**数学建模**

论文名称**:**  基于整数规划和SARIMA预测

的高速公路动态优化模型

2024年8月10日

基于整数规划和SARIMA预测的高速公路动态优化模型

**摘要**

近年来，随着我国高速公路交通流量大幅增加，为克服传统的全人工及半自动收费中存在的种种弊端，以不断提升高速公路通行效率和服务水平，电子收费系统(ETC)应运而生。该系统依托高速公路一定规模区域内密集、连续布设的门架系统，结合高速公路收费站等其他必要的数据资源，可以实现车辆行驶轨迹检测。

针对问题一，我们基于附件2作出了时间、门架和车辆通行数量三者关系的三维分布图，根据得到的图像，描述车流量的时空分布特征，也得到了维修门架的编号和维修的时间。

针对问题二，我们对附件2数据再次分析，发现的确存在门架漏失对过路车辆记录的情况。通过检索门架缺失记录，我们得到了各门架漏失过路车辆记录总数，进而可以求出各门架的漏失记录率。再对门架漏失过路车辆类型进行分析，得出了容易漏失记录的车辆的特征。

针对问题三，我们首先将每日收费站总流量和每日路段总流量进行对比，再将单位时间内通过收费站的车辆数和沿某一路段通过的车辆数量进行一一比较分析，作出体现两者关系的图像，进而综合分析收费站流量与路段流量的关系。然后，我们基于K-means聚类分析对各收费站进行分类，建立了单目标线性整数规划模型，在保证通行顺畅的前提下，确定了各站收费车道数以及应急收费系统（便携式收费机或移动电站收费车）的数量，实现费用的最省。

针对问题四，我们首先运用傅里叶变换，发现各路段车流量以一天为周期大致进行周期性变化，因此我们建立了季节性ARIMA预测模型，预测出各路段未来24小时的车流量。基于本问中得到的各路段未来24小时的车流量，建立了动态优化模型，并利用模拟退火算法对模型进行求解，在确保最大限度提高路网安全畅通的前提下，实现了实时优化未来24小时各个路段的限速值。

综上所述，本文利用基于聚类分析的单目标线性整数规划模型，确定了各站收费车道数以及应急收费系统的数量。建立了季节性ARIMA预测模型，预测出各路段未来24小时的车流量。使用基于模拟退火算法的动态优化模型，实现了实时优化未来24小时各个路段的限速值。

关键词：聚类分析；整数规划；SARIMA预测；动态优化模型；模拟退火

# **问题重述**

电子收费系统(ETC)广泛应用于高速公路车辆行驶轨迹检测，该系统包含了车辆视觉识别、天线、边缘计算服务单元等关键设备，主要依赖于高速公路一定规模区域内密集、连续布设的门架系统，结合高速公路收费站等其他必要的数据资源，实现车辆行驶轨迹检测。

**1.1问题一：**基于2022年2月22日至2月27日六天的数据，描述车流量的时空分布特征。并依据分布特征指出在规定时间范围内，运维部门维修门架的编号和维修的时间。

**1.2问题二**：判断门架是否会漏失对过路车辆的记录，如果会，指出各门架的漏失记录率和容易漏失记录的车辆的特征。

**1.3问题三**：说明收费站流量与路段流量的关系，并以在保证通行顺畅的前提下达到费用的最省为目的，确定各站收费车道数以及应急收费系统的数量。

**1.4问题四**：预测14个路段未来24小时各自的车流量，并依据预测模块实时优化未来24小时各个路段的限速值，以最大限度地确保路网安全畅通。

# **问题分析**

## 问题一的分析

针对问题一，基于附件2中2022年2月22日至2月27日的六天时间内经过该高速路区间的全部车辆的轨迹信息，由于完全基于所给数据进行统计分析，且数据量庞大，所以我们先对附件2的30个轨迹表进行数据预处理，发现并处理了异常数据。然后作出时间、门架和车辆通行数量三者关系的三维分布图，根据得到的图像，描述车流量的时空分布特征。在该段时间内，运维部门对一些ETC门架进行了维修，门架维修期间，记录到通过该门架的车辆数据信息显然会大幅降低。根据这个特点，结合三维分布图，可以得到维修门架的编号和维修的时间。

## 问题二的分析

针对问题二，我们首先将问题1中发现的附件2数据异常情况进行分类处理，发现的确存在门架漏失对过路车辆记录的情况。通过检索门架缺失记录，我们得到各门架漏失过路车辆记录总数，进而可以求得各门架的漏失记录率。再对门架漏失过路车辆类型进行分析，不难得到容易漏失记录的车辆的特征。

## 问题三的分析

针对问题三，我们首先将每日收费站总流量和每日路段总流量进行对比，为了进一步得到各收费站与各路段的关系，再将单位时间内通过收费站的车辆数和沿某一路段通过的车辆数量进行一一比较分析，作出体现两者关系的图像，进而综合分析收费站流量与路段流量的关系。然后，我们基于K-means聚类分析对各收费站进行分类，然后建立了单目标线性规划模型，以通行时间和总费用为目标函数，以各站收费车道数以及应急收费系统的数量，车辆经过收费车道用时及车辆经过应急收费系统用时为决策变量，进行整数线性规划，得到整数可行域，进而可以得到最优解，使得在保证通行顺畅的前提下达到费用的最省。

## 问题四的分析

针对问题四，我们首先运用傅里叶变换，发现各路段车流量以一天为周期大致进行周期性变化。由于每天各路段车流量大体具有周期性，我们建立了季节性ARIMA预测模型，对各路段未来24小时的车流量作出了预测。为实现实时优化未来24小时各个路段的限速值，我们基于本问中得到的各路段未来24小时的车流量，建立了动态优化模型，并利用模拟退火算法对模型进行求解。

# **模型假设**

1. 假设两相邻门架间车速异常对数据分析结果的影响可以忽略。
2. 假设不考虑车辆在进出记录时间段以外被门架漏失记录轨迹的情况。
3. 假设门架6在维修期间必然漏失的记录都修补为可以正常记录。
4. 假设所有车型通过ETC与MTC时间相同。
5. 假设车道造价与车道数成正比。
6. 假设不考虑驾驶员行为、天气状况等因素对交通流的影响

# **符号说明**

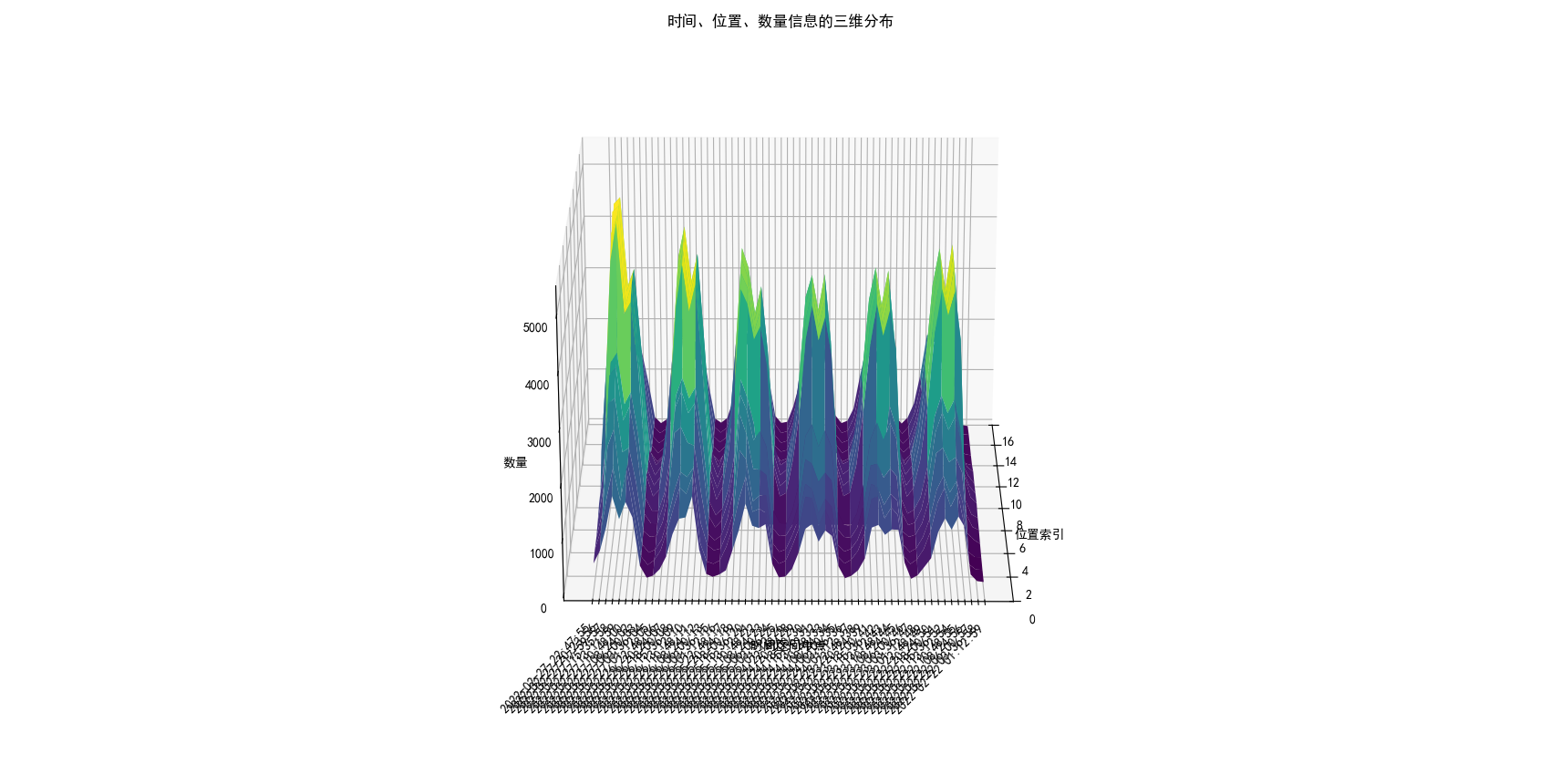
|  |  |
| --- | --- |
| 符号 | 说明 |
| latexmath | 第i号收费站修建车道及应急收费系统所用总费用 |
| latexmath | 第i号收费站修建的车道数 |
| latexmath | 第i号收费站修建的应急收费系统数 |
| latexmath | t时刻通过第i号收费站的车流量 |
| latexmath | 第i号收费站应急收费系统启动时间 |
| latexmath | 第i号收费站应急收费系统关闭时间 |
| latexmath | 第j个路段时间序列的全变动 |
| latexmath | 第j个路段长期趋势 |
| latexmath | 第j个路段季节变动 |
| latexmath | 第j个路段循环变动 |
| latexmath | 第j个路段不规则变动 |
| latexmath | 第j个路段t时刻的预测值 |
| latexmath | 第j个路段在t时刻的限速值 |
| latexmath | 第j个路段在t时刻的交通流量 |
| latexmath | 第j个路段在t时刻的平均车速 |
| latexmath | 第j个路段在t时刻的车辆间距 |
| latexmath | 第j个路段长度 |
|  |  |
|  |  |

