基于两阶段 STFT 和贝叶斯变点检测的信号灯周期估计研究

摘　要

本文根据某电子地图服务商提供的行车轨迹数据，建立了基于**两阶段 STFT 的时频分析模型**和**基于 DBSCAN 聚类分析改进的周期检测模型**对不同路口的周期进行估计；然后利用**贝叶斯变点检测**求出周期变化时各阶段的周期参数。

问题一中，本文首先对 A1-A5 五个互不相关路口的行车轨迹数据进行**离群点检测、轨迹连续性检验**，然后进行**异常值处理**。然后，通过指标构建得到车辆运行**加速度状态**的时序数据，本文建立了**基于两阶段 STFT 的时频分析模型**，对时序数据进行周期特征挖掘，得出信号灯的周期性规律。然后通过绘制**功率谱密度图**得到信号灯完整周期长度，再将车辆处于**停车状态最长的时间**作为红灯周期，进而得到绿灯周期。最终得到 A1-A5 路口的**红灯周期**分别为**[73,59,81,69,63]**;**绿灯周期**分别为**[33,28,26,19,26]**。

问题二中，由于样本车辆减少，存在定位偏差等情况，因此，本文在第一问所建立模型的基础上**融入 DBSCAN 聚类分析**，得到红绿灯完整周期时长或者完整周期时长的倍数，然后利用**基于两阶段 STFT 的时频分析模型**对红绿灯周期进行估计，得到 B1-B5 路口的**红灯周期**分别为**[78,83,54,78,95]**，**绿灯周期**分别为**[27,33,34,27,21]**。最后对模型设计**对比实验分析**探究影响因素，发现车流量减小对模型的影响较大，定位误差因素对模型基本无影响，发现模型有较高的鲁棒性。

问题三中，每个路口信号灯周期均可能发生变化，故本文使用**贝叶斯变点检测**对周期的变化点进行检验；然后将时序数据沿变化点进行分段，对分段后的每段时序数据使用**加 Manning 窗的 FFT 算法**求解周期，得到 C1-C6 路口的主要完整周期长度分别为**[88,88,105,105,88,105]**，每个路口周期变化次数分别为**[6,5,3,0,4,0]**。

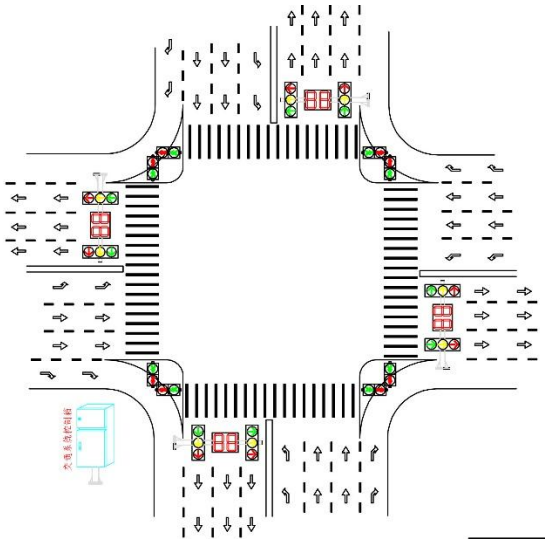
问题四中，附件 4 给出的是某路口 D 所有方向的车辆轨迹数据，本文首先建立**多方向汽车轨迹分解**模型对该路口的车辆行驶**方向**进行**分解**，得到沿一个方向的所有车辆轨迹；再使用前文模型求解各个方向的周期，最后使用滑动窗口稳态检测对周期进行检验，得到路口所以方向完整红绿灯周期时长均为 **142s**，且对向车道信号灯周期时间相同，符合现实红绿灯规律。

关键词：两阶段 STFT 的时频分析　DBSCAN 聚类分析　贝叶斯变点检测

目录

一、 问题重述

交通信号灯的周期长度对于驾驶员来说是一个非常重要的参考信息。合理估计信号灯周期可以帮助驾驶员更好地规划行驶路线和时间，提高行车效率。然而，在实际复杂的道路交通环境中，由于多种因素的影响，信号灯周期长度很难通过简单的观察直接获得。近年来，随着车载传感器、车联网等技术的发展，使用车辆行驶轨迹数据来估算交通信号灯周期长度成为一种可行的新方法。车辆在通过路口时会产生连续的定位数据，通过分析这些数据，可以推算出车辆到达绿灯和红灯的时间点，从而反推出信号灯的工作周期。



某电子地图服务商计划使用客户的行车轨迹数据估计交通信号灯周期以提供更好的导航服务。本文主要解决以下问题：

问题一：若交通信号灯的周期固定不变，且已知各车辆的行车轨迹数据，建立相应的数学模型，利用附件中的轨迹数据来估计出信号灯的红绿周期；

问题二：实际上，公司只能获取到部分样本车辆的行驶轨迹数据。需要讨论车辆比例、车流量、定位误差等因素对前述模型估计精度的影响，根据附件，估算这些路口对应方向的信号灯周期；

问题三：若信号灯周期发生变化，分析附件中的数据，判断这些路口对应方向的信号灯周期在两个小时内是否发生变化。如果发现了变化，请尝试确定周期切换的时刻，以及新旧周期的参数；同时指出周期变化所需的时间和条件；

问题四：根据附件中某路口所有方向样本车辆的轨迹数据给出该路口信号灯的周期。

二、 模型假设

1、假设每个路口在正常情况下,不会发生交通事故、堵车等意外情况,道路通行仅受红绿灯信号的影响；

2、针对附件提供的数据,假设这些数据是真实可靠的,并且车辆在行驶过程中均遵守交通规则,因此可以直接基于这些数据进行计算分析；

3、假设题目中给出的若干路口的红绿灯信号情况是相互独立的,即上一个路口的红绿灯状态不会对下一个路口的红绿灯信号产生任何影响。

三、 符号说明

符号	说明
(x_m, y_m)	车辆当前位置点坐标
(x_0, y_0)	路口中心坐标
$\nu_i(t)$	第i辆车在t时刻的速度
$a_i(t)$	第i辆车在t时刻的加速度
T	完整红绿灯周期时长
T_{red}	红灯周期
T_{green}	绿灯周期
n	t时刻路口内车辆总数

四、 问题一的模型建立与求解

4.1 问题分析

针对问题一，首先需要对车辆行车轨迹数据进行清洗，剔除异常值。这可以通过确实轨迹正方向、离群点检验、轨迹连续性检验等方式来识别异常点，并用通过差值进行替换，确保数据的连贯性。接下来,构建一系列反映车辆运行状态的指标,包括车辆运行状态、车辆速度、车辆加速度等。这些指标能够共同形成车辆状态的时序数据。最后,利用基于两阶段 STFT 的时频分析模型，可以对这些时序数据进行周期特征挖掘,

得出车辆行驶的周期性规律。思维导图如下：

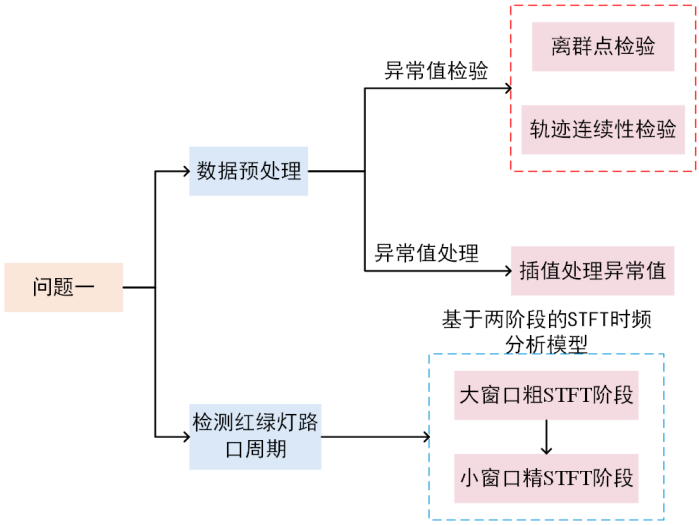


图 1　问题一思维导图

4.2 数据预处理

4.2.1 确定正方向

根据标准的笛卡尔坐标系定义,x 轴正方向(向右)为东方向,y 轴正方向(向上)为北方向。因此,通过可视化附件 1 中的坐标数据,可以得到车辆的轨迹散点图。以 A4 中的数据为例，将其可视化如下图：

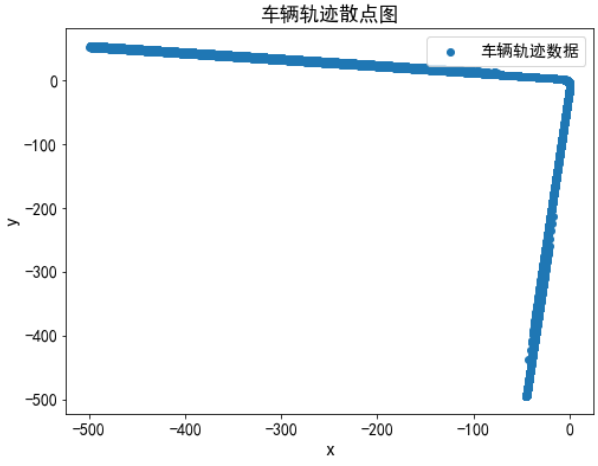


图 2 A4 中车辆轨迹散点图

4.2.2 异常值的筛选与处理

- **离群点检测**

在基于距离的离群点检测方法中，离群点就是远离大部分对象的点，即与数据集中的大多数对象的距离都大于某个给定阈值的点。基于距离的检测方法考虑的是对象

给定半径的邻域。如果在某个对象的邻域内没有足够的其他的点，则称此对象为离群点。本文定义偏离大多数车辆轨迹坐标的点为离群点。

对于数据对象集合 D，指定一个距离阈值 r 定义对象的邻域。设对象 d，$d' \in D$。考察 d 的 r 邻域中的其他数据对象的个数，如果 D 中的大多数对象远离 d，则 d 被视为一个离群点。令$r(r \geqslant 0)$是距离阈值，$a(0 \leqslant a \leqslant 1)$为分数阈值，如果 d 满足下式，则 d 是一个$DB(r,a)$离群点。

$$\frac{\| \{d' \mid \mathrm{dist}(d,d') \leq r\} \|}{\| D \|} \leq a \qquad (1)$$

