_filtered['previous_speed'] =
df_filtered.groupby('vehicle_id')['speed'].shift(1)

# 填补每一个 id 对应的第一个速度值，因为没有办法差分得到上一个速度
missing_rows = df_filtered['previous_speed'].isnull()
# 使用 loc 来定义上述找到的 missing_rows 然后填入经过 vehicle_id 进行分组后的上一个 speed 的
df_filtered.loc[missing_rows, 'previous_speed'] =
df_filtered.groupby('vehicle_id')['previous_speed'].shift(-1)

df_filtered['future_speed'] =
df_filtered.groupby('vehicle_id')['speed'].shift(-1)
# 填补每一个 id 对应的第一个速度值，因为没有办法差分得到上一个速度
missing_rows = df_filtered['future_speed'].isnull()
# 使用 loc 来定义上述找到的 missing_rows 然后填入经过 vehicle_id 进行分组后的上一个 speed 的
df_filtered.loc[missing_rows, 'future_speed'] =
df_filtered.groupby('vehicle_id')['future_speed'].shift(1)
# print(df_filtered)


df_filtered['future_future_speed'] =
df_filtered.groupby('vehicle_id')['future_speed'].shift(-1)
missing_rows = df_filtered['future_future_speed'].isnull()
df_filtered.loc[missing_rows, 'future_future_speed'] =
df_filtered.groupby('vehicle_id')['future_future_speed'].shift(1)

df_filtered.to_csv(file + "Filtered.csv", index=False)

i = 0
```

```
greenTime = []
startTime = []
startID = []
endTime = []
endID = []
while i < len(df_filtered):
    # 获取当前行的 vehicle_id
    # current_vehicle_id = df_filtered.loc[i, 'vehicle_id']
    # 获取当前行的 x 坐标对应的上一行的值
   previousSpeed = df_filtered.loc[i, 'previous_speed']
    # futureSpeed = df_filtered.loc[i, 'future_speed']
    # 如果当前行的速度小于等于 1.39 并且上一行的 x 坐标为 0，则输出当前行的时间
   if df_filtered.loc[i, 'speed'] < 1.39 <= df_filtered.loc[i,
'future_speed'] and df_filtered.loc[
        i, 'future_future_speed'] >= 1.39:
        start_time = df_filtered.loc[i, 'time']
        startTime.append(start_time)
        startID.append(df_filtered.loc[i, 'vehicle_id'])

        print('start_time')
        print(start_time)
        while not (df_filtered.loc[i, 'speed'] >= 1.39 >
df_filtered.loc[i, 'future_speed'] and df_filtered.loc[
            i, 'future_future_speed'] < 1.39):
            if i < len(df_filtered) - 1:
                i += 1
            else:
                break
        if i < len(df_filtered) - 1:
            i += 1
        end_time = df_filtered.loc[i, 'time']
        print('end_time')
        print(end_time)
        time_diff = end_time - start_time
        greenTime.append(time_diff)
        endTime.append(end_time)
        endID.append(df_filtered.loc[i, 'vehicle_id'])
        print(len(greenTime))
    i += 1
# print(greenTime)
greenTime.pop()
endTime.pop()
startTime.pop()
startID.pop()
endID.pop()

result_dataframe = pd.DataFrame({
    'greenTime': greenTime,
    'startTime': startTime,
    'startID': startID,
    'endTime': endTime,
    'endID': endID
})

result_dataframe.to_csv(file+'_greenTime.csv', index=False)

# 对 DataFrame 按照 'greenTime' 列升序排列
result_dataframe = result_dataframe.sort_values(by='greenTime')

# 重置索引
```

```
result_dataframe = result_dataframe.reset_index(drop=True)

print(result_dataframe)