# **模型的建立与求解**

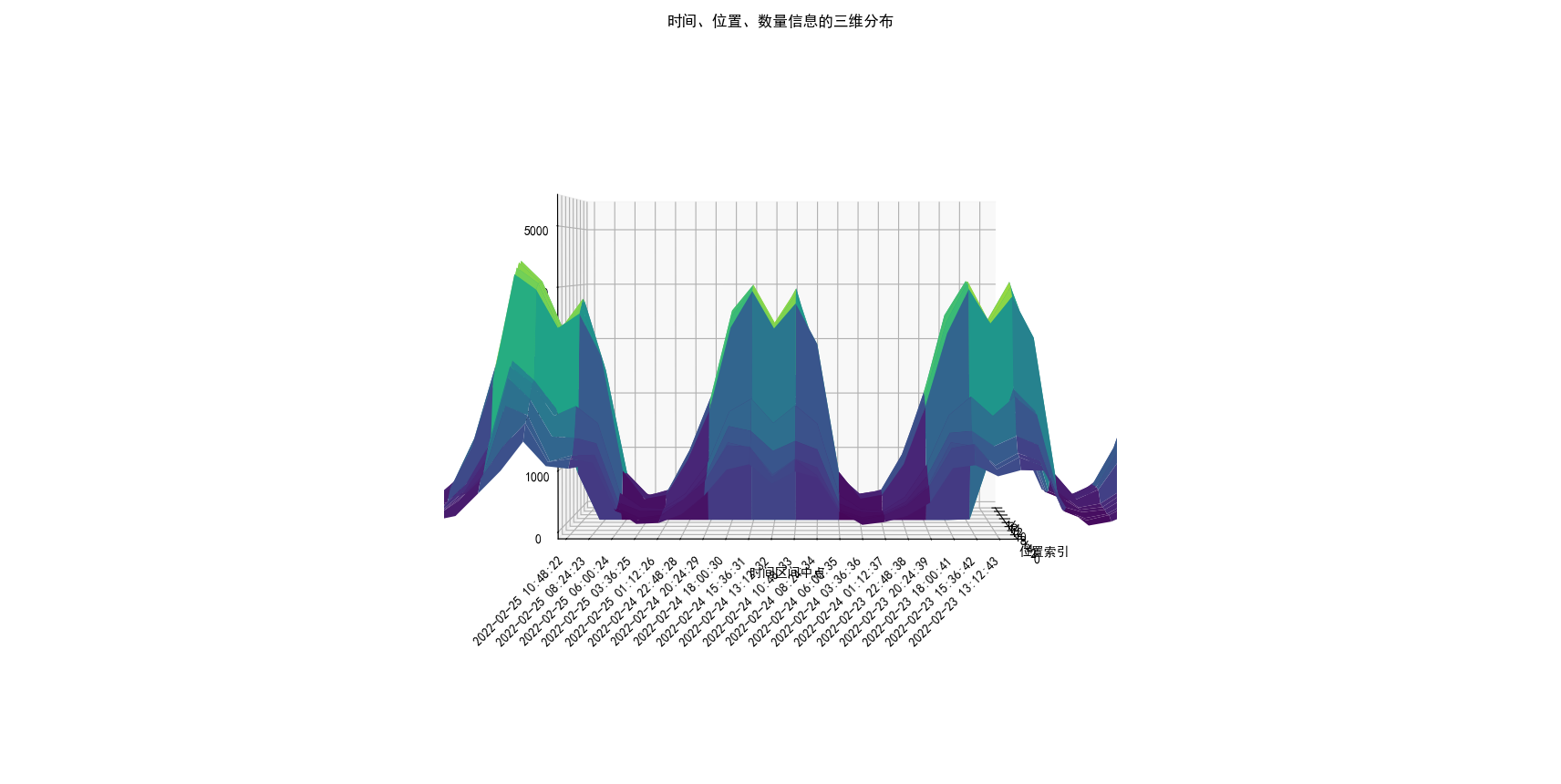
## 问题一模型的建立与求解

通过对附件2的数据检索，我们发现出现了极少数超出题目规定的分析时间段的轨迹记录以及出入站信息缺失的数据。由于30张轨迹表全体数据太过庞大，不妨以附件2轨迹表1为例，通过数据统计，总轨迹数为8646，而超出给定时间段的轨迹数为5条，出入站信息不完整的为5条，占总轨迹数的0.058%，对统计几乎没有影响。我们采用数据删除的方式，对符合题目条件的数据进行进一步处理。

在对附件2数据进一步观察中，我们发现极少数轨迹里出现门架记录漏失（在问题二中具体讨论）、通过门架编号顺序错乱、两相邻门架间车速计算异常的情况（误差时间短，不影响后续以小时为单位预测，在假设中已做处理）。针对门架编号顺序错乱，我们发现顺序颠倒的门架测量时间相差在3秒以内，粗略可以归为系统问题，直接进行顺序调换，归入两相邻门架间车速计算异常的情况，不影响分析。