- **轨迹的连续性检验**

为了保证行车轨迹的连续性，本文通过判断车辆的速度是否超过通过信号灯路口的规定速度进行判断。根据法律法规，车辆在通过设有交通信号灯的十字路口或转弯处，时速不得超过$30km/h$（$\approx 8.33m/s$）。若汽车的速度超过该允许值，则说明轨迹不连续。

- **异常值处理**

对于异常值点,计算其前后两个坐标点的平均值,并将该点的坐标替换为平均值。这样可以保留数据的整体趋势,同时减小异常值对分析结果的影响。

$$x_m = \frac{x_{m-1} + x_{m+1}}{2}, \quad y_m = \frac{y_{m-1} + y_{m+1}}{2} \qquad (2)$$

上式中，(x_m, y_m)为异常值点的新坐标，(x_{m-1}, y_{m-1})为异常值点的前一个坐标点，(x_{m+1}, y_{m+1})为异常值的后一个坐标点。

4.2.3 指标构建

汽车在通过信号灯路口的过程中分为两种状态：行（R）与停（P）。当汽车的速度$v \in [0, 0.5]\, m/s$时，表示汽车遇到红灯而处于停止状态；当速度$v > 0.5m/s$时，表示汽车正在前进。定义加速度$a < -3m/s^2$时代表汽车遇到红灯正在减速；$a > 3m/s^2$时代表红灯变绿灯，汽车开始行的状态。

$$v \in \begin{cases} [0, 0.5]\, m/s, \text{ 停}(P) \\ (0.5, +\infty)\, m/s, \text{ 行}(R) \end{cases} \qquad (3)$$

定义加速度$a < -3m/s^2$时代表汽车遇到红灯正在减速；$a > 3m/s^2$时代表红灯变绿灯，汽车开始行的状态。

$$state=\begin{cases}1, & a\in(3,+\infty)\,m/s^2\\ 0, & a\in[-3,3]\,m/s^2\\ -1, & a\in(-\infty,3)\,m/s^2\end{cases} \tag{4}$$

以 A1 的 8 号车为例，绘制该车辆通过信号灯路口的速度变化图，并在图中标出了汽车遇到红灯正在减速以及红灯变绿灯汽车开始加速的状态。

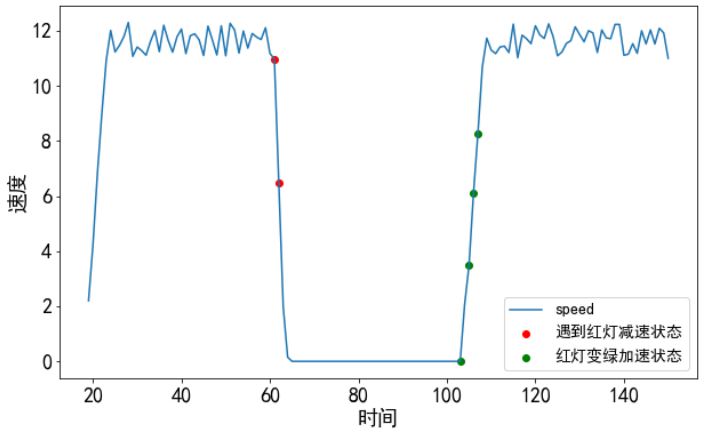


图 3 A1 路口 8 号车速度变化图

4.3 基于两阶段 STFT 的时频分析模型

本文建立一种两阶段 STFT 模型来提高时频分析性能：粗 STFT 阶段和精 STFT 阶段。在粗 STFT 阶段，划分接收信号到多个短时窗口，并进行傅里叶变换；在精 STFT 阶段，利用两个窗口内粗估计结果来校正另一个窗口的搜索范围，从而过滤掉更多噪声并提高检测性能[1]。

4.3.1 信号模型

在汽车状态变化信号不明显的情况下，时序数据中噪音和接收信号的能量相当，很难有效甄别出噪音和时序数据的区别，从而降低了快速傅里叶变换（Fast Fourier Transform, FFT）的检测效果。本文采用基于两阶段 STFT 对时序数据进行分析，能有效提高监测效率。将文中时序数据 r 经过离散化采样后，一个一个接收码元可表示为：$r=\{r_0,...,r_{N-1}\}$，其中 r 的第 n 个元素 r_n 可表示为：

$$\begin{aligned}&r_n=A\cos\left[2\pi\left(\frac{f_{d,0}}{f_s}n+0.5\frac{f_a^2}{f_s^2}n^2\right)+\varphi_0\right]+w_n,\\ &n=1,2,...,N+(1-\alpha)(M-1)N\end{aligned} \tag{5}$$

上式中，M 为短时窗口个数，N 是短时窗口长度，α 为短时窗口间重叠因子，A 是信号幅度，$f_{d,0}$ 为初始频率，f_a 为加速度，f_s 是采样频率，φ_0 是初始相位，ω_n 为均

值为 0、方差为σ^2的加性高斯白噪声。

4.3.2 两阶段 STFT

短时傅里叶变换（Short Time Fourier Transform, STFT）的实质是加窗的傅里叶变换。在分析非平稳信号时，一个很短的窗口内的信号可以看做平稳信号进行傅里叶变换，然后对信号进行时频分析。但是，在微弱信号场景下，估计信号频率的正确率随噪声的变大而减小，从而降低了 STFT 的时频分析效果[1]。

针对微弱信号，本文提出一种两阶段 STFT 方法，包括粗 STFT 阶段和精 STFT 阶段；其中，精 STFT 阶段是用来缩小粗 STFT 阶段的搜素范围的，如下图所示：

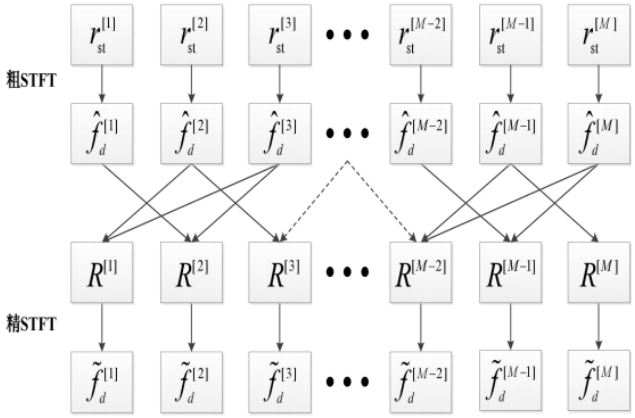


图 4 原理图

STEP1: **将收到的信号，依次划分到 M 个短时窗口中。**

其中第$t(1\leqslant t\leqslant M)$个窗口内的 N 个信号为$\boldsymbol{r}_{\mathrm{st}}^{[t]}=\{r_{\mathrm{st},1}^{[t]},...,r_{\mathrm{st},N}^{[t]}\}$，在$\boldsymbol{r}_{\mathrm{st}}^{[t]}$中，第$k(1\leqslant k\leqslant N)$个信号$\boldsymbol{r}_{\mathrm{st},k}^{[t]}$，如下所示：

$$r_{\mathrm{st},k}^{[t]}=r_{(i-1)(1-\alpha)N+k} \tag{6}$$

STEP2: **对每个窗口内信号进行傅里叶变换。**

对第$t(1\leqslant t\leqslant M)$个窗口内的 N 个信号$\boldsymbol{r}_{\mathrm{st}}^{[t]}$进行$N_f(N_f\geqslant N)$点傅里叶变换，从而得到频域向量$\tilde{\boldsymbol{r}}_{\mathrm{st}}^{[i]}=\{\tilde{r}_{\mathrm{st},1}^{[i]},...,\tilde{r}_{\mathrm{st},N_f}^{[i]}\}$。在$\tilde{\boldsymbol{r}}_{\mathrm{st}}^{[i]}$中，搜索最大元素$\tilde{\boldsymbol{r}}_{\mathrm{st},\hat{\lambda}_{\max}^{[t]}}^{[i]}$，即$\tilde{\boldsymbol{r}}_{\mathrm{st},\hat{\lambda}_{\max}^{[t]}}^{[i]}=\max\ \tilde{\boldsymbol{r}}_{\mathrm{st}}^{[i]}$。此时最大元素的位置为$\hat{\lambda}_{\max}^{[t]}$，如下所示：

$$\hat{\lambda}_{\max}^{[t]}=arg\max\ \tilde{\boldsymbol{r}}_{\mathrm{st}}^{[i]} \tag{7}$$

根据最大元素$\tilde{\boldsymbol{r}}_{\mathrm{st},\hat{\lambda}_{\max}^{[t]}}^{[i]}$的位置$\hat{\lambda}_{\max}^{[t]}$，本文可以得到粗估计结果$\hat{f}_d^{[t]}$，如下所示：

$$\hat{f}_d^{[t]} = \frac{\left(\hat{\lambda}_{\max}^{[t]} - 1\right)}{N_f} f_s \tag{8}$$

STEP3: 利用临近两个粗估计结果，校正每个窗口的搜索范围。

对于第 t 个短时窗口，其搜索范围为$R^{[t]}$：

$$R^{[t]} = \left[\max\left\{ \hat{f}_{\min}^{[t]} - \frac{2Nf_{a,\max}}{f_s}, -f_s/2 \right\}, \min\left\{ \hat{f}_{\max}^{[t]} + \frac{2Nf_{a,\max}}{f_s}, f_s/2 \right\} \right] \tag{9}$$

上式中，$\hat{f}_{\max}^{[t]} = \max\left\{ \hat{f}_d^{[t_1]}, \hat{f}_d^{[t_2]} \right\}$，$\hat{f}_{\min}^{[t]} = \min\left\{ \hat{f}_d^{[t_1]}, \hat{f}_d^{[t_2]} \right\}$，$f_{a,\max}$是加速度的最大可能值。另外，当$t=1$时，$t_1=2, t_2=3$；当$t=M$时，$t_1=M-2, t_2=M-1$；$2 \leqslant t \leqslant M-1$时，$t_1=t-1, t_2=t+1$。