# print(greenTime)
# greenTime.sort()
# print(greenTime)
```

附录 5

介绍：该代码是 Python 语言编写的，作用是计算第二问中样本车辆比例对于红灯时间估计的影响，使用蒙特卡洛算法进行模拟抽取不同比例的样本

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

# 修改文件名，以获取特定数据集的结果
file = 'A5'

# 读取带有速度数据的 CSV 文件
filename = file + 'waiting_time.csv'
df = pd.read_csv(filename)

print(df)
originalMAX = df['waiting_time'].max()
# 设定模拟次数
num_simulations = 1000
max_waiting_times = []

f = 0.1
exampleRatio = []
waitingTime = []
bias = []
while f <= 0.9:
    # 执行蒙特卡洛模拟
    for _ in range(num_simulations):
        # 从 DataFrame 中随机抽取 10% 的数据
        sample_df = df.sample(frac=f)
        # print(sample_df)
        # 计算抽取数据的 waiting_time 列的最大值
        max_waiting_time = sample_df['waiting_time'].max()
        max_waiting_times.append(max_waiting_time) \
            # 计算所有最大值的均值
    exampleRatio.append(f)
    average_max_waiting_time = np.mean(max_waiting_times)
    waitingTime.append(average_max_waiting_time)
    bias.append(abs(originalMAX - average_max_waiting_time) /
originalMAX)
    print("Average Max Waiting Time:", average_max_waiting_time)
    f += 0.1
resultDF = pd.DataFrame({
    'exampleRatio': exampleRatio,
    'originalMAX': originalMAX,
    'averageMaxWaitingTime': waitingTime,
    'bias': bias
})
```

```
resultDF.to_csv(file + 'Bias.csv', index=False)
```

附录 6

介绍：该代码是 Python 语言编写的，作用是计算第二问中样本车辆比例对于绿灯时间估计的影响，使用蒙特卡洛算法进行模拟抽取不同比例的样本

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


def getGreenTime(df):
    # 筛选出距离路口欧氏距离在 15m 以内的车，即路口的车
    condition = np.sqrt(df['x'] ** 2 + df['y'] ** 2) <= 15  # 设定筛选条件
    df_filtered = df[condition]  # 使用布尔索引筛选满足条件的行
    # print(df_filtered)
    # 重置筛选出来的 df 的索引
    df_filtered.reset_index(drop=True, inplace=True)
    # print(df_filtered)

    # 创建一个新列，存储上一行的速度
    df_filtered['previous_speed'] =
df_filtered.groupby('vehicle_id')['speed'].shift(1)

    # 填补每一个 id 对应的第一个速度值，因为没有办法差分得到上一个速度
    missing_rows = df_filtered['previous_speed'].isnull()
    # 使用 loc 来定义上述找到的 missing_rows 然后填入经过 vehicle_id 进行分组后的
上一个 speed 的
    df_filtered.loc[missing_rows, 'previous_speed'] =
df_filtered.groupby('vehicle_id')['previous_speed'].shift(-1)

    df_filtered['future_speed'] =
df_filtered.groupby('vehicle_id')['speed'].shift(-1)
    # 填补每一个 id 对应的第一个速度值，因为没有办法差分得到上一个速度
    missing_rows = df_filtered['future_speed'].isnull()
    # 使用 loc 来定义上述找到的 missing_rows 然后填入经过 vehicle_id 进行分组后的
上一个 speed 的
    df_filtered.loc[missing_rows, 'future_speed'] =
df_filtered.groupby('vehicle_id')['future_speed'].shift(1)
    # print(df_filtered)

    df_filtered['future_future_speed'] =
df_filtered.groupby('vehicle_id')['future_speed'].shift(-1)
    missing_rows = df_filtered['future_future_speed'].isnull()
    df_filtered.loc[missing_rows, 'future_future_speed'] =
df_filtered.groupby('vehicle_id')[
        'future_future_speed'].shift(1)
    print(df_filtered)
    # 输出！！！！！！！
    # df_filtered.to_csv(file + "Filtered.csv", index=False)
```

```
    i = 0
    greenTime = []
    startTime = []
    startID = []
    endTime = []
    endID = []
    while i < len(df_filtered):
        # 获取当前行的 vehicle_id
        # current_vehicle_id = df_filtered.loc[i, 'vehicle_id']
        # 获取当前行的 x 坐标对应的上一行的值
       previousSpeed = df_filtered.loc[i, 'previous_speed']
        # futureSpeed = df_filtered.loc[i, 'future_speed']
        # 如果当前行的速度小于等于 1.39 并且上一行的 x 坐标为 0，则输出当前行的时
间
       if df_filtered.loc[i, 'speed'] < 1.39 <= df_filtered.loc[i,
'future_speed'] and df_filtered.loc[
            i, 'future_future_speed'] >= 1.39:
            start_time = df_filtered.loc[i, 'time']
            startTime.append(start_time)
            startID.append(df_filtered.loc[i, 'vehicle_id'])