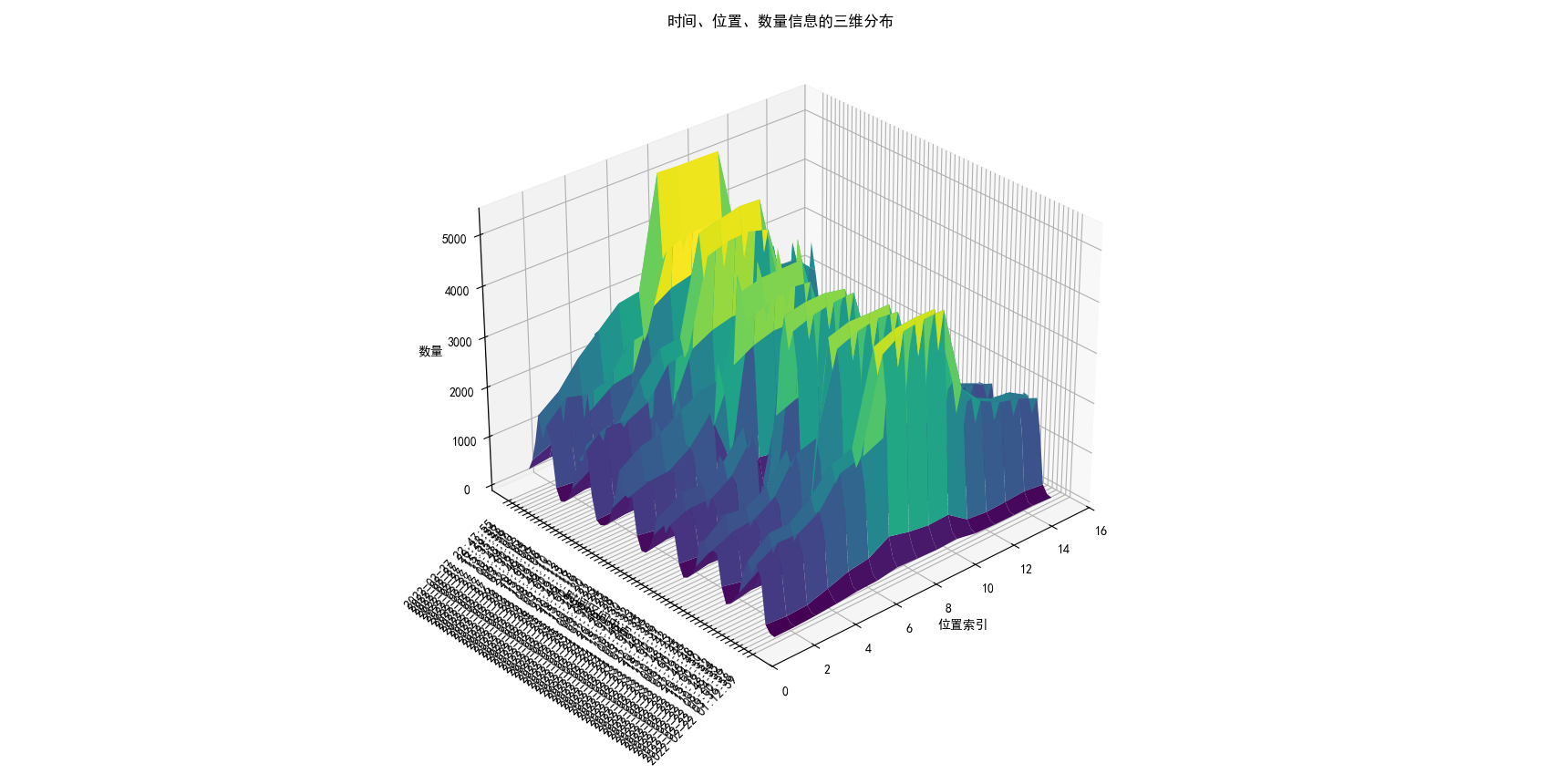


**图1 60等分时间区间各门架通行车辆数量（时间分布特征）**

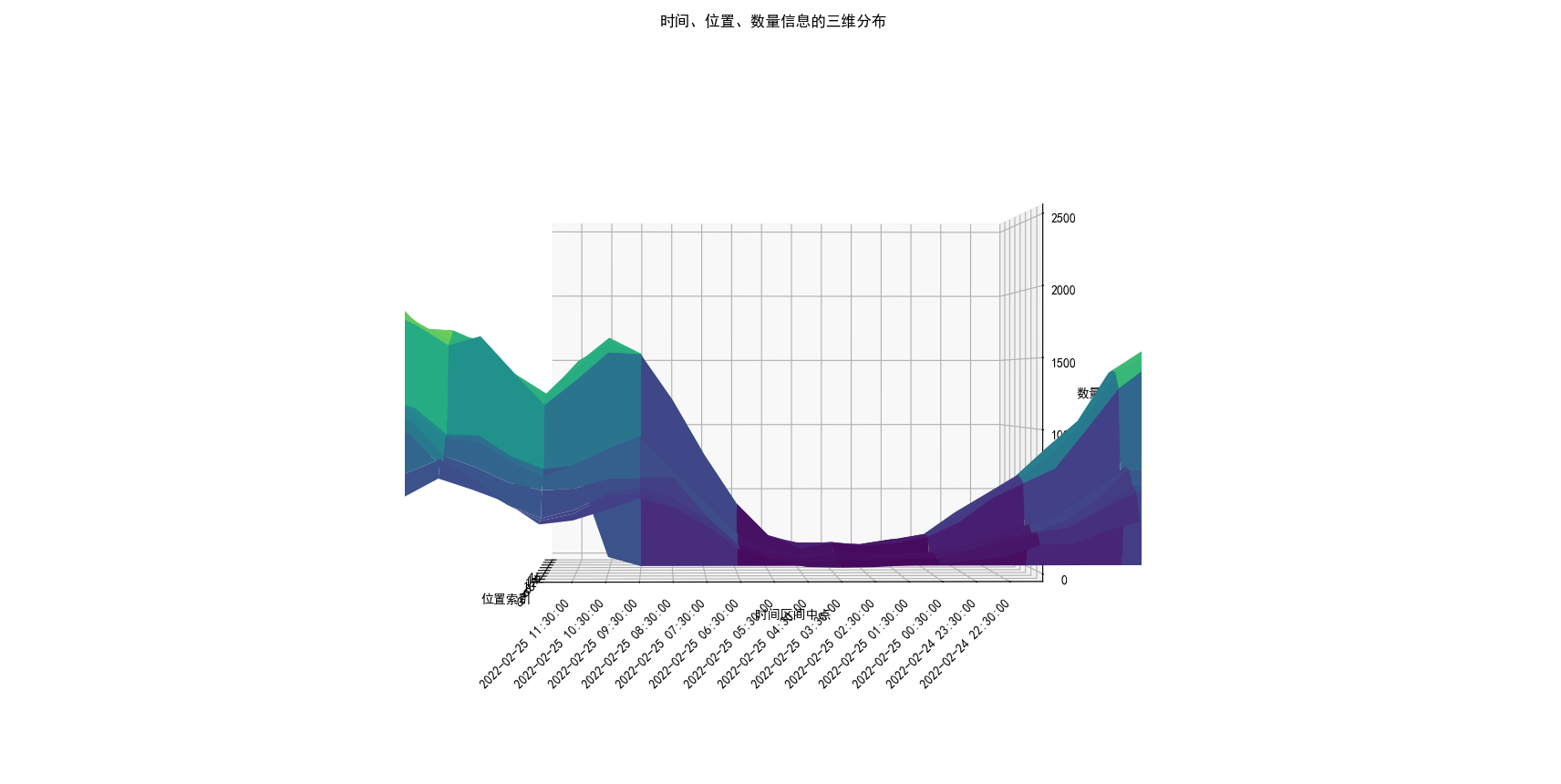
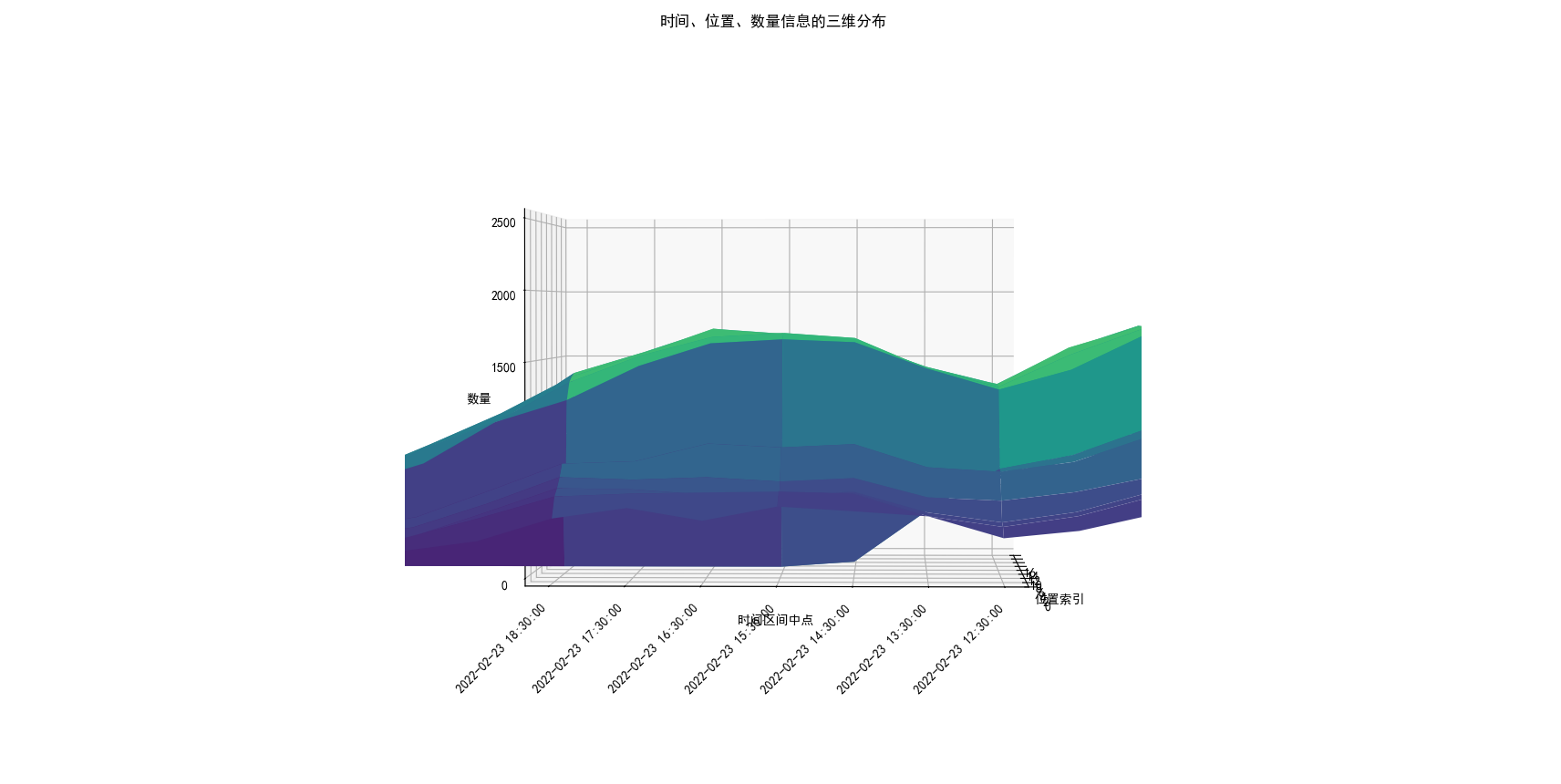


**图2 60等分时间区间各门架通行车辆数量局部放大图（时间分布特征）**

由图1可知，各门架通过车辆数量在时间分布上以24小时为一个最小周期进行循环增减。为得到更加精确的时间分布特征，我们对图1进行局部放大，得到了图2。由图2不难看出，在规定时间段内，每天各门架通过车辆数量在0时至10时逐渐增加，10时至13时会有小幅度下降，13时至15时会回升到10时水平甚至超越，15时至23时通过车辆数逐渐下降至当天0时水平。



**图3 60等分时间区间各门架通行车辆数量（空间分布特征）**

通过对图1进行方向变换得到图-3，我们可以以门架编号为主轴，观察车流量在空间上的分布特征。可以发现门架7、门架8、门架9、门架10相对来说通过的车辆最多。从图形上体现为靠近该路段中央，通过车辆较多；靠近该路段两端，通过车辆较少。 

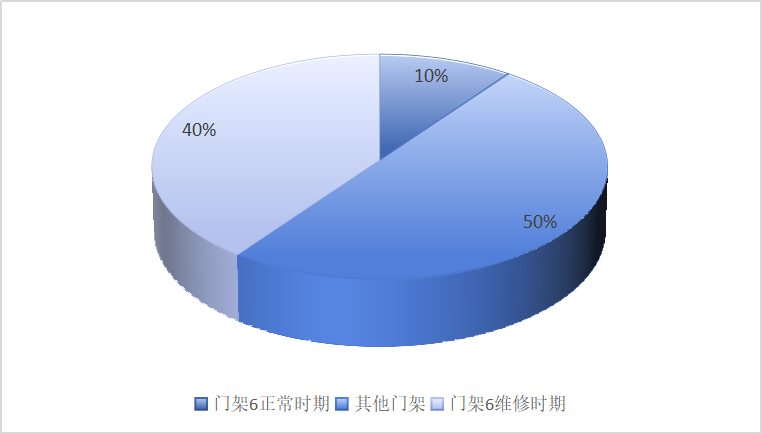
**图4 60等分时间区间门架维修特征图**

再次对图1进行方向变换和局部放大得到图4，我们由图4可以观察门架维修情况。可以看出，门架6通过车辆数在2月24日为零，推出门架6为维修门架。为得到更加准确的维修时间，我们结合图2，可以发现门架6的维修时间大致位于2022年2月23日14时至2022年2月25日9时。