从频域向量$\tilde{\boldsymbol{r}}_{\mathrm{st}}^{[t]}$中，搜素$R^{[t]}$范围内最大元素$\tilde{\boldsymbol{r}}_{\mathrm{st},\hat{\lambda}_{\max}^{[t]}}^{[t]} = \max\ \tilde{\boldsymbol{r}}_{\mathrm{st}}^{[t]}$，即

$$\begin{aligned} &\tilde{\lambda}_{\max}^{[t]} = \operatorname{argmax} \tilde{\boldsymbol{r}}_{\mathrm{st}}^{[t]} \\ &\text{s.t. } \left(\tilde{\lambda}_{\max}^{[t]} - 1\right) f_s / N_f \in R^{[t]} \end{aligned} \tag{10}$$

STEP4: 根据最大元素$\tilde{\boldsymbol{r}}_{\mathrm{st},\hat{\lambda}_{\max}^{[t]}}^{[t]}$的位置$\hat{\lambda}_{\max}^{[t]}$，得到精确估计结果$\tilde{f}_d^{[t]}$，如下所示：

$$\tilde{f}_d^{[t]} = \frac{\left(\hat{\lambda}_{\max}^{[t]} - 1\right)}{N_f} f_s \tag{11}$$

4.4 模型求解与结果分析

4.4.1 预估红灯时长

由于附件提供的数据是针对一个小时内,一个路口单向通过的所有车辆的行车轨迹。由于车辆数量较大，因此,本文认为可以取处于车速为$0m/s$（即处于停止状态）的最长时间作为该路口的红灯周期时长。以 A4 路口为例，该路口在一个小时内的停车等待时间频率图如下：

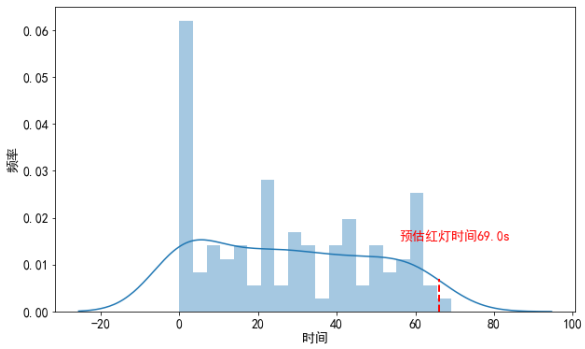


图 5 A4 路口的停车等待时间频率图

4.4.2 功率谱密度图得到完整周期时长

我们对 A4 路口车辆状态变化的时序数据导入模型进行计划，并对 STFT 结果进行幅度平方操作，得到信号在每个频率点上的功率大小。然后,将功率值作为纵轴，对应的频率值作为横轴，绘制出 PDS 图，以展示信号在不同频率上的能量分布情况。PDS 结果图如下：

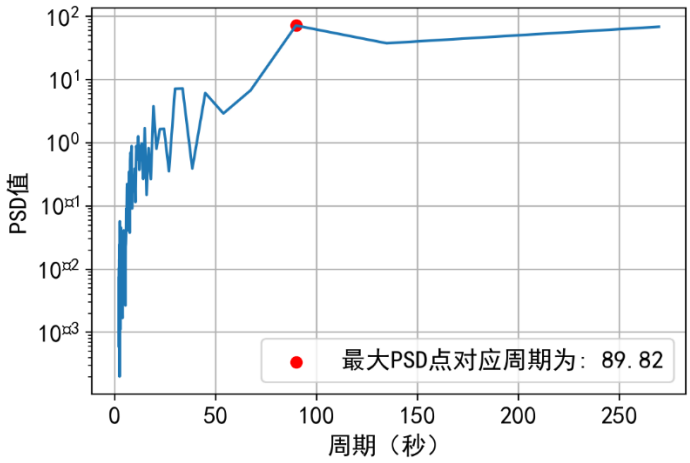


图 6　PDS 结果图

可以发现，最大 PSD 点对应周期为 89，因此 A4 路口的红绿灯完整周期为 89s，根据上述估计红灯周期为 69s，可以得到绿灯周期为 89-69=20s。

4.4.3 A1-A5 所有路口的红绿灯周期时长

利用所建立的模型对所有路口车辆状态的时序数据进行求解，得到 A1-A5 所有路口的红绿灯周期如下：

表 1　A1-A5 路口红绿灯周期结果

路口	A1	A2	A3	A4	A5
红灯时长（秒）	73	59	81	69	63
绿灯时长（秒）	33	28	26	20	26

五、 问题二的模型建立与求解

5.1 问题分析

实际上,只有部分用户在使用该公司的产品，因此只能获取部分样本车辆的行车轨迹数据。而且,样本车辆比例的减少、车流量的减少以及存在定位误差等因素，都会对

上述 STFT 预测模型的预测精度产生较大的影响。因此，本文在第一问的基础上结合 DBSCAN 聚类分析，通过聚类可以得到这一次红灯结束的时间到下一次红灯结束的时间，即为红绿灯完整周期时长或者周期时长的倍数，进而得到红绿灯完整周期得大致范围，从而对第一问中的两阶段 STFT 的时频分析模型进行修正。

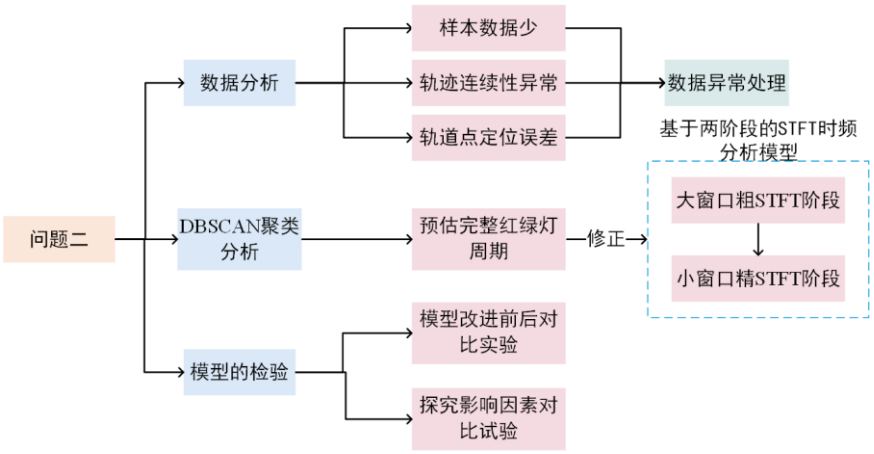


图 7　问题二思维导图

5.2 基于 DBSCAN 聚类分析改进的周期检测模型

5.2.1 持续性分析

首先对附件处理得到的时序数据进行持续性分析，从而为后续建模和计算做准备。平稳时间序列建模：如果时间序列$\{X_t, t \in T\}$在某一常数附近波动且波动范围有限，即有常数均值和常数方差，并且延迟 k 期的序列变量的自协方差和自相关系数是相等的，或者说延迟 k 期的序列变量之间的影响程度是一样的，则称$\{X_t, t \in T\}$为平稳序列，即具有持续性。

1)　计算自相关系数 ACF。设定 k 阶滞后，其自相关系数定义为

$$\beta = \frac{E(x_t, x_{t+k})}{E(x_i^2)} \tag{12}$$

2)　计算偏自相关系数 PACF

$$\gamma_{kk} = \frac{\beta_k - \sum_{k-1}^{j-1} \gamma_{k-1,j} \beta_{k-1}}{1 - \sum_{k-1}^{j-1} \gamma_{k-1,j} \beta_j} \tag{13}$$

3)　ARMA 模型识别。

模型定阶，根据$ARMA(p,q)$模型的自相关系数和偏自相关系数的性质选择合适的模型进行识别检验。判断相对应时刻及对应状态 state 的持续性。若具有持

续性则进行以下联动性分析，不具有持续性的数据则为非风险性异常数据，予以剔除。

5.2.2 DBSCAN 聚类算法

常见的判断异常数据的聚类算法有 K－means 聚类、系统聚类、DBSCAN 聚类等。本文比较多种聚类算法，最终选择基于密度的聚类算法（DBSCAN）聚类算法提取异常数据，同时利用轮廓系数和噪声点比例来确定最优的参数选择。

DBSCAN 的重要参数之一为 Eps，表示领域的半径大小，半径$d(x_i,x_j)$计算的是数据x_i和x_j之间的欧式距离，其中x_i代表时间，x_j代表 state 指标。公式如下，其中 m 表示数据的维度：

$$d(x_i,x_j)=\sqrt{\sum_{k-1}^{m}(x_{ik}-x_{jk})^2} \tag{14}$$

Eps 领域定义为传感器数据空间 D 中与其中某个数据x_i的距离不大于 Eps 的数据集合，Eps 领域公式如下：

$$N_{eps}(x_i)=\{x_j\in D|d(x_i,x_j)\leq Eps\} \tag{15}$$

另一重要参数为密度的阈值 MinPts，依此来划分核心点，边界点，噪音点。

若某个传感器数据的 Eps 邻域中的传感器数据的量大于等于密度的阈值 MinPts，称该数据点为核心点，记核心点组成的集合为D_{center}可以表示为：

$$D_{center}=\{x_i|x_i\in D|N_{Eps}(x_i)|\geq MinPts\} \tag{16}$$

若在某数据点的领域内至少有一个核心点，则该点可以看作为边界点，记边界点的集合为D_{border}，可以表示为

$$D_{border}=\{x_i|x_i\in D,N_{Eps}(x_i)\cap D_{center}\neq\varnothing\} \tag{17}$$

核心点和边界点以外的点都称为噪音点，记噪音点的集合为D_{noise}，其示意图如下：

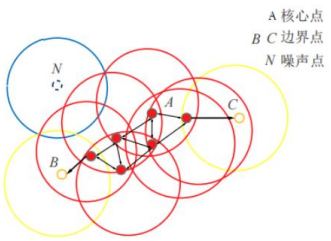


图 8 DBSCAN 算法示意图

此外我们根据轮廓系数、噪音点比例对参数的选择进行比较，求解噪音点比例，即用噪音点的数目除以总的数目，公式如下：

$$\alpha = \frac{Num_{noise}}{Num_{noise} + border + center} \tag{18}$$

解轮廓系数公式如下：

$$S(x_i) = \frac{b(x_i) - a(x_i)}{\max\ a(x_i), b(x_i)} \tag{19}$$

其中$a(x_i)$表示x_i到同簇内其他点不相似程度的平均值，体现凝聚度；$b(x_i)$表示x_i向量到其他簇的平均不相似程度的最小值，体现分离度。S(xi) 越接近 1，表示聚类的效果越好。