            # print('start_time')
            # print(start_time)
            while not (df_filtered.loc[i, 'speed'] >= 1.39 >
df_filtered.loc[i, 'future_speed'] and df_filtered.loc[
                i, 'future_future_speed'] < 1.39):
                if i < len(df_filtered) - 1:
                    i += 1
                else:
                    break
            if i < len(df_filtered) - 1:
                i += 1
            end_time = df_filtered.loc[i, 'time']
            # print('end_time')
            # print(end_time)
            time_diff = end_time - start_time
            greenTime.append(time_diff)
            endTime.append(end_time)
            endID.append(df_filtered.loc[i, 'vehicle_id'])
            # print(len(greenTime))
        i += 1
    # print(greenTime)

    greenTime.pop()
    endTime.pop()
    startTime.pop()
    startID.pop()
    endID.pop()
    minGreenTime = min(greenTime)
    return minGreenTime

for i in range(2,6):
    # 修改文件名，以获取特定数据集的结果
    file = 'A'+str(i)
    print(file)
    # 读取带有速度数据的 CSV 文件
    filename = file + 'WithSpeed.csv'
    df = pd.read_csv(filename)
    greenTime_Original = getGreenTime(df)
```

```
    num_simulations = 1000
    f = 0.5
    exampleRatio = []
    greenTimeList = []
    greenTimeBias = []
    while f <= 0.9:
        # 执行蒙特卡洛模拟
       mingreenTimeList = []
        for _ in range(num_simulations):
            # 从 DataFrame 中随机抽取 10% 的数据
          AllVehicleID=df['vehicle_id'].unique()
            sample_vehicle_id=pd.Series(AllVehicleID).sample(frac=f)

            sample_df = df[df['vehicle_id'].isin(sample_vehicle_id)]
            sample_df.reset_index(drop=True, inplace=True)
            print(sample_df)
            min_greenTime = getGreenTime(sample_df)
            mingreenTimeList.append(min_greenTime)
        exampleRatio.append(f)
        greenTimeList.append(np.mean(mingreenTimeList))
        greenTimeBias.append(abs(greenTime_Original-
np.mean(mingreenTimeList))/greenTime_Original)
        f += 0.1
    print(greenTimeList)
    result_df = pd.DataFrame({
        'exampleRatio': exampleRatio,
        'greenTime_Original': greenTime_Original,
        'greenTime': greenTimeList,
        'greenTimeBias': greenTimeBias
    })
    result_df.to_csv(file+'_greenTimeBias.csv',index=False)
```

附录 7

介绍：该代码是 Python 语言编写的，作用是得到每分钟车流量图

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

# 修改文件名，以获取特定数据集的结果
file = 'A4'

# 读取带有速度数据的 CSV 文件
filename = file + 'WithSpeed.csv'
df = pd.read_csv(filename)

# print(df)
# 使用 groupby 和 nunique 方法获取 column1 值相同但 column2 有不同值的情况
result = df.groupby('time')['vehicle_id'].nunique()
print(result)
# 将结果转换为 DataFrame
```

```
result_df = result.reset_index(name='unique_vehicle_ids')

result_df.to_csv(file+'flowCar.csv', index=False)
```