## 问题二模型的建立与求解

我们对附件2中门架记录通过车辆数据再次进行检索，以轨迹表1为例，发现共有39983条有效数据，1250条数据被漏失。由此，我们发现部分门架确实会漏失对过路车辆的记录。

我们按上述方法对剩余29个轨迹表漏失数据进行组成结构分析，综合得到如下结果：



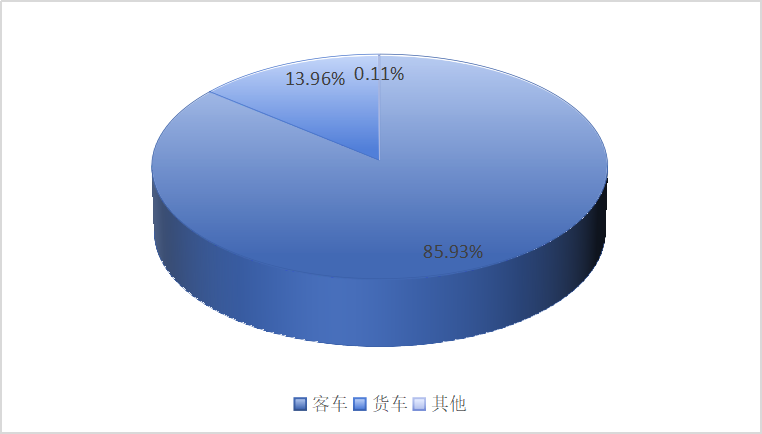
**图5 附件2轨迹表漏失数据构成图（总）**

由问题一的解答与图5进行分析，门架6在2月23日15时至2月25日8时左右会进行维修。由此，我们作出假设，门架6在维修期间必然漏失的记录都修补为可以正常记录。同时对于漏过的门架事件，时间通过相邻事件的时间推断，以此保证各门架漏失记录率的准确性。

各门架的漏失记录率如表1所示。

**表1 各门架漏失记录率**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 门架编号 | 漏失纪录率 | 门架编号 | 漏失纪录率 | 门架编号 | 漏失纪录率 |
| 门架1 | 0.0000% | 门架6 | 4.7692% | 门架11 | 0.9739% |
| 门架2 | 1.4348% | 门架7 | 0.6715% | 门架12 | 4.9906% |
| 门架3 | 0.5650% | 门架8 | 1.0999% | 门架13 | 5.0973% |
| 门架4 | 0.2089% | 门架9 | 2.0154% | 门架14 | 1.2244% |
| 门架5 | 0.7224% | 门架10 | 1.4499% | 门架15 | 0.7224% |
|  |  |  |  |  |  |

关于容易漏失记录的车辆特征的研究，我们作出了图6所示漏失车辆车型占比图。 

**图6 漏失车辆车型占比**

由图6可知，在所有漏失记录的车型中，客车占比大幅度高于货车和其他车型。可以看出，相较于其他车型，客车更容易漏失记录。

## 问题三模型的建立与求解

针对分析收费站流量与路段流量的关系，先对每日的收费站总流量与路段总流量进行对比。此处我们选择轨迹表1与轨迹表2为例，分析收费站流量与路段流量的关系，得到结果如表2所示：

**表2 每日收费站流量和每日路段流量统计**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 2月22日 | 2月23日 | 2月24日 | 2月25日 | 2月26日 | 2月27日 |
| 收费站流量 | 3445 | 3075 | 2989 | 3432 | 3340 | 3252 |
| 路段流量 | 14702 | 12506 | 11859 | 13698 | 13743 | 13635 |

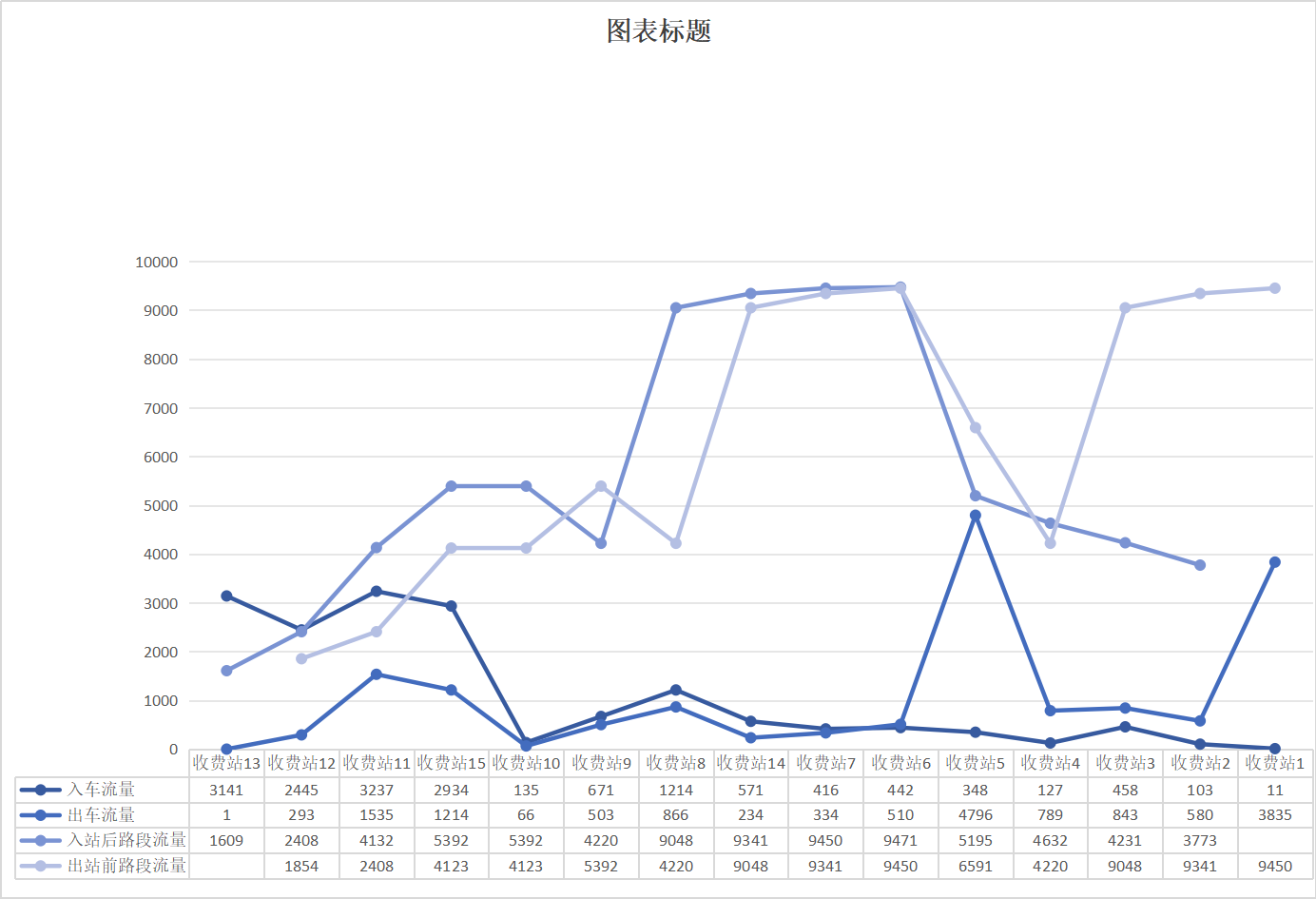
由表2可以看出，每日的收费站总流量与路段总流量成正相关。

基于上述结果，为了进一步得到各收费站与各路段的关系，我们根据附件1高速路段图中各收费站与各路段的相对位置关系，作出如表3对应关系，进行比较。

**表3 入出收费站与各路段比较关系对应表**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 入站收费站与路段 | | | | 出站收费站与路段 | | | |
| 收费站13 | 门架1 | 收费站14 | 门架8 | 收费站12 | 门架2 | 收费站7 | 门架8 |
| 收费站12 | 门架3 | 收费站7 | 门架9 | 收费站11 | 门架3 | 收费站6 | 门架9 |
| 收费站11 | 门架4 | 收费站6 | 门架10 | 收费站15 | 门架4 | 收费站5 | 门架11 |
| 收费站15 | 门架5 | 收费站5 | 门架12 | 收费站10 | 门架4 | 收费站4 | 门架12 |
| 收费站10 | 门架5 | 收费站4 | 门架13 | 收费站9 | 门架5 | 收费站3 | 门架13 |
| 收费站9 | 门架6 | 收费站3 | 门架14 | 收费站8 | 门架6 | 收费站2 | 门架14 |
| 收费站8 | 门架7 | 收费站2 | 门架15 | 收费站14 | 门架7 | 收费站1 | 门架15 |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

根据表3中的对应关系，我们利用Excel软件分析处理出入收费站与各路段流量的关系，并作出图7所示图像。



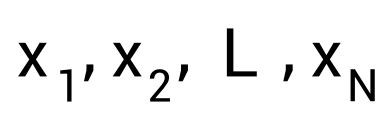
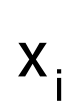
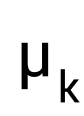
**图7 入出收费站与各路段流量折线图**

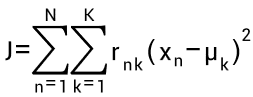
由图7可以看出，各收费站流量与对应路段流量大体成正比。收费站 8、11、12、13车流量较高，收费站14、3、15、5、1、7、 2车流量适中，收费站10、4、6、9车流量较低。收费站6、7、14、3、8对应路段流量较高，收费站5、2、4、9、10、15对应路段流量适中，收费站11、12、13对应路段流量较低。

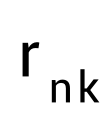
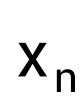
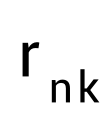
由于本文中涉及到的收费站车道数以及应急收费系统（便携式收费机或移动电站收费车）的数量均为整数，所以我们在此问选择基于K-means聚类分析的单目标线性整数规划模型。