5.3 模型求解与结果分析

5.3.1 模型求解——DBSCAN 算法流程

邻域内至少有一个核心点。核心点和边界点以外的点都称为噪音点通过判断点 P 周围半径为 Eps 内点的个数是否小于某个值 MinPts，从而判断点 P 是否为核心对象。

若集合中存在一点 O 同时在核心对象 p 和 q 的邻域内，则核心对象 p 和 q 密度相连。DBSCAN 的目的便是找到所有密度相连的数据点，以此建立正常数据集。

Step1:对所有样本数据添加常值纵坐标；

Step2:对平面内直线上这些点标记为 0；

Step3: 随机选取一个为 0 的点将其标记为 1，检验在其半径 r 内的样本点是否（n），否则为离群点。（如果圈内点有从属类，则这个点也归为这类；否则定义新类，则此点属于新类。）；

Step4:重复操作直至所有点均被标记为 1。

5.3.2 ACF 和 PACF 判断

以 B1 路口为例，计算其 ACF 和 PACF 的结果如下：

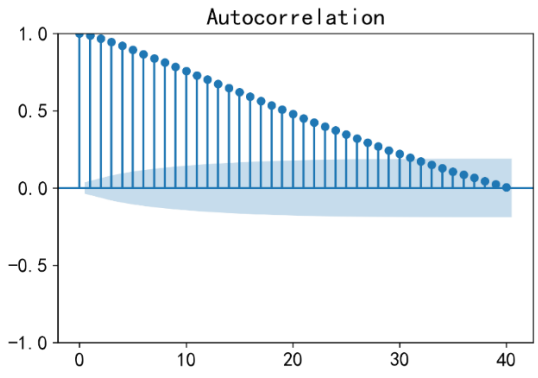


图 9 ACF 示意图

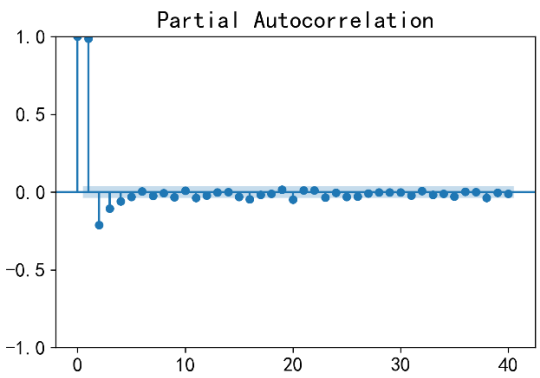


图 10 PACF 示意图

根据 ACF 和 PACF 图分析后，可以发现 state 数据的 ACF 图呈现明显的周期性波动,表示序列存在周期性成分，并且 ACF 图在滞后阶数处截断,说明序列存在特定阶数的自相关性。可以发现数据具有一定的平稳性。

5.3.3 DBSCAN 聚类结果

以 B1 路口为例，DBSCAN 聚类结果如下：

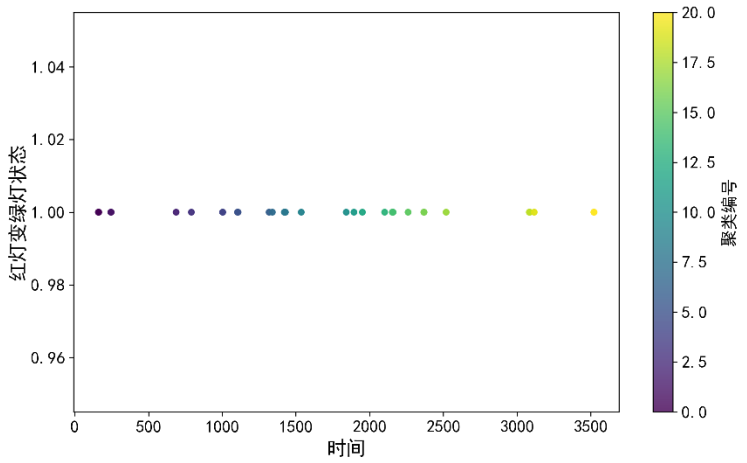


图 11 聚类结果图

由图可知，根据前文设定的 state 指标，聚类结果为 20 个类，图中每一个点代表红灯变绿灯的时间点，说明在 B1 路口连续一个小时内至少经历了 20 个周期。因此，每两个相邻点所对应的时间的差值为一个完整红绿灯周期或者完整周期的倍数。

5.3.4 B1-B5 路口的红绿灯周期结果

表 2 B1-B5 路口红绿灯周期结果

路口	B1	B2	B3	B4	B5

红灯时长（秒）	78	83	54	78	95
绿灯时长（秒）	27	33	34	27	21

5.4 模型的检验

5.4.1 模型改进前后对比

将第一问模型与第二问模型求解完整红绿灯周期结果进行对比：

表 3 模型改进前后检测周期结果

	B1	B2	B3	B4	B5
原模型周期	101	121	299	113	163
新模型周期	105	116	88	105	116

5.4.2 模型影响因素探究

为详细探究车样本车辆比例、车流量、定位误差对模型的影响，本文设计对比实验，去探究各个因素的影响即影响程度。

- **样本车辆比例的影响**

本文随机选取 10%~20%的各个路口数据中的车辆代号，将选取的车辆代号从数据中抹除，以模拟样本车辆减少时，模型周期检测结果的变化。

- **定位误差的影响**

本文对各个路口车辆定位数据随机添加-15%~15%的扰动，以模拟样本车辆定位不精准定位数据错误时，模型周期检测结果的变化。

- **最终结果**

表 4 对比实验结果

	B1	B2	B3	B4	B5
减少车辆比例	101	113	99	113	88
增加定位误差	105	121	88	105	113
原始结果	105	116	88	105	116

分析上表可以得出以下结论：减少样本车辆数量可能会对数据产生较大影响，由于样本量下降，时序数据量会相应减少，从而影响到周期预测结果的准确性；在定位误差方面，模型已经采取了突出点检测、数据连续性检验和数据滤波、平滑等处理手段,因此定位误差对最终结果的影响较小，处于正常的误差波动范围内，这同时也表明

所使用的模型具有较好的鲁棒性，能够在一定程度上抵御数据噪声的干扰。

六、 问题三的模型建立与求解

6.1 问题分析

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

因此本文使用贝叶斯变点检测检测出周期变化点，然后将时序数据沿变化点进行分段，对每段时序数据使用加 manning 窗的 FFT 算法求解周期，通过上述方式得到周期的变化时间点，然后对时间点分割，对分割后的数据采用前文模型，得到具体周期时间。思维导图如下：

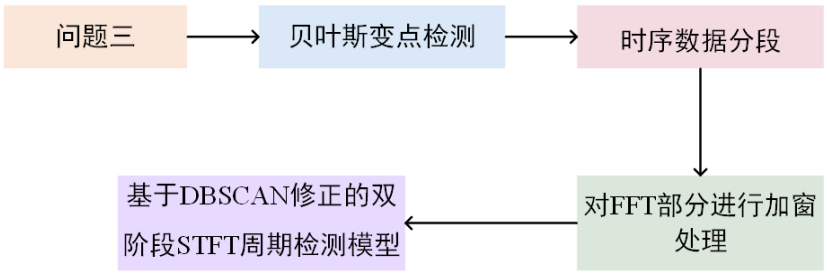


图 12　问题三思维导图

6.2 贝叶斯变点检测

贝叶斯变点检测的步骤如下[6]：

Step1：输入观测变量$x_1, x_2, \cdots, x_n$；

Step2：初始化，设定$t=0$的时刻与任意一个θ值$(0<\theta<1)$；

Step3：观测新数据x_t：(i) 初始化后，t 从 1 开始，或(ii) t 是从上一个周期开始标识的第一个断点；

Step4：通过下式计算$\omega_{t+1}^{(j)}$：

$$\omega_{t+1}^{(j)}=\begin{cases}\dfrac{\sqrt{t-j+1}\,\Gamma\left(\frac{t-j+2\alpha+1}{2}\right)(B+2\beta)^{-\frac{t-j+2\alpha+1}{2}}}{\sqrt{\pi(t-j+2)}\,\Gamma\left(\frac{t-j+2\alpha}{2}\right)}(A+2\beta)^{-\frac{t-j+2\alpha}{2}} & j<t \\ (\pi)^{-\frac{1}{2}}2^{\alpha-\frac{1}{2}}\Gamma\left(\frac{2\alpha+1}{2}\right)\left(\frac{x_{t+1}^2}{2}+2\beta\right)^{-\frac{2\alpha+1}{2}} & j=t\end{cases} \tag{20}$$

Step5：根据下式计算转移概率$P(R_{t+1}=j|R_t=i)$

$$P(R_{t+1}=j\,|\,R_t=i)=\begin{cases}\dfrac{1-G(t-i)}{1-G(t-i-1)} & j=i \\ \dfrac{G(t-i)-G(t-i-1)}{1-G(t-i-1)} & j=t \\ 0 & \text{其他}\end{cases} \tag{21}$$

Step6：对所有的$j\leqslant t$，根据 Step5 中的式子计算变点后的后验概率$P(R_{t+1}=j|R_t=i)$，如果后验概率的最大值达到$\hat{J}_1$，并且大于阈值p_0，则$\hat{\tau}_1=\hat{J}_1$，并跳转到 Step6。否则，没有更多的变点。

Step7：返回到 Step3~ Step6，使用$t=\hat{J}_1$直到不再识别出变点，$\hat{J}_1,\hat{J}_2,\cdots\hat{J}_m$，将是变点位置的最终估计值。