附录 8

介绍：该代码是 Python 语言编写的，作用是将 A1-5 的数据进行时间段划分，以获得车流量大和车量小的时间段，并计算这个时间段的车流量

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

# 修改文件名，以获取特定数据集的结果
file = 'A5'

# 读取带有速度数据的 CSV 文件
filename = file + 'WithSpeed.csv'
df = pd.read_csv(filename)
totalTime = df['time'].nunique()
# print(totalTime)
interval = totalTime // 6
i = 0
count = 0
writer = pd.ExcelWriter(file + 'Divided.xlsx')
while i < len(df):
    start_time = df['time'][i]
    end_time = start_time + interval
    indices = df[df.eq(end_time)].stack().index.tolist()
    # print(indices)
    count += 1
    # 对每个值，找到具有最大索引的位置,然后将 i 到max 放到一个新的里面
   if count < 6:
        max_index = max(indices)
        # print(max_index)
        maxi = max_index[0]+1
        # 创建当前时间段的 DataFrame
        current_df = df.iloc[i:maxi]
        print(current_df['vehicle_id'].nunique())
        # 写入当前时间段的 DataFrame 到 Excel 的不同 sheet 中
      current_df.to_excel(writer, sheet_name='Sheet_'+str(count),
index=False)
        i = max_index[0] + 1
    else:
        max_index = len(df) - 1
        # print(max_index)
        maxi = max_index+1
        # 创建当前时间段的 DataFrame
        current_df = df.iloc[i:maxi]
        print(current_df['vehicle_id'].nunique())
```

```
        # 写入当前时间段的 DataFrame 到 Excel 的不同 sheet 中
        current_df.to_excel(writer, sheet_name='Sheet_'+str(count),
index=False)
        i = len(df)