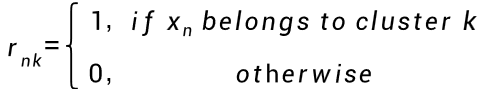
为便于计算，我们首先基于K-平均聚类算法对各收费站车流量进行聚类。

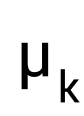
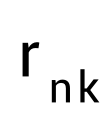
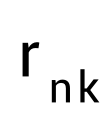
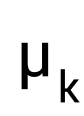
1. 平均聚类算法（K-means）是一种常用的无监督学习算法，用于将数据点分成不同的组或簇。其主要目标是将数据点分配到K个簇中，以使簇内对象具有较高相似度，而簇外相似度较低，其中相似度的计算根据一个簇中对象的平均值来进行。通过最小化数据点与其所属簇的中心点之间的距离的平方和来实现聚类。

我们假设存在N个数据点，每个数据点的维度为D。同时，假设要将数据点分成K个簇，每个簇有一个中心点，k=1,2,…,K。K均值的目标是最小化每个数据点与其所属簇的中心点之间的距离的平方和，即最小化如下损失函数：

 （1）

其中：是一个指示变量，当数据点属于第k个簇时为1，否则为0。通常情况下，可以表示为：

 （2）

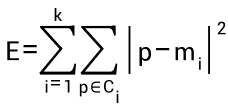
K均值算法的目标是找到最优的簇中心点以及相应的，使得损失函数J最小化。这通常通过迭代更新和来实现。K均值算法是一种迭代算法，其收敛性取决于初始簇中心点的选择以及数据的分布情况。

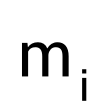
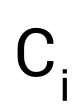
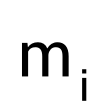
从n个数据对象中任意选择K个对象，作为初始聚类中心。

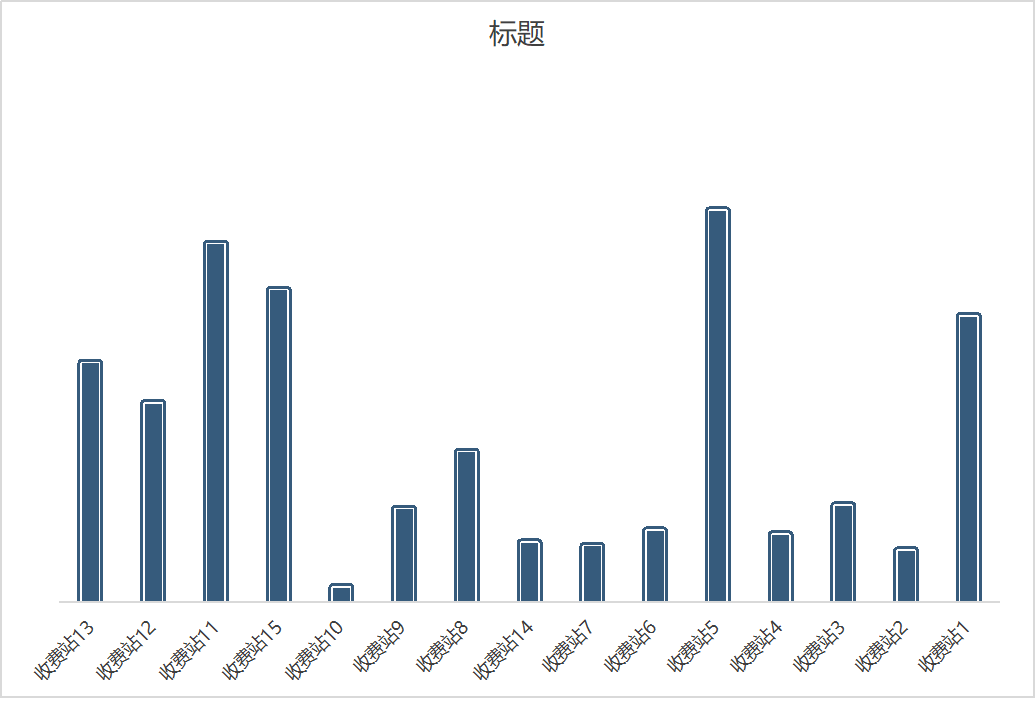
将每个数据点分配到离它最近的簇中心点所代表的簇。计算每个数据对象与各个簇中心点的距离，并由最小距离重新对相应对象进行划分，即将数据点分配到距离最近的簇中心点所代表的簇。

重新计算每个有变化的聚类的中心对象，即取该簇内所有数据点的平均值作为新的中心点。循环上述步骤，直到簇中心点不再发生变化或变化小于某个阈值。

通常选择均方差作为收敛准则函数：

 （3）

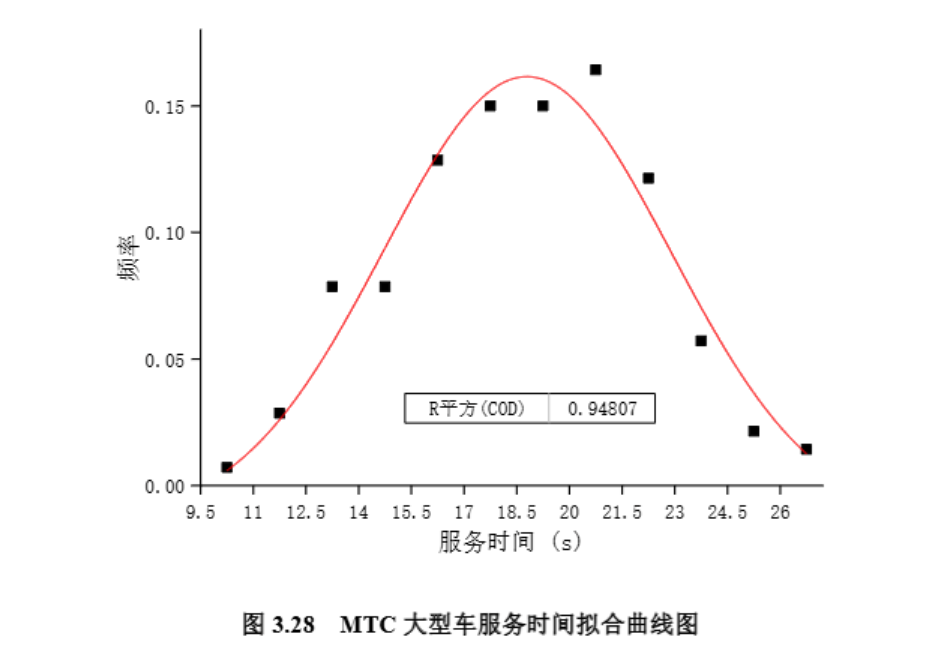
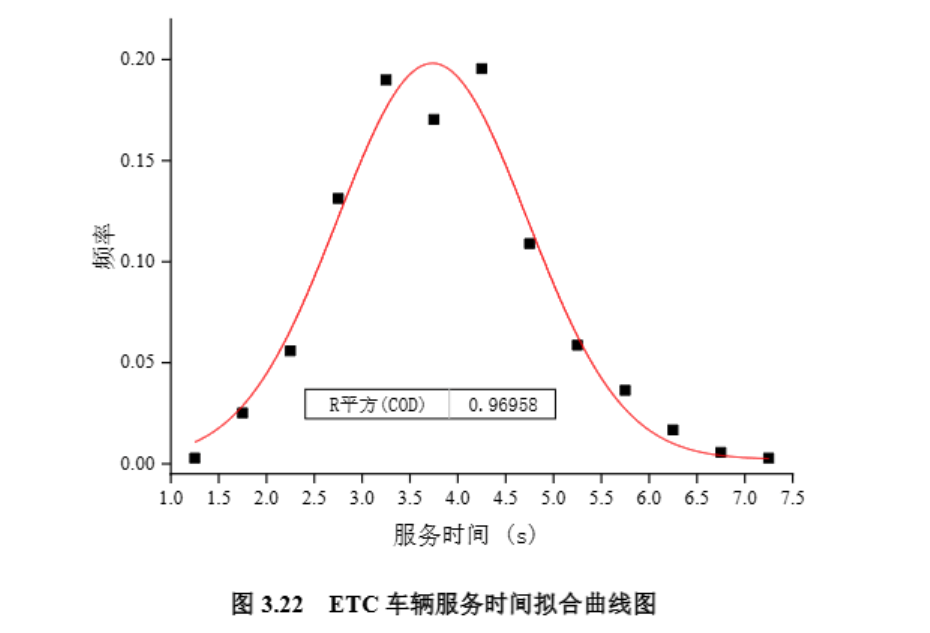
其中：E为数据库中所有对象的均方差之和；p为代表对象的空间中的一点；为聚类的均值。p和均是多维的。



**图8 各收费站出入车流量对比图**

我们通过聚类分析，依据图9所示各收费站出入车流量，将收费站分为3类：收费站5、11、15、1为第一类，收费站13、12、8为第二类，收费站9、3、6、4、14、7、2、10为第三类。

在具体计算中，我们认为所有车道都是ETC/MTC混合车道，不做具体区分。为了方便处理，我们通过大量数据得到ETC与MTC的车辆出入站比例为1：3，由此对两者通行效率进行加权，得到综合通行效率。



**图9 ETC通过车辆时间拟合曲线图 图10 MTC通过车辆时间拟合曲线图**

依据图7，图8的拟合曲线，我们设车辆ETC通过的时间为3.5s，MTC或补录通过时间为19s，代入模型中进行计算。

经过资料搜集与整理，我们粗略确定收费站车道每小时成本计算如下：

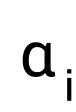
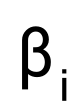
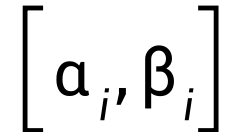
**单车道成本=单车道造价÷使用寿命（小时）≈25.8元**