Step8：估计$\hat{\tau}_1$、分段均值和置信区间等。

6.3 加窗插值 FFT 算法

为减小普通 FFT 算法存在的频谱泄漏和栅栏效应所产生的误差，并有效预测信号周期变化，本文使用加窗插值法对信号的相位和幅值进行修正。一些常用的窗函数包括矩形窗、布莱克曼窗、汉宁窗和海明窗等，我们根据各窗函数的特性可知：矩形窗的主瓣最窄，但旁瓣较高，泄漏较大；布莱克曼窗虽然旁瓣衰减值大，但其主瓣很宽，因此计算相对复杂；Hamming 窗的旁瓣衰减略比 Hanning 窗大，但随旁瓣的增加其衰减速度很慢；Hanning 窗分析数据信号时，除计算量较小外，还可通过调节采样长度来减少谐波间泄漏。为此采用双峰谱线插值修正和多项式逼近相结合的方法，来最终确定 Hanning 窗修正公式:

1) Hanning 窗公式为:

$$w(n)=0.5-0.5\cos(\frac{2\pi n}{N}),\text{其中}(n=0，\ 1，\ \ldots，\ N-1); \tag{22}$$

2) 差值公式为

$$\beta=\frac{y_2-y_1}{y_2+y_1},\quad \alpha=1.5\beta,\quad A=N^{-1}(y_1+y_2)(a+ba^2+ca^4+da^6)。\qquad (23)$$

其中峰值点附近幅度最大和次最大谱线幅值分别 为 y1、y2，且式中 a、b、c、d 取值分别为: 2.35619403、1.15543682、0.32607873、0.07891461。

模型示意图如下：

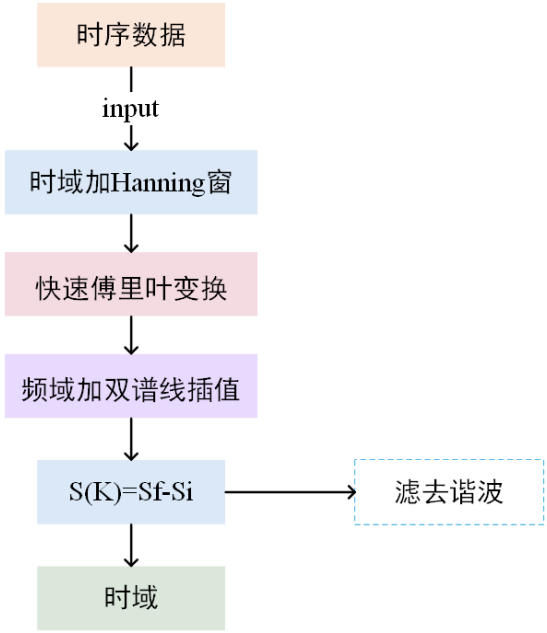


图 13　加窗差值 FFT 示意图

6.4 模型求解与结果分析

6.4.1 贝叶斯变点检测部分结果图

首先使用前文的提到的双阶段 STFT 时频分析模型对附件中的 C1 的车辆状态的时序数据处理分析，去除趋势部分，得到周期部分。随后使用贝叶斯变点检测算法对处理后的数据进行突变点检测。得到结果如下，横坐标为时刻，纵坐标为频率，图中红色虚线代表突变时刻，即周期切换时刻：

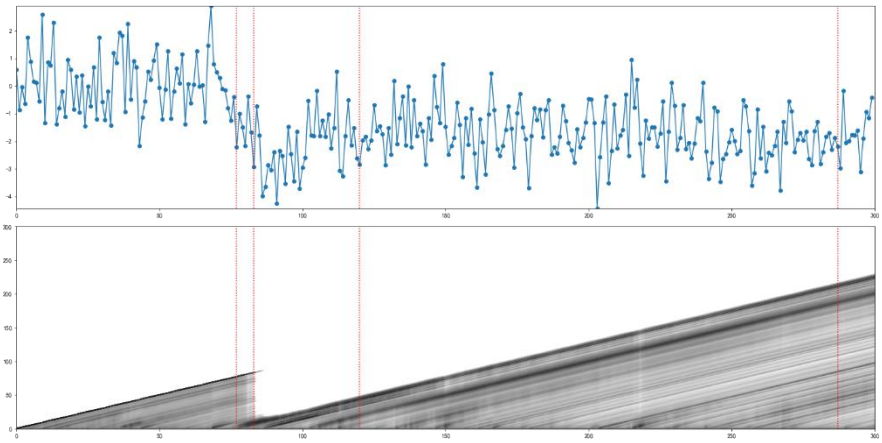


图 14　贝叶斯变点检测结果图

6.4.2 信号灯周期结果

C1-C6 路口每次的周期切换时刻以及新旧周期参数如下：

	C1	C2	C3	C4	C5	C6
周期 1 红灯时长（秒）	56	42	79	77	51	77
周期 1 绿灯时长（秒）	32	23	26	28	37	28
周期切换时刻（秒）	1494	246	1340	无	1318	无
周期 2 红灯时长（秒）	45	48	38		46	
周期 2 绿灯时长（秒）	16	28	23		24	
周期切换时刻（秒）	1670	398	1443		1670	
周期 3 红灯时长（秒）	56	67	79		51	
周期 3 绿灯时长（秒）	32	21	26		37	
周期切换时刻（秒）	2374	2092	2598		3342	
周期 4 红灯时长（秒）	45	48	38		38	
周期 4 绿灯时长（秒）	16	28	23		27	
周期切换时刻（秒）	2726	2245	6483		3606	
周期 5 红灯时长（秒）	56	67	52		51	
周期 5 绿灯时长（秒）	32	21	28		37	
周期切换时刻（秒）	2990	3325	6798		6510	

七、 问题四的模型建立与求解

7.1 问题分析

问题四涉及某路口的全方位数据，我们可以将其拆分为前述问题的子问题。具体来说，先对该路口的车辆运行模式进行分解分析，然后应用前文提及的模型对各个方向进行周期性特征提取。接下来，可以利用改进的线性互相关算法对周期预估结果进行验证。通过这种方式，们可以将问题四转化为先前提出的问题类型，并继续采用前文介绍的分析方法进行求解。这样不仅避免重复阐述，也能充分利用前期的建模和分析思路。

7.2 多方向汽车轨迹分解

附件 4 给出了某路口连续 2 小时内各个方向车辆的轨迹数据，可以对其进行可视化展示。如果按照前述分析模型，用车辆停止时间来预估红灯时长，会发现任何时刻都存在车辆在行驶和等待红灯的情况。因此，无法直接从这些数据中得出红绿灯的具体时长。换个角度来说，依赖车辆停止时间这一指标，无法准确推断出红绿灯信号的持续时间。我们需要探索其他方法，才能从给定的轨迹数据中有效地提取出红绿灯时长的信息。

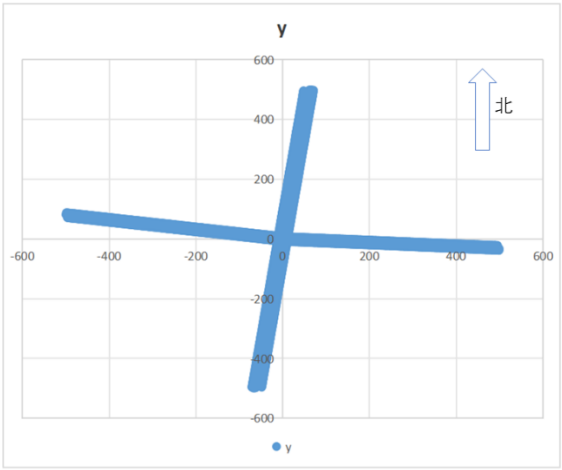


图 15　D 路口车辆轨迹图

因此，本文将对路口 D 各个方向的车辆轨迹数据进行独立研究分析。以路口 D 从东向南方向为例，如下图所示。在后续中，我们将继续分析该方向所有车辆的轨迹数据及其行车状态，作为例子。

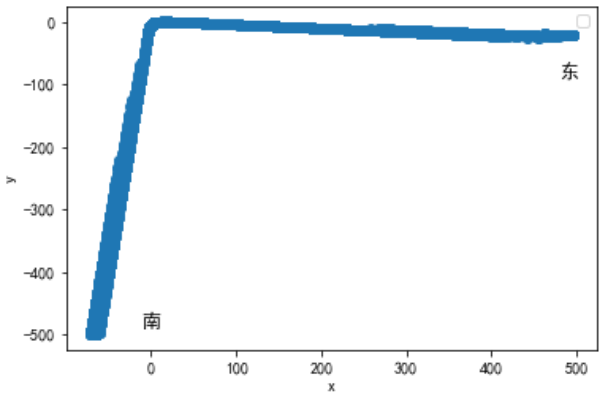


图 16　D 路口从东开往南车辆轨迹汇总

7.3 滑动窗口稳态检测模型

7.3.1 滑动窗口原理

本文使用一种基于滑动窗口的快速筛选时序数据稳态的方法。该方法将包含一段数据的滑动窗口作为判断稳态数据的最小单元。每次窗口向前滑动一步，都会有一个新数据进入窗口，同时一个旧数据滑出。我们计算窗口内数据的标准差，并与给定阈值进行比较。若标准差小于阈值,则将该窗口内的数据标记为稳态数据。假设窗口长度为 p，则窗口数据的平均值、方差和标准差可以用以下表达式计算:

$$\bar{x}_1 = \frac{1}{p}\sum_{i=1}^{p} x(i)$$

$$Var_1 = \sum_{i=1}^{p} (x(i) - \bar{x}_1)^2 \tag{24}$$

$$Sd_1 = \sqrt{\frac{Var_1}{p-1}}$$

当窗口位于 k 点时，窗口数据均值为 $\bar{x}_k$，当位于 k+1 时刻时，窗口均值为

$$\bar{x}_{k+1} = \bar{x}_k + \frac{(x(k+p) - x(k))}{p} \tag{25}$$

由此，得到 k 时刻的滑动窗口内的数据方差值为：

$$\begin{aligned} \mathrm{Var}_{k+1} &= \sum_{i=k+1}^{k+p} x^2(i) - p \cdot \bar{x}_{k+1}^2 \\ &= \mathrm{Var}_k + (x^2(k+p) - x^2(k)) - \bar{p}(x_{k+1}^2 - \bar{x}_k^2) \end{aligned} \tag{26}$$