writer.close()
```

附录 9

介绍：该代码是 Python 语言编写的，作用是在先前 A 样本中计算的 speed 数据中加入高斯噪声，然后计算红绿灯周期的改变。

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

plt.rcParams['font.family'] = 'SimHei'  # 设置字体为宋体或 SimHei
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号


def whether0behind(speed, i, length):
    count1 = 10
    m = 1
    while m <= count1 and i + m < length:
        if speed[i + m] <= 1.39:  # 1.39 是通过网上得到的车辆停止的 speed 阈值
          return True
        m += 1
    return False


file = 'A1'
original_red_time = 73
original_green_time = 33
# original_red_time = 73
# 读取带有速度数据的 CSV 文件
filename = file + 'WithSpeed.csv'
df = pd.read_csv(filename)
tt = 0.1
biasList = []
while tt <= 2:
    randomAddition = np.random.normal(0, tt, len(df))  # 这里的 1000 是生
成的样本数量，你可以根据需要进行调整
    # print(randomAddition)
    for i in range(len(df)):
        df.loc[i, 'speed'] += randomAddition[i]
    df['gauss'] = randomAddition
    print(df)
    df.to_csv(file + ' ' + str(tt) + 'WithGauss.csv', index=False)

    # 求红灯！！！！！！！
    # 将速度数据按车辆 ID 分组并转换为字典，并得到速度
    groupedSpeed =
df.groupby('vehicle_id')['speed'].apply(list).to_dict()

    # 获取数据集中存在的车辆 ID，最后能够得到数据里面单独的独一无二的 ID
    existing_vehicle_ids = df['vehicle_id'].unique()
```

```
    # 初始化等待时间字典
    waitingTime = {}

    # 遍历数据集中的每个车辆 ID，来计算等待的时间
    for vehicle_id in existing_vehicle_ids:
         count = 0  # 初始化等待计数器
        i = 0  # 初始化速度列表索引
        # 遍历该车辆的速度列表
        while i < len(groupedSpeed[vehicle_id]):
              # 如果当前速度小于等于 1.39，认为车辆在等待
            while i < len(groupedSpeed[vehicle_id]) and
groupedSpeed[vehicle_id][i] <= 1.39:
                count += 1  # 等待计数器加 1
                i += 1  # 更新速度列表索引
            # 如果连续等待超过 1 次并且之后还有速度大于 1.39，则继续计数
            if count >= 1 and whether0behind(groupedSpeed[vehicle_id], i,
len(groupedSpeed[vehicle_id])):
                count += 1  # 等待计数器加 1
             i += 1  # 更新速度列表索引
        # 如果等待时间大于 0，则记录到等待时间字典中
        # if count > 0:
         waitingTime[vehicle_id] = count

    # waitingTime = {k: v for k, v in waitingTime.items() if v != 0}
    # 输出等待时间字典
    # print(waitingTime)

    # 创建 DataFrame，存储车辆等待时间数据
    waitingTime_df = pd.DataFrame(waitingTime.items(),
columns=['vehicle_id', 'waiting_time'])

    # 输出到 CSV 文件
    waitingTime_df.to_csv(file + 'waiting_time_gauss.csv', index=False)

    # 按照等待时间升序排列 DataFrame
     waiting_time_df_sorted =
waitingTime_df.sort_values(by='waiting_time')

    # 重置索引，命名为车辆序号
    waiting_time_df_sorted.reset_index(drop=True, inplace=True)

    # 输出排序后的 DataFrame
    # print(waiting_time_df_sorted)

    # 绘制散点图
    # plt.figure(figsize=(10, 6))  # 设置图形大小再 10,6
    # plt.scatter(waiting_time_df_sorted.index,
waiting_time_df_sorted['waiting_time'])  # 绘制散点图
    # plt.title(file)  # 设置图表标题为 A 几 2
    # plt.xlabel('车辆序号', loc='right')  # 设置 x 轴标签，并指定位置为右侧
    # plt.ylabel('等待时间', loc='top')  # 设置 y 轴标签，并指定位置为顶部
    # plt.grid(False)  # 关闭网格线
    # plt.savefig(file + ' ' + str(tt) + '.png')  # 保存散点图为文件
    # plt.show()  # 显示图形
    bias = abs(original_red_time -
waiting_time_df_sorted['waiting_time'].max()) / original_red_time
```

```
    print(bias)
    biasList.append(bias)
    tt += 0.1
print(biasList)
plt.figure(figsize=(10, 6))  # 设置图形大小再10,6
plt.scatter(np.arange(0.1, 2, 0.1), biasList)  # 绘制散点图
# plt.title(file)  # 设置图表标题为A几2
plt.xlabel('标准差', loc='right')
plt.ylabel('红灯时长估计偏差', loc='top')
plt.show()  # 显示图形


    # # 求绿灯

    # 筛选出距离路口欧氏距离在15m以内的车，即路口的车
    # condition = np.sqrt(df['x'] ** 2 + df['y'] ** 2) <= 15  # 设定筛选条
件
    # df_filtered = df[condition]  # 使用布尔索引筛选满足条件的行
    #
     # # 重置筛选出来的df的索引
    # df_filtered.reset_index(drop=True, inplace=True)
     # # 创建一个新列，存储上一行的速度
    # df_filtered['previous_speed'] =
df_filtered.groupby('vehicle_id')['speed'].shift(1)
     #
     # # 填补每一个id对应的第一个速度值，因为没有办法差分得到上一个速度
    # missing_rows = df_filtered['previous_speed'].isnull()
     # # 使用loc来定义上述找到的missing_rows然后填入经过vehicle_id进行分组后
的上一个speed的
    # df_filtered.loc[missing_rows, 'previous_speed'] =
df_filtered.groupby('vehicle_id')['previous_speed'].shift(-1)
     #
     # df_filtered['future_speed'] =
df_filtered.groupby('vehicle_id')['speed'].shift(-1)