单车道每小时能处理的车流量为：**3600÷（0.25×3.5+0.75×19）≈238辆。**

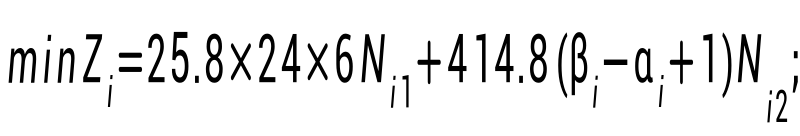
每套应急收费系统每小时成本计算如下：

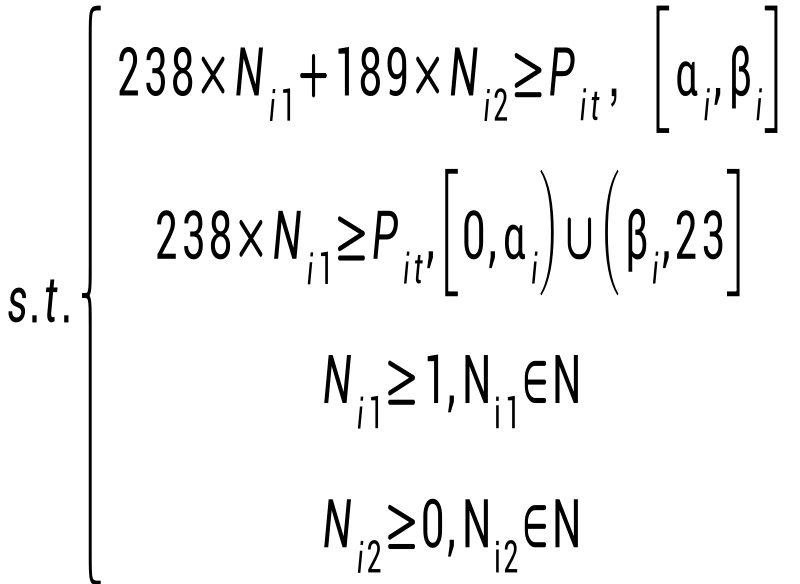
**应急收费系统成本=人员成本＋设备折旧＋能源消耗＋通讯费用＋维护修理+运输成本+保险成本≈414.8元**

每套应急收费系统每小时能处理的车流量为：**3600÷19≈189辆。**

通过查阅文献，我们了解到应急收费系统可以在车流量高峰时等效增加收费站车道数，即一个应急收费系统在车流量高峰期等价为一条MTC车道。由于应急收费系统并不是一天中每个小时都在工作，同时为了简化，我们以为第i个收费站应急收费系统每天开启的时间区间起点，以为时间区间终点，确定第i个收费站应急收费系统开启时间均为。

我们在对附件2进行数据处理时发现，车辆补录出站的时间未记录，又因不知道收费站具体位置，故我们采用经过的最后一个门架的时间代替其出站时间。

 （4）

 （5）

该模型接受某一收费站6天的逐小时车流量以及应急收费系统的开启时间区间，在此基础上尝试求得费用最优时，车道和应急收费系统的布局。在此基础上，再采取遍历所有的时间区间的方式，利用这一模型，寻找所有时间区间的最优费用中最优时的开启时间区间，从而得到针对该收费站的全局最优车道和应急收费系统的运营方案。

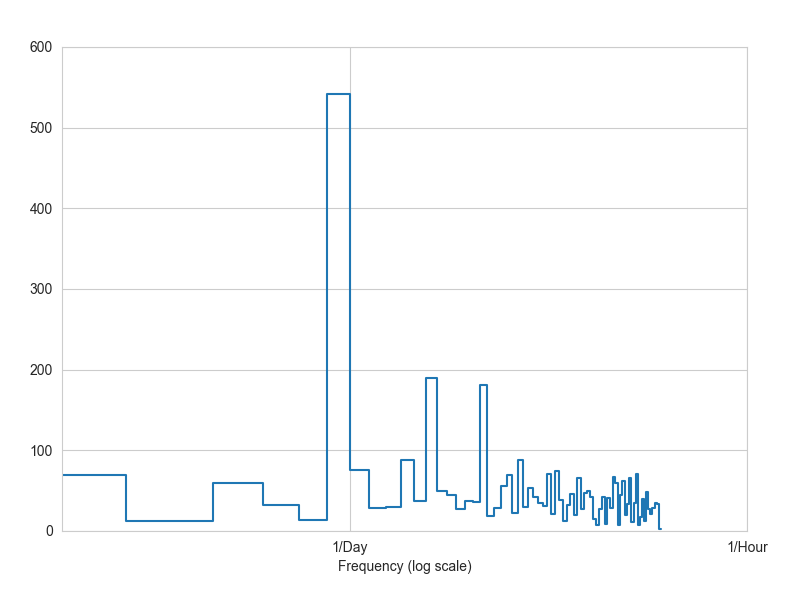
在对所有的收费站通行数据依次运用以上的方法后，我们得到了如下数据：

**表4 各站收费车道数以及应急收费系统数结果表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 收费站 | 车道数 | 应急收费系统数 | 应急系统开启 | 应急系统关闭 | 总费用/元 |
| 收费站13 | 4 | 1 | 15:00 | 16:00 | 14860.8 |
| 收费站12 | 4 | 1 | 14:00 | 17:00 | 15690.4 |
| 收费站11 | 6 | 1 | 16:00 | 17:00 | 22291.2 |
| 收费站15 | 5 | 1 | 8:00 | 10:00 | 18990.8 |
| 收费站10 | 1 | 0 | － | － | 3715.2 |
| 收费站9 | 2 | 0 | － | － | 7430.4 |
| 收费站8 | 3 | 1 | 9:00 | 18:00 | 14464 |
| 收费站14 | 2 | 0 | － | － | 7430.4 |
| 收费站7 | 2 | 0 | － | － | 7430.4 |
| 收费站6 | 1 | 1 | 8:00 | 17:00 | 7033.6 |
| 收费站5 | 6 | 2 | 9:00 | 10:00 | 22291.2 |
| 收费站4 | 2 | 0 | － | － | 7430.4 |
| 收费站3 | 2 | 1 | 10:00 | 11:00 | 7430.4 |
| 收费站2 | 1 | 1 | 15:00 | 19:00 | 4959.6 |
| 收费站1 | 4 | 1 | 10:00 | 18:00 | 17764.4 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

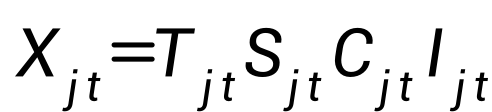
## 问题四模型的建立与求解

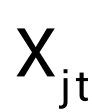
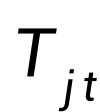
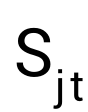
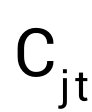
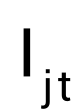
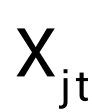
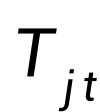
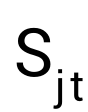
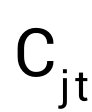
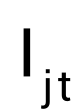
由问题一的解，以第一个路段车流量数据为例，进一步运用傅里叶变换得到如下结果：

****

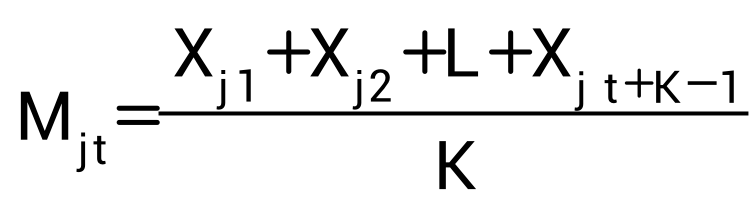
**图11 第1个路段车流量周期频率图**

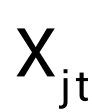
由图11可知，各路段车流量以一天为周期大致进行周期性变化，因此我们建立了季节性ARIMA预测模型，以预测各路段未来24小时的车流量。

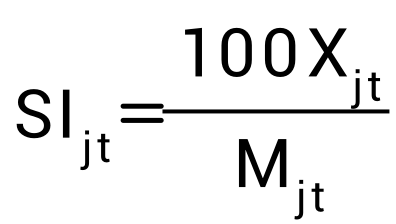
 （6）

式中为第j个路段时间序列的全变动，为第j个路段长期趋势，为第j个路段季节变动，为第j个路段循环变动，为第j个路段不规则变动。其中和用绝对数表示，，与用相对数（百分数）表示。

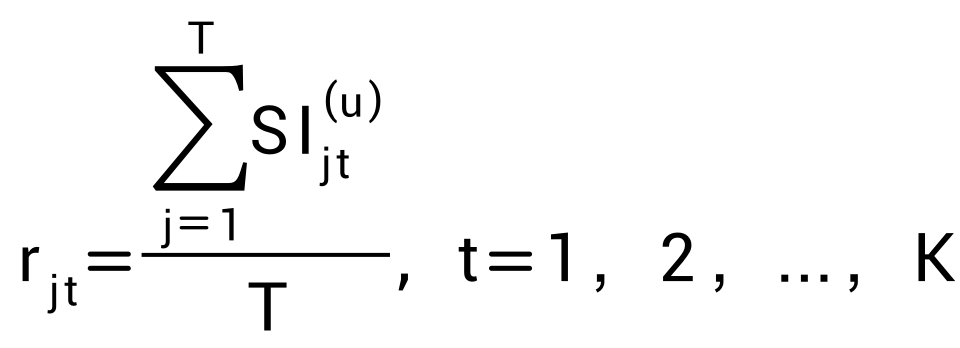
先对季节性S与随机性I求解：

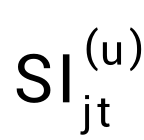
 （7）

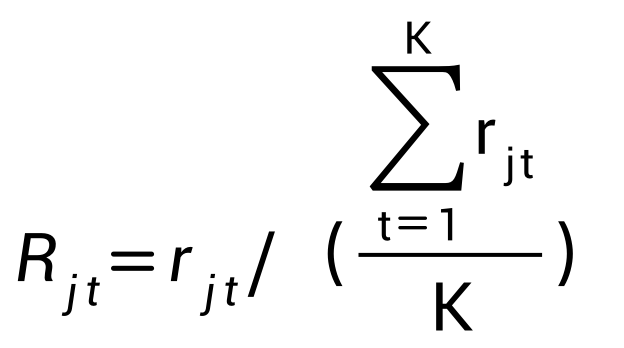
K为一个周期内的数据量，为第j个路段t时刻数据的观察值。

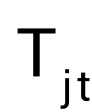
 （8）

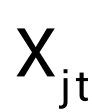
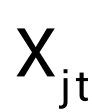
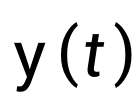
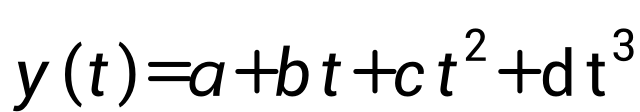
然后计算季节指数latexmath，计算各个周期数据点的平均值，相当从多个周期寻找一个周期内数据规律。