7.3.2 参数选择

窗口长度和标准差阈值是影响该算法性能的两个关键因素。当窗口长度过小时，

在滑动过程中窗口的平均值和标准差会过于受新数据的影响，不利于判断数据整体趋势。而当窗口过大时，算法对数据点的变化就会不太敏感。标准差阈值 Sd 是判断数据是否处于稳态的关键参数。为此，我们利用一段具有稳态和非稳态标签的历史数据作为识别样本，对窗口长度和标准差阈值进行识别。识别原理如下图所示，目标函数为滑动窗口法筛选出的稳态数据、非稳态数据与标记数据之间的误差。每当筛选结果与标记结果不同时，我们最终选取目标值最小时的窗口长度和阈值。

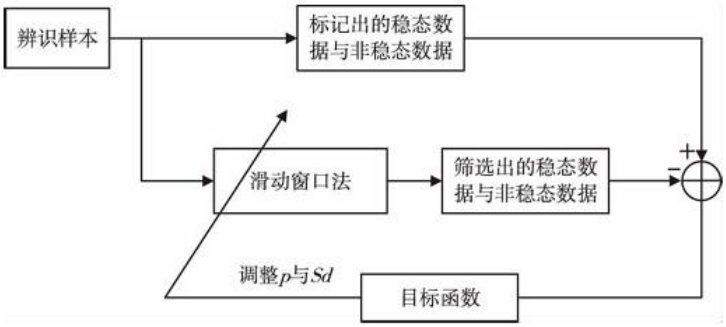


图 17　滑动窗口参数辨识原理示意图

7.4 模型求解与结果分析

路口 D 沿各方向的红绿灯周期结果如下：

	红灯周期（秒）	绿灯周期（秒）
北向南	96	46
北向东	125	17
北向西	92	50
西向东	102	40
西向南	95	47
西向北	113	29
南向北	96	48
南向西	116	26
南向东	90	52
东向西	101	41
东向南	111	31

东向北	95	47

根据上述模型，对各个方向红绿灯周期求解，结果如上表所示，所以方向行驶的红绿灯周期均为 142s，一定程度上说明了模型检验的准确性。

同时，按照现实生活的经验，通过信号灯路口时，北向南和南向北方向属于相反车流，遵守同一对信号灯，这两个方向的预估红绿灯周期应该比较接近。结果显示，北向南方向的红灯周期为 96 秒，绿灯周期为 46 秒；南向北方向的红灯周期也为 96 秒，绿灯周期也为 46 秒，符合预期。与之相同，西向东和东向西是遵守同一对信号灯的两个方向。其中，西向东的红灯周期为 102 秒，绿灯周期为 40 秒；东向西的红灯周期为 101 秒，绿灯周期为 41 秒，两者结果较为接近。

综上所述，上述预估结果与实际情况相符，且结果较为准确。这是由于相反方向遵守同一信号灯，其预估红绿灯周期应该相近；同时，遵守同一信号灯的两个方向，其预估结果也较为接近。

八、 模型的评价、改进与推广

8.1 模型的优点

1. 我们对附件中提供的数据进行了预处理。我们进行了离群点检测和轨迹的连续性检测。我们用前后两个车辆位置坐标的均值替换了异常值点，从而降低了主观因素对数据的影响，为后续建模做准备。
2. 本文建立的两阶段 STFT 时频分析模型，在粗 STFT 阶段,我们将接收信号划分为多个短时窗口,并对每个窗口进行傅里叶变换。在精 STFT 阶段,我们利用两个窗口内粗略估计的结果,来校正另一个窗口的搜索范围。这样不仅可以过滤掉更多噪声,还可以进一步提高检测性能。
3. 本文利用的 DBSCAN 算法能够有效识别数据中的噪声点和异常点,减小了异常值对周期检测的影响。DBSCAN 对异常点和噪声点的抗干扰能力强,不会因极端值的出现而产生较大偏差，更好的鲁棒性
4. 贝叶斯变点检测算法能够精准地识别序列中的变点位置,定位周期性的开始和结束时间。相比传统的基于窗口滑动的方法,贝叶斯方法能够更可靠地发现间隔不均匀的周期性。

8.2 模型的缺点

1. DBSCAN 算法需要设定两个关键参数:半径阈值 eps 和最小样本数 minPts。这些参数的选择会显著影响聚类结果,需要通过经验值和试错方式进行调整。
2. 贝叶斯变点检测方法需要事先设定合适的先验分布,这一选择对最终结果有很大影响。

8.3 模型的改进

1. 在 DBSCAN 的基础上引入确定最优聚类数的机制,提高周期数量判断的可靠性。可以采用 DBI、Calinski-Harabasz 等聚类质量评价指标来辅助选择。
2. 通过研究基于转移学习、元学习等方法,提高贝叶斯模型在小样本数据上的泛化性能。

九、 参考文献

[1] 吴同,苏莉娜,田佳俊,等.基于两阶段 STFT 的微弱信号时频分析方法[C]//中国自动化学会.2023 中国自动化大会论文集.南京邮电大学物联网学院;,2023:6.DOI:10.26914/c.cnkihy.2023.092688.

[2] 王海军,徐忠富,李志鹏,等.基于 STFT 预分选的 PRI 变换雷达信号分选方法[P].河南省:CN202010417093.3,2024-03-15.

[3] 全大英,李广阳,魏强.一种基于 GPU 异构并行加速的 STFT 时频分析实现方法[P].浙江省:CN202311470063.9,2024-02-20.

[4] 孙培清.基于 STFT 的雷达信号数字信道化侦收及实现方法研究[D].电子科技大学,2020.DOI:10.27005/d.cnki.gdzku.2020.001871.

[5] 唐建.从 CTFT 公式出发推导周期序列的 DTFT[J].电气电子教学学报,2024,46(01):133-136.

[6] 夏娟. 贝叶斯变点检测模型的构建及应用研究[D].华中农业大学,2023.DOI:10.27158/d.cnki.ghznu.2023.001874.

[7] 金鹏鹏.基于贝叶斯方法的时间序列变点问题研究[D].哈尔滨工业大学,2019.DOI:10.27061/d.cnki.ghgdu.2019.003261.

[8] 张婷.时间序列模型的变点检测及在预警监测中的应用[D].东南大学,2018.

[9] Marcela B S S ,Arrigo C ,AlbertoC T .Genetic algorithm with a Bayesian approach for multiple change-point detection in time series of counting exceedances for specific thresholds[J].Journal of the Korean Statistical Society,2023,52(4):982-1024.

[10] Konstantinos B ,Frederic S ,Panagiotis T .Predictive ratio CUSUM (PRC): A Bayesian approach in online change point detection of short runs[J].Journal of Quality Technology,2023,55(4):391-403.

附录

附录 1

介绍：支撑材料的文件列表

1.“华中杯”大学生数学建模挑战赛承诺书

2.img

3.代码文件 txt

4.处理后文件

附录 2

介绍：该代码是某某语言编写的，作用是什么

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import numpy as np
filename = 'data/2/B5.csv'
df = pd.read_csv(filename)
#对 df.x,df.y 做散点图，并做离群点检测
plt.figure(figsize=(8, 6))
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
#设置字体大小
plt.rcParams['font.size'] = 14
plt.scatter(df['x'], df['y'], label='车辆轨迹数据')
plt.xlabel('x',fontsize = 16)
plt.ylabel('y',fontsize = 16)
plt.title('车辆轨迹散点图')
plt.legend()
# plt.savefig('data2/img/B1_scatterDetect.png')
plt.show()
def detect_outliers(x, y, threshold=3):
    z_scores = np.abs(stats.zscore(np.vstack([x, y]).T))
    outlier_indices = np.where(z_scores > threshold)[0]
    return outlier_indices

outlier_indices = detect_outliers(df['x'], df['y'])
outliers = df.iloc[outlier_indices]
```

```
print(f"Outlier indices: {outlier_indices}")
print(f"Number of outliers: {len(outlier_indices)}")
print(outliers)
#在图中画出异常点
plt.figure(figsize=(8, 6))
plt.scatter(df['x'], df['y'], label='车辆轨迹数据')
plt.scatter(outliers['x'], outliers['y'], color='red', label='异常点')
plt.xlabel('x')
plt.ylabel('y')
plt.title('车辆轨迹散点图')
plt.legend()
plt.savefig('data2/img/B5_scatterDetect.png')
plt.show()


from statsmodels.tsa.seasonal import seasonal_decompose
#
#读取 data 中 1 中的 A1.csv 文件
import pandas as pd
import math
import os
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

df = pd.read_csv('data/4/D.csv')

# def cal_distance(df,x,y):
#       df['x_y'] = np.sqrt((df[x]-df[x].shift(1))**2+(df[y]-df[y].shift(1))**2)
#       return df
# df = cal_distance(df,'x','y')

df['x_y'] = 0
#对 df 中 vehicle_id 列进行分组
for i in df.groupby('vehicle_id'):
    #对每组的使用下一行 x,y 列与上一行的欧式距离，并将结果保存在 df 中，列名为 x_y
    df.loc[i[1].index,'x_y'] = np.sqrt((i[1].x.shift(-1)-i[1].x)**2+(i[1].y.shift(-1)-i[1].y)**2)
    #若 df.loc[i[1].index,'x_y']为 nan，则赋值 2
    df.loc[i[1].index,'x_y'] = df.loc[i[1].index,'x_y'].fillna(2)
#对 df 中 x_y 列中的值大于 0.5 的行，添加一个标签，值为 1，否则为 0，列名为 label
def add_label(df,x_y,label):
```

```
    df[label] = np.where(df[x_y]<0.5,1,0)
    return df
df = add_label(df,'x_y','label')
#根据 df 中的 time 分组，对 label 列求和
def cal_label(df,label):
    df['label_sum'] = df.groupby('time')[label].transform('sum')
    return df
df = cal_label(df,'label')
#输出 df 的 time 分组
s = df.groupby('time')['label'].sum()
#将 s 变为 dataframe，并对 value 添加标签 count，将 time 列名改为 data
s = pd.DataFrame(s).reset_index()
s.columns = ['date','count']
#计算 s 的周期
data = s.set_index('date')
# data.index = pd.to_datetime(data.index)
ts = data['count']
#探究 ts 的周期性
decomposition = seasonal_decompose(ts,period=72,
                                   model="add")
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
plt.plot(ts, label='Original')
plt.legend(loc='best')
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.savefig('data2/img/D_trend.png',dpi=200)
plt.show()