     # # 填补每一个id对应的第一个速度值，因为没有办法差分得到上一个速度
    # missing_rows = df_filtered['future_speed'].isnull()
     # # 使用loc来定义上述找到的missing_rows然后填入经过vehicle_id进行分组后
的上一个speed的
    # df_filtered.loc[missing_rows, 'future_speed'] =
df_filtered.groupby('vehicle_id')['future_speed'].shift(1)
     # # print(df_filtered)
     #
     # df_filtered['future_future_speed'] =
df_filtered.groupby('vehicle_id')['future_speed'].shift(-1)
     # missing_rows = df_filtered['future_future_speed'].isnull()
     # df_filtered.loc[missing_rows, 'future_future_speed'] =
df_filtered.groupby('vehicle_id')[
     #     'future_future_speed'].shift(1)
     #
     # df_filtered.to_csv(file + "TempFiltered.csv", index=False)
     #
     # i = 0
     # greenTime = []
     # startTime = []
     # startID = []
     # endTime = []
     # endID = []
     # while i < len(df_filtered):
```

```
    #     # 获取当前行的 vehicle_id
    #     # current_vehicle_id = df_filtered.loc[i, 'vehicle_id']
    #     # 获取当前行的 x 坐标对应的上一行的值
    #     previousSpeed = df_filtered.loc[i, 'previous_speed']
    #     # futureSpeed = df_filtered.loc[i, 'future_speed']
    #     # 如果当前行的速度小于等于 1.39 并且上一行的 x 坐标为 0，则输出当前行的时间
    #     if df_filtered.loc[i, 'speed'] < 1.39 <= df_filtered.loc[i, 'future_speed'] and df_filtered.loc[
    #         i, 'future_future_speed'] >= 1.39:
    #         start_time = df_filtered.loc[i, 'time']
    #         startTime.append(start_time)
    #         startID.append(df_filtered.loc[i, 'vehicle_id'])
    #
    #         print('start_time')
    #         print(start_time)
    #         while not (df_filtered.loc[i, 'speed'] >= 1.39 > df_filtered.loc[i, 'future_speed'] and df_filtered.loc[
    #             i, 'future_future_speed'] < 1.39):
    #             if i < len(df_filtered) - 1:
    #                 i += 1
    #             else:
    #                 break
    #         if i < len(df_filtered) - 1:
    #             i += 1
    #         end_time = df_filtered.loc[i, 'time']
    #         print('end_time')
    #         print(end_time)
    #         time_diff = end_time - start_time
    #         greenTime.append(time_diff)
    #         endTime.append(end_time)
    #         endID.append(df_filtered.loc[i, 'vehicle_id'])
    #         print(len(greenTime))
    #     i += 1
    # greenTime.pop()
    # endTime.pop()
    # startTime.pop()
    # startID.pop()
    # endID.pop()
    # print(greenTime)
    # result_dataframe = pd.DataFrame({
    #     'greenTime': greenTime,
    #     'startTime': startTime,
    #     'startID': startID,
    #     'endTime': endTime,
    #     'endID': endID
    # })
    #
    # # 对 DataFrame 按照 'greenTime' 列升序排列
    # result_dataframe = result_dataframe.sort_values(by='greenTime')
    #
    # # 重置索引
    # result_dataframe = result_dataframe.reset_index(drop=True)
    #
    # # print(result_dataframe)
    # tem = result_dataframe[result_dataframe['greenTime'] > 10]
    # minGreenTime = tem['greenTime'].min()
    # print('minGreenTime', minGreenTime)
    # biasList.append(abs(original_green_time - minGreenTime) / original_green_time)
```

```
    # # result_dataframe.to_csv(file + '_greenTime.csv', index=False)
    # tt += 0.1
# print(biasList)
# plt.figure(figsize=(10, 6))  # 设置图形大小再 10,6
# plt.scatter(np.arange(0.1, 2, 0.1), biasList)  # 绘制散点图
# # plt.title(file)  # 设置图表标题为 A 几 2
# plt.xlabel('标准差', loc='right')
# plt.ylabel('绿灯时长估计偏差', loc='top')
# plt.show()  # 显示图形
```

附录 10

介绍：该代码是 Python 语言编写的，作用是计算车辆行驶方向所用的夹角

```
    import math


def clockwise_angle_between_vectors(v1, v2):
    # 计算向量的点积
    dot_product = v1[0] * v2[0] + v1[1] * v2[1]

    # 计算向量的叉积
    cross_product = v1[0] * v2[1] - v1[1] * v2[0]

    # 计算夹角的弧度值
    angle_rad = math.atan2(cross_product, dot_product)

    # 转换弧度值为角度值
    angle_deg = math.degrees(angle_rad)

    # 确保角度在 [0, 360) 范围内
    if angle_deg < 0:
        angle_deg += 360

    # 如果想要得到顺时针方向的夹角，可以使用 360 - angle_deg
    clockwise_angle = 360 - angle_deg

    return clockwise_angle


vector1 = (3, 4)
vector2 = (-2, 5)

angle = clockwise_angle_between_vectors(vector1, vector2)
print(f"顺时针方向夹角为 {angle:.2f} 度")
```