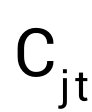
 （9）

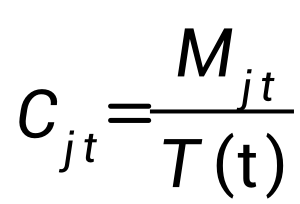
T为周期数量，为第j个路段第u个周期内第t个数据，设R为r修正后的数据，即为季节性和随机性：

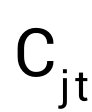
 （10）

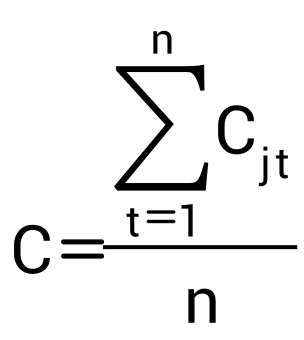
长期趋势的求解：

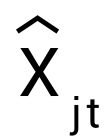
对数据进行拟合，根据整体趋势观察与t的整体关系，使用二次函数拟合，用表示拟合关系。可设，最后用最小二乘法求解出系数。

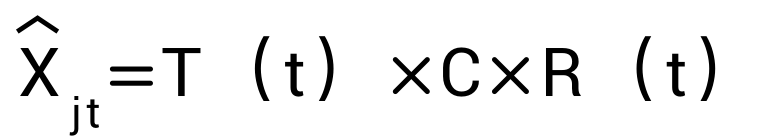
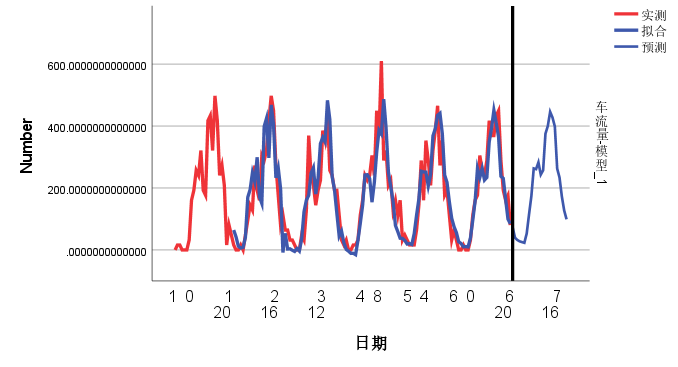
循环变动的求解：

 （11）

对求平均值可得到C：

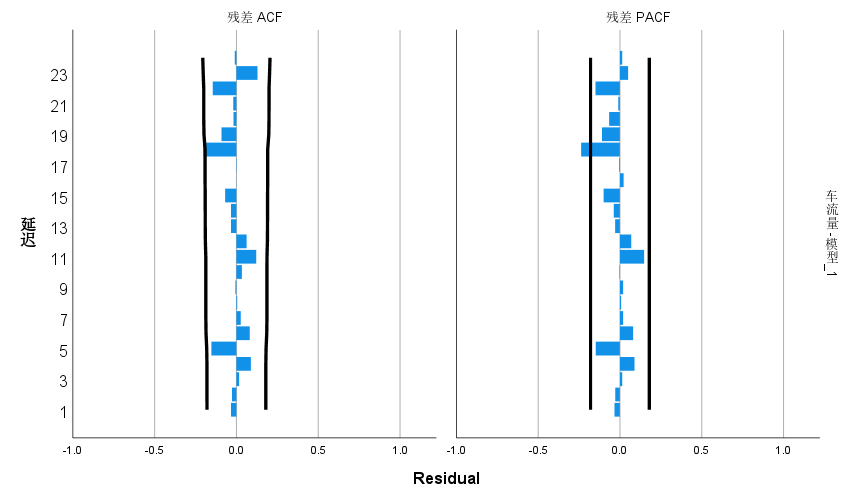
 （12）

设第j个路段t时刻的预测值为，则：

 （13）以路段1为例，用季节性ARIMA预测模型得到拟合曲线： 

**图12 路段1季节性ARIMA预测拟合曲线**

残差分析与自相关性检验如下，拟合度高，通过检验。

**图13 路段1ARIMA预测残差报告 图14 路段1ARIMA自相关性检验**

最终求得未来24小时的车流量具体数值如表5所示：

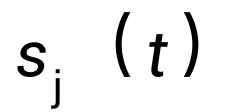
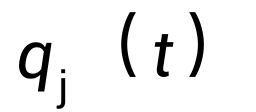
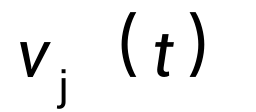
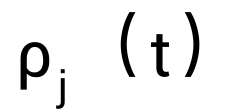
**表5 2月28日路段1各小时车流量预测**

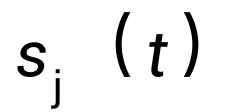
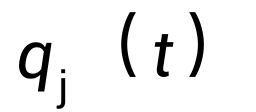
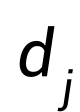
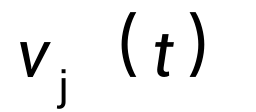
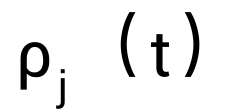
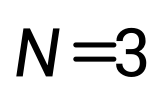
|  |  |  |  |
| --- | --- | --- | --- |
| 时段 | 车流量预测 | 时段 | 车流量预测 |
| 2022/2/28 0:00 | 73 | 2022/2/28 12:00 | 243 |
| 2022/2/28 1:00 | 40 | 2022/2/28 13:00 | 257 |
| 2022/2/28 2:00 | 32 | 2022/2/28 14:00 | 376 |
| 2022/2/28 3:00 | 28 | 2022/2/28 15:00 | 400 |
| 2022/2/28 4:00 | 25 | 2022/2/28 16:00 | 446 |
| 2022/2/28 5:00 | 23 | 2022/2/28 17:00 | 428 |
| 2022/2/28 6:00 | 53 | 2022/2/28 18:00 | 399 |
| 2022/2/28 7:00 | 115 | 2022/2/28 19:00 | 263 |
| 2022/2/28 8:00 | 174 | 2022/2/28 20:00 | 234 |
| 2022/2/28 9:00 | 263 | 2022/2/28 21:00 | 173 |
| 2022/2/28 10:00 | 261 | 2022/2/28 22:00 | 127 |
| 2022/2/28 11:00 | 283 | 2022/2/28 23:00 | 99 |

由于数据量较大，其余路段未来24小时车流量预测见附录。

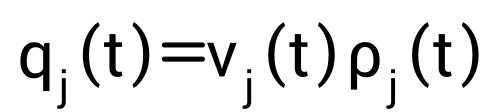
交通流时空分布不均衡即交通流量在时间和空间上呈现出非均匀分布的现象。在时间分布上，早晚高峰时段是交通流量最大的时期，导致道路拥堵、车速下降。而在平峰时段，交通流量相对较小，道路状况较为畅通。在空间分布上，不同路段的交通流量存在显著差异。

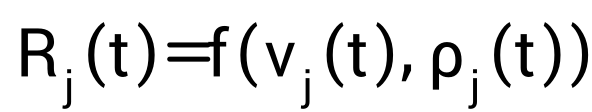
为了实时优化未来24小时各个路段的限速值，我们建立了动态优化模型。

确定决策变量为，状态变量、、，其中：

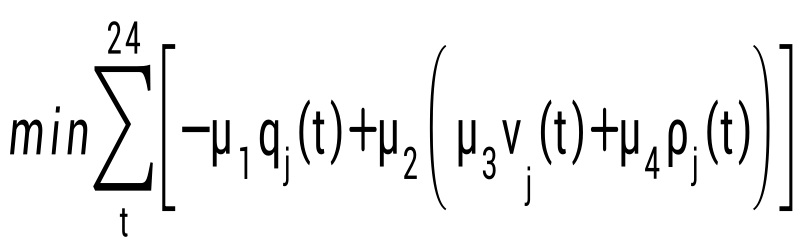
为第j个路段在t时刻的限速值,为第j个路段在t时刻的交通流量，为第j个路段的长度，为第j个路段在t时刻的平均车速，为第j个路段在t时刻的车辆间距，为高速路车道数。