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf

# 给定数组
data = [134, 238, 344, 448, 658, 868, 1186, 1288, 1393, 1499, 1919,
        2023, 2233, 2338, 2443, 2550, 2653, 2758, 2863, 2969, 3073, 3493,
        3598]

# 转换为 numpy 数组
data_array = np.array(data)
```

```
# 差分序列分析
diff_data = np.diff(data_array)

# 打印差分序列
print("差分序列：", diff_data)

# 绘制差分序列图
plt.figure(figsize=(12, 4))
plt.plot(diff_data, label="差分序列")
plt.xlabel("索引")
plt.ylabel("差分值")
plt.title("差分序列图")
plt.legend()
plt.show()

# 自相关函数（ACF）分析
plt.figure(figsize=(12, 4))
plot_acf(diff_data, lags=1, title="差分序列自相关函数（ACF）")
plt.show()



array = [ 134,  246,  398,  485,  685,  948, 1028, 1213, 1653, 1828, 2004,
          2092, 2245, 2621, 2796, 2886, 3148, 3325, 3390, 3500, 3589, 3677,
          3764, 3940, 4028, 4292, 4380, 4469, 4646, 4820, 4997, 5172, 5436,
          5612, 5704, 5964, 6140, 6228, 6381, 6581, 6828, 7108, 7196]

percentage_threshold = 8  # 百分比阈值
pe = 88
differences = []
previous_difference = array[1] - array[0]
change_points = []

for i in range(2, len(array)):
    current_difference = array[i] - array[i - 1]
    relative_change = abs(abs(current_difference - previous_difference) % pe )
    print(current_difference , previous_difference)
    print(relative_change)
    if relative_change > percentage_threshold:
        change_points.append((i - 1, previous_difference))
        previous_difference = current_difference
    differences.append(current_difference)
```

```
if abs(current_difference - previous_difference) > percentage_threshold:
    change_points.append((len(array) - 1, current_difference))

print("等差值序列：", differences)
print("改变点及其对应的等差值：", change_points)



from scipy.fftpack import fft, fftfreq
import seaborn as sns
import matplotlib.pyplot as plt
def STFT(filename):
    df = pd.read_csv(filename)
    df['xy_distance'] = 0
    for i in df.groupby('vehicle_id'):
                df.loc[i[1].index,'xy_distance']   =   np.sqrt((i[1].x.shift(-1)-i[1].x)**2+(i[1].y.shift(-1)-
i[1].y)**2)/1
        df.loc[i[1].index,'xy_distance'] = df.loc[i[1].index,'xy_distance'].fillna(11)
    df['state'] = np.where(df['xy_distance']<0.5,1,0)
    df['state_sum'] = df.groupby('time')['state'].transform('sum')
    s = df.groupby('time')['state'].sum()
    s = pd.DataFrame(s).reset_index()
    s.columns = ['date','count']
    data = s.set_index('date')
    ts = data['count']
    fft_series = fft(ts.values)
    power = np.abs(fft_series)
    sample_freq = fftfreq(fft_series.size)
    pos_mask = np.where(sample_freq > 0)
    freqs = sample_freq[pos_mask]
    powers = power[pos_mask]
    top_k_seasons = 3
    top_k_idxs = np.argpartition(powers, -top_k_seasons)[-top_k_seasons:]
    top_k_power = powers[top_k_idxs]
    fft_periods = (1 / freqs[top_k_idxs]).astype(int)

    print(f"top_k_power: {top_k_power}")
    print(f"fft_periods: {fft_periods}")

    NFFT = 256
    TEMP = np.abs(np.fft.fft(ts.values, n=NFFT))
    PSD = (TEMP[0:NFFT//2+1])**2 / NFFT
    frequencies = np.linspace(0, 0.78/2, NFFT//2+1)  # 单边频率轴，从 0 到采样率一半
    periods = 1./frequencies  # 将频率转换为周期
```

```
    fig, ax = plt.subplots()
    ax.semilogy(periods, PSD, linewidth=1.5)  # 使用对数尺度展示功率谱密度
    #标出 periods 范围为 50~100 间的对应 PSD 的最大值点
    max_index = np.argmax((periods >= 50) & (periods <= 100) & (PSD > 0))  # 需确保 PSD 大于0，避免选取到噪声
    ax.scatter(periods[max_index], PSD[max_index], color='red', marker='o', label=f"最大 PSD 点对应周期为: {periods[max_index]:.2f}")
    # print(periods[np.argmax(PSD)], PSD[np.argmax(PSD)])
    # ax.scatter(periods[np.argmax(PSD)], PSD[np.argmax(PSD)], color='red', s=100)
    ax.set_xlabel('周期（秒）')
    ax.set_ylabel('PSD 值')
    ax.grid(True)
    ax.legend()
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    plt.savefig('data2/img/STFT.png',dpi=200)
    plt.show()

    from statsmodels.tsa.stattools import acf

    for lag in fft_periods:
        acf_score = acf(ts.values, nlags=lag)[-1]
        print(f"lag: {lag} fft acf: {acf_score}")

    expected_lags = np.array([30, 60, 90]).astype(int)
    for lag in expected_lags:
        acf_score = acf(ts.values, nlags=lag, fft=False)[-1]
        print(f"lag: {lag} expected acf: {acf_score}")
    pl = df
    print(pl.groupby('vehicle_id')['state'].sum().max(),pl.groupby('vehicle_id')['state'].sum().idxmax())
     print(pl.groupby('vehicle_id')['state'].sum().nlargest(2).iloc[1],pl.groupby('vehicle_id')['state'].sum().nlargest(2).index[1])
    print(pl.groupby('vehicle_id')['state'].sum().nlargest(3).mean())
    sns.distplot(pl.groupby('vehicle_id')['state'].sum(),bins=20)
STFT('data/4/D.csv')


#读取 data 中 1 中的 A1.csv 文件
import pandas as pd
import math
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import numpy as np
import pandas as pd

def Question1(filename,savepath):
    def add_label(df,x_y,label):
        df[label] = np.where(df[x_y]<0.5,1,0)
        return df
    df = pd.read_csv(filename)

    df['speed'] = 0
    for i in df.groupby('vehicle_id'):
        df.loc[i[1].index,'speed'] = np.sqrt((i[1].x.shift(-1)-i[1].x)**2+(i[1].y.shift(-1)-i[1].y)**2)
        df.loc[i[1].index,'speed'] = df.loc[i[1].index,'speed'].fillna(11)
    df = add_label(df,'speed','state')
    df['a'] = 0
    #通过计算下一行的 df.spead 减去本行的 df.speed 得到加速度赋值到 a 列
    df['a'] = df.groupby('vehicle_id')['speed'].shift(-1)-df['speed']
    #建立新的列 astate,如果本行 a 大于 3 则赋值为 1，如果小于-3 则赋值为-1，否则赋值为 0
    df['astate'] = 0
    df['astate'] = np.where(df['a']>2,1,0)
    df['astate'] = np.where(df['a']<-2,-1,df['astate'])
    df.to_csv('data2/'+savepath+'_speed.csv',index=False)
    def cal_label(df,label):
        df['state_sum'] = df.groupby('time')[label].transform('sum')
        return df
    df = cal_label(df,'state')
    s = df.groupby('time')['state'].sum()
    s = pd.DataFrame(s).reset_index()
    s.columns = ['date','count']
    s.to_csv('data2/'+savepath+'_count.csv',index=False)
Question1('data/1/A5.csv','1/A5')



df = pd.read_csv('data2/1/A1_speed.csv')
#显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.figure(figsize=(10,5))
#画出 vehicle_id=8 的速度变化图
# df[df['vehicle_id']==8].plot(x='time',y='speed')
# #对 df 中画出 vehicle_id=8 时 astate 列为-1 和 1 的点画出
# df[df['vehicle_id']==8][df['astate']==-1].plot(x='time',y='speed',color='red',label='astate=-1')
# df[df['vehicle_id']==8][df['astate']==1].plot(x='time',y='speed',color='blue',label='astate=1')
#将上述三图画到同一图中
```

```
plt.figure(figsize=(10,6))
plt.plot(df[df['vehicle_id']==8]['time'],df[df['vehicle_id']==8]['speed'],label='speed')
plt.scatter(df[df['vehicle_id']==8][df['astate']==-1]['time'],df[df['vehicle_id']==8][df['astate']==-
1]['speed'],color='red',label='遇到红灯减速状态')
plt.scatter(df[df['vehicle_id']==8][df['astate']==1]['time'][3:],df[df['vehicle_id']==8][df['astate']==1]['s
peed'][3:],color='green',label='红灯变绿加速状态')
plt.xticks(fontsize = 18)
plt.yticks(fontsize = 18)
plt.ylabel('速度',fontsize = 18)
plt.xlabel('时间',fontsize = 18)
plt.legend(loc = 'lower right')
plt.savefig('data2/img/A1_speed 示意图.png')
plt.show()


import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('data2/2/B1_speed.csv')
if len(df) >= 3:
    df['prev_astate'] = df['astate'].shift(1)
    df['next_astate'] = df['astate'].shift(-1)

    condition = (df['astate'] == 1) & \
                ((df['prev_astate'] != 1) & (df['next_astate'] != 1))
    rows_to_remove = df[condition]

    if not rows_to_remove.empty:
        df = df.drop(rows_to_remove.index, axis=0)
        print("已成功移除满足条件的行")
    else:
        print("无需要移除的行")
else:
    print("数据长度不足，无法检查条件")
#将 df 中的 time 与 astate 列不为 0 与-1 的值建立二维数组
X = df[['time','astate']].values[df['astate'] > 0]
#除去单行 astate 为 1 的独立数据
#对 X 进行裁剪，得到部分数据

# 数据标准化
# scaler = StandardScaler()
```

```
# X_scaled = scaler.fit_transform(X)
X_scaled = X
# 应用 DBSCAN 聚类
eps = 3
min_samples = 1
db = DBSCAN(eps=eps, min_samples=min_samples).fit(X_scaled)
y_pred = db.fit_predict(X)
print(y_pred)
# 绘制聚类结果
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=db.labels_, cmap='viridis', s=20, alpha=0.8)
# plt.title('DBSCAN Clustering')
plt.xlabel('时间',fontsize = 16)
plt.ylabel('红灯变绿灯状态',fontsize = 16)
plt.colorbar(label='聚类编号')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.savefig('data2/img/B1_dbscan.png',dpi=300)
plt.show()


import numpy as np

# 获取 DBSCAN 聚类结果
labels = db.labels_

# 初始化簇中心列表
cluster_centers = []

# 遍历所有簇
unique_labels = set(labels)
for label in unique_labels:
    # 检查是否为噪声点（label=-1）
    if label == -1:
        continue

    # 找出属于该簇的所有样本点
    cluster_points = X[labels == label]

    # 计算簇中心（即样本点在各维度上的均值）
    center = np.mean(cluster_points, axis=0)

    # 将簇中心添加到列表
```

```
    cluster_centers.append(center)

# 输出簇中心列表
print("簇中心点列表:")
print(cluster_centers)
#画出中心点
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
# plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', marker='x', s=200)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()



import pandas as pd
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf

#对 ts 进行白噪声检验
def white_noise_test(ts):
    from statsmodels.stats.diagnostic import acorr_ljungbox
    lb_test = acorr_ljungbox(ts, lags=10, return_df=True)
    print(lb_test)
    if lb_test['lb_pvalue'].all() > 0.05:
        print('The time series is white noise.')
    else:
        print('The time series is not white noise.')
        return False
    return True

#对 ts 进行 Box-Pierce's  混成检验
def box_pierce_test(ts):
    from statsmodels.stats.diagnostic import acorr_ljungbox
    bp_test = acorr_ljungbox(ts, lags=10, return_df=True)
    print(bp_test)
    if bp_test['lb_pvalue'].all() > 0.05:
        print('The time series is white noise.')
    else:
        print('The time series is not white noise.')
        return False
    return True
s = pd.read_csv('data2/2/B1_count.csv')
data = s.set_index('date')
# data.index = pd.to_datetime(data.index)
ts = data['count']
```

```
white_noise_test(ts)
box_pierce_test(ts)

acf = plot_acf(ts, lags=40)
acf.savefig('data2/img/acf.png',dpi=300)
#画出 PACF 图
pacf = plot_pacf(ts, lags=40)
pacf.savefig('data2/img/pacf.png',dpi=300)



import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 1. 读取数据
pl = pd.read_csv('data2/1/A4_speed.csv')

# 2. 计算各 vehicle_id 下 state 列的总和，并找出最大值及其对应的 vehicle_id
max_state_sum = pl.groupby('vehicle_id')['state'].sum().max()
max_vehicle_id = pl.groupby('vehicle_id')['state'].sum().idxmax()

# 3. 输出最大状态和与对应 vehicle_id
print(f"最大状态和为 {max_state_sum}，对应 vehicle_id 为 {max_vehicle_id}")

# 4. 绘制各 vehicle_id 下 state 列总和的分布图
fig, ax = plt.subplots(figsize=(10, 6))

# 设置字体大小
plt.rcParams['font.size'] = 14

# 使用 seaborn 绘制分布图，设置 bins=20
sns.distplot(pl.groupby('vehicle_id')['state'].sum(),bins=20)

# 5. 找出 state 总和最大的前 3 个 vehicle_id 的平均值
px = pl.groupby('vehicle_id')['state'].sum().nlargest(3).mean()

# 6. 在图上绘制一条竖线表示该平均值
plt.plot([px, px], [0, 0.007], 'r--', linewidth=2)

# 7. 在图上标注平均值
# 注意：这里原代码注释掉的部分使用了错误的 xytext 坐标，已修正
plt.annotate("预估红灯时间"+str(px+3)+'s', xy=(px, 0.013), xytext=(px-10, 0.015),c = 'r', fontsize=14)
```

```
# 8. 设置图表标签
plt.xlabel('时间')
plt.ylabel('频率')

# 9. 保存图表
plt.savefig('data2/img/车辆速度为 0 的等待时间频率图.png')



from scipy.fftpack import fft, fftfreq
import seaborn as sns
import matplotlib.pyplot as plt
def OBCSN_STFT(filename):
    df = pd.read_csv(filename)
    df['xy_distance'] = 0
    for i in df.groupby('vehicle_id'):
        df.loc[i[1].index,'xy_distance'] = np.sqrt((i[1].x.shift(-1)-i[1].x)**2+(i[1].y.shift(-1)-
i[1].y)**2)/1
        df.loc[i[1].index,'xy_distance'] = df.loc[i[1].index,'xy_distance'].fillna(11)
    df['state'] = np.where(df['xy_distance']<0.5,1,0)
    df['state_sum'] = df.groupby('time')['state'].transform('sum')

    grouped = df.groupby('vehicle_id')
    def find_special_rows(group):
        group['pp_state'] = group['state'].shift(2)
        group['prev_state'] = group['state'].shift(1)
        group['next_state'] = group['state'].shift(-1)
        condition = (group['pp_state'] == 1) & (group['state'] == 1) & (group['prev_state'] == 1) &
(group['next_state'] == 0)
        return group.loc[condition, 'time']
    special_times = grouped.apply(find_special_rows).dropna()
    special_times = np.array(special_times.values)
    special_times = np.sort(special_times)
    filtered_arr = [special_times[0]]
    for i in range(1, len(special_times)):
        if abs(special_times[i] - special_times[i - 1]) >= 25:
            filtered_arr = np.append(filtered_arr, special_times[i])

    s = df.groupby('time')['state'].sum()
    s = pd.DataFrame(s).reset_index()
    s.columns = ['date','count']
    data = s.set_index('date')
    ts = data['count']
```

```
    fft_series = fft(ts.values)
    power = np.abs(fft_series)
    sample_freq = fftfreq(fft_series.size)
    pos_mask = np.where(sample_freq > 0)
    freqs = sample_freq[pos_mask]
    powers = power[pos_mask]
    top_k_seasons = 3
    top_k_idxs = np.argpartition(powers, -top_k_seasons)[-top_k_seasons:]
    top_k_power = powers[top_k_idxs]
    fft_periods = (1 / freqs[top_k_idxs]).astype(int)

    print(f"top_k_power: {top_k_power}")
    print(f"fft_periods: {fft_periods}")

    NFFT = 256
    TEMP = np.abs(np.fft.fft(ts.values, n=NFFT))
    PSD = (TEMP[0:NFFT//2+1])**2 / NFFT
    frequencies = np.linspace(0, 0.95/2, NFFT//2+1)  # 单边频率轴，从 0 到采样率一半
    periods = 1./frequencies  # 将频率转换为周期
    fig, ax = plt.subplots()
    ax.semilogy(periods, PSD, linewidth=1.5)  # 使用对数尺度展示功率谱密度
    #标出 periods 范围为 50~100 间的对应 PSD 的最大值点
    max_index = np.argmax((periods >= 50) & (periods <= 100) & (PSD > 0))  # 需确保 PSD 大
于 0，避免选取到噪声
    ax.scatter(periods[max_index], PSD[max_index], color='red', marker='o', label=f"最大 PSD 点对
应周期为: {periods[max_index]:.2f}")
    # print(periods[np.argmax(PSD)], PSD[np.argmax(PSD)])
    # ax.scatter(periods[np.argmax(PSD)], PSD[np.argmax(PSD)], color='red', s=100)
    ax.set_xlabel('周期（秒）')
    ax.set_ylabel('PSD 值')
    ax.grid(True)
    ax.legend()
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    # plt.savefig('data2/img/STFT.png',dpi=200)
    plt.show()



    from statsmodels.tsa.stattools import acf

    for lag in fft_periods:
        acf_score = acf(ts.values, nlags=lag)[-1]
        print(f"lag: {lag} fft acf: {acf_score}")
```

```
    expected_lags = np.array([30, 60, 90]).astype(int)
    for lag in expected_lags:
        acf_score = acf(ts.values, nlags=lag, fft=False)[-1]
        print(f"lag: {lag} expected acf: {acf_score}")
    pl = df

print(pl.groupby('vehicle_id')['state'].sum().max(),pl.groupby('vehicle_id')['state'].sum().idxmax())

print(pl.groupby('vehicle_id')['state'].sum().nlargest(2).iloc[1],pl.groupby('vehicle_id')['state'].sum().nlargest(2).index[1])
    print(pl.groupby('vehicle_id')['state'].sum().nlargest(3).mean())
    sns.distplot(pl.groupby('vehicle_id')['state'].sum(),bins=20)

    return filtered_arr

l = OBCSN_STFT('data/1/A4.csv')
print(l)
print(np.diff(l))
```

附录 3

介绍：问题三的周期变换时刻以及新旧周期参数的全部结果

	C1	C2	C3	C4	C5	C6
周期 1 红灯时长（秒）	56	42	79	77	51	77
周期 1 绿灯时长（秒）	32	23	26	28	37	28
周期切换时刻（秒）	1494	246	1340	无	1318	无
周期 2 红灯时长（秒）	45	48	38		46	

周期 2 绿灯时长（秒）	16	28	23		24	
周期切换时刻（秒）	1670	398	1443		1670	
周期 3 红灯时长（秒）	56	67	79		51	
周期 3 绿灯时长（秒）	32	21	26		37	
周期切换时刻（秒）	2374	2092	2598		3342	
周期 4 红灯时长（秒）	45	48	38		38	
周期 4 绿灯时长（秒）	16	28	23		27	
周期切换时刻（秒）	2726	2245	6483		3606	
周期 5 红灯时长（秒）	56	67	52		51	
周期 5 绿灯时长（秒）	32	21	28		37	
周期切换时刻（秒）	2990	3325	6798		6510	
周期 6 红灯时长（秒）	45	42	79		30	
周期 6 绿灯时长（秒）	16	23	26		28	
周期切换时刻（秒）	3518	3500	无		无	
周期 7 红灯时长（秒）	56	67				
周期 7 绿灯时长（秒）	32	21				
周期切换时刻（秒）	6422	6228				
周期 8 红灯时长（秒）	45	48				
周期 8 绿灯时长（秒）	16	28				
周期切换时刻（秒）	6598	6381				
周期 9 红灯时长（秒）	56	42				
周期 9 绿灯时长（秒）	32	23				
周期切换时刻（秒）	无	7108				
周期 8 红灯时长（秒）		67				
周期 8 绿灯时长（秒）		21				
周期切换时刻（秒）		无				