为了最大限度提高路网安全畅通，我们引入两个局部目标函数：

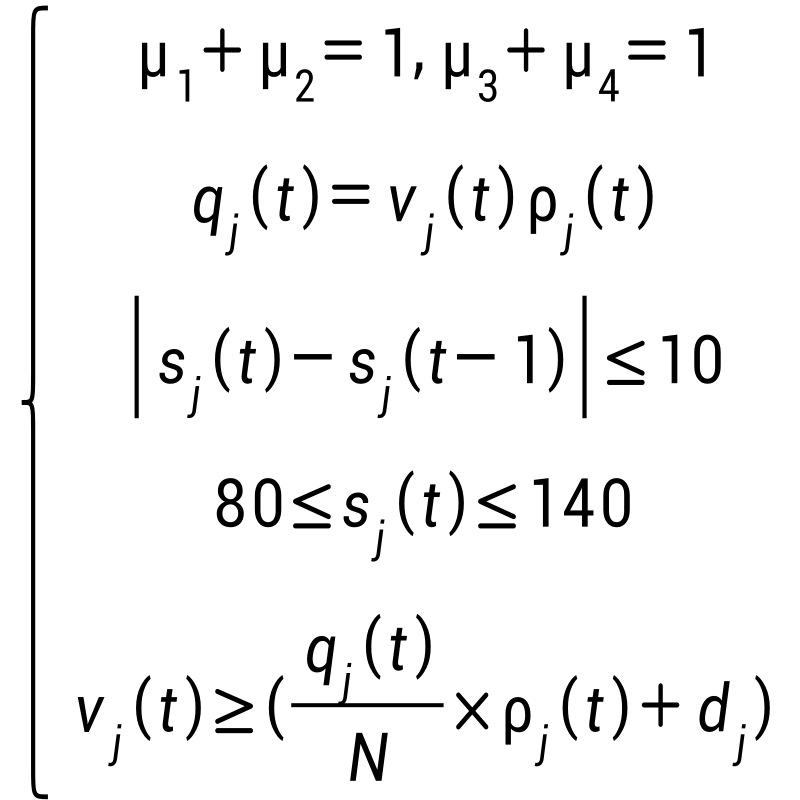
最大化通行能力：  （14）

最小化交通事故风险:  （15）

综合构建目标函数：

 （16）

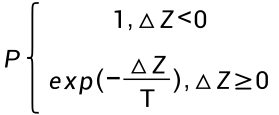
约束条件如下：

 （17）

本问我们选择了基于模拟退火算法的动态优化模型求解。该模型是一种随机搜 索方法，来源于物理中的退火过程。算法从一个初始解（或初始温度）开始，通过随机搜索邻域状态来寻找更好的解。在搜索过程中，算法会根据 Metropolis 准则，以一定概率接受邻域解作为新的当前解，这有助于让算法跳出局部最优解的局限，从而去求解全局最优解。随着温度的逐渐降低，算法接受较差解的概率也会逐渐减小，最终收敛到一个全局最优解或近似全局最优解。

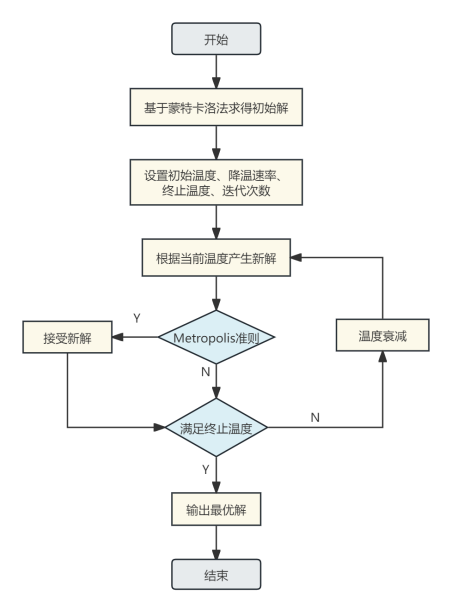
令 。

我们依据Metropolis准则判断是否接受新解：

 （18）

其中，T为当前温度。

模拟退火的流程图如下所示：



**图15 模拟退火流程图**

其中，基于蒙特卡罗法求初始解我们直接使用Python中的Random库求得。

由于结果数据体量较大，此处以路段7为例，其余路段限速见附录，路段7未来24小时各个路段的限速值结果如下：

**表6 路段7未来24小时各个路段的限速值**

|  |  |  |  |
| --- | --- | --- | --- |
| 时间 | 路段7限速 | 时间 | 路段7限速 |
| 2022/2/28 0:00 | 80 | 2022/2/28 12:00 | 110 |
| 2022/2/28 1:00 | 80 | 2022/2/28 13:00 | 115 |
| 2022/2/28 2:00 | 80 | 2022/2/28 14:00 | 120 |
| 2022/2/28 3:00 | 80 | 2022/2/28 15:00 | 115 |
| 2022/2/28 4:00 | 90 | 2022/2/28 16:00 | 120 |
| 2022/2/28 5:00 | 100 | 2022/2/28 17:00 | 115 |
| 2022/2/28 6:00 | 110 | 2022/2/28 18:00 | 105 |
| 2022/2/28 7:00 | 120 | 2022/2/28 19:00 | 95 |
| 2022/2/28 8:00 | 120 | 2022/2/28 20:00 | 85 |
| 2022/2/28 9:00 | 120 | 2022/2/28 21:00 | 80 |
| 2022/2/28 10:00 | 120 | 2022/2/28 22:00 | 80 |
|  |  |  |  |

# **模型的优缺点与改进**

## 模型优点评价

在问题三中，我们首先基于K-means聚类分析对各收费站进行分类，然后建立了单目标线性整数规划模型。K-means聚类分析具有适用性广和可拓展性强等优点，单目标线性整数规划模型可以直接解决整数约束问题，还具有明确性和灵活性等好处。

在问题四中，我们建立了季节性ARIMA预测模型，对各路段未来24小时的车流量作出了预测。季节性ARIMA预测模型能够捕捉季节性因素，同时预测能力较强。我们基于本问中得到的各路段未来24小时的车流量，建立了动态优化模型，并利用模拟退火算法对模型进行求解。动态优化模型引入了时间因素，提高了决策效率，可以实现全局优化。

## 模型缺点评价

单目标线性整数规划模型可行域限制性强，数值稳定性有时不强，同时在进行参数调整时比较困难。

季节性ARIMA预测模型在参数选择上较为复杂，对数据要求较为严格，对异常值比较敏感。

动态优化模型参数难以确定，模型稳定性和鲁棒性较低。尽管动态优化模型旨在解决随时间变化的优化问题，但其对系统动态变化的捕捉能力仍然有限。特别是在系统动态变化非常复杂或不可预测的情况下，模型可能无法准确反映系统的实际运行状况，从而影响优化结果的有效性。

## 模型改进

可能的改进方案首先是可将SARIMA模型与其他模型（如神经网络、支持向量机等）结合使用，形成组合模型。这种组合模型可以充分利用不同模型的优点，提高预测的准确性。其次，可以对动态优化模型的求解算法进行改进，如引入启发式搜索、遗传算法、粒子群优化等智能优化算法，以提高求解效率和精度。同时，可以尝试建立反馈机制，将模型的预测结果与实际观测值进行比较，并根据误差大小调整模型的参数或结构。还可以考虑使用交通仿真软件来评估不同限速策略的效果。

# 参考文献

[1]常志恒. ETC推广对高速公路交通排放的减碳潜力评估[D].北京工业大学,2023.DOI:10.26935/d.cnki.gbjgu.2023.000274.

[2]苏帅杰. 基于推广ETC政策下的高速公路收费站通行能力和车道配置研究[D].长安大学,2021.DOI:10.26976/d.cnki.gchau.2020.001185.

[3]黄世彬.基于ETC数据的高速公路在途车辆路径预测研究[D].福建工程学院,2023.DOI:10.27865/d.cnki.gfgxy.2023.000240.

[4] 黄志敏,梁承东.基于K-means聚类算法的等级测评数据分析[J].电子质量,2023,(12):40-44.

[5]孙晶.高速公路收费站车道配置与建设成本对比分析[J].黑龙江交通科技, 2019, 42(6):4.DOI:CNKI:SUN:HLJJ.0.2019-06-093.

[6]王薇,杨兆升,赵丁选.有限阶段马尔可夫决策的可变限速控制模型[J].交通运输工程报,2011,11(05):109-114.DOI:10.19818/j.cnki.1671-1637.2011.05.017.

[7]张乐飞.高速公路瓶颈路段可变限速控制方法研究[J].北京交通大学学报,2012,36(05):78-83.

[8]谭学武,邓木生,彭思程,等.基于模拟退火算法的模型预测控制策略[J].电工电气,2023,(09):14-19.

附录

问题四中其余13个路段未来24小时车流量预测如下。

**附表1 2月28日路段2各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 73 |
| 2022/2/28 1:00 | 68 |
| 2022/2/28 2:00 | 42 |
| 2022/2/28 3:00 | 23 |
| 2022/2/28 4:00 | 19 |
| 2022/2/28 5:00 | 21 |
| 2022/2/28 6:00 | 35 |
| 2022/2/28 7:00 | 62 |
| 2022/2/28 8:00 | 171 |
| 2022/2/28 9:00 | 226 |
| 2022/2/28 10:00 | 326 |
| 2022/2/28 11:00 | 250 |
| 2022/2/28 12:00 | 286 |
| 2022/2/28 13:00 | 227 |
| 2022/2/28 14:00 | 365 |
| 2022/2/28 15:00 | 384 |
| 2022/2/28 16:00 | 519 |
| 2022/2/28 17:00 | 413 |
| 2022/2/28 18:00 | 534 |
| 2022/2/28 19:00 | 342 |
| 2022/2/28 20:00 | 255 |
| 2022/2/28 21:00 | 246 |
| 2022/2/28 22:00 | 148 |
| 2022/2/28 23:00 | 145 |
|  |  |
|  |  |

**附表2 2月28日路段3各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 151 |
| 2022/2/28 1:00 | 119 |
| 2022/2/28 2:00 | 114 |
| 2022/2/28 3:00 | 91 |
| 2022/2/28 4:00 | 96 |
| 2022/2/28 5:00 | 105 |
| 2022/2/28 6:00 | 150 |
| 2022/2/28 7:00 | 179 |
| 2022/2/28 8:00 | 274 |
| 2022/2/28 9:00 | 345 |
| 2022/2/28 10:00 | 492 |
| 2022/2/28 11:00 | 421 |
| 2022/2/28 12:00 | 449 |
| 2022/2/28 13:00 | 362 |
| 2022/2/28 14:00 | 628 |
| 2022/2/28 15:00 | 680 |
| 2022/2/28 16:00 | 940 |
| 2022/2/28 17:00 | 788 |
| 2022/2/28 18:00 | 783 |
| 2022/2/28 19:00 | 492 |
| 2022/2/28 20:00 | 358 |
| 2022/2/28 21:00 | 360 |
| 2022/2/28 22:00 | 264 |
| 2022/2/28 23:00 | 231 |
|  |  |
|  |  |

**附表3 2月28日路段4各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 127 |
| 2022/2/28 1:00 | 64 |
| 2022/2/28 2:00 | 110 |
| 2022/2/28 3:00 | 92 |
| 2022/2/28 4:00 | 91 |
| 2022/2/28 5:00 | 109 |
| 2022/2/28 6:00 | 242 |
| 2022/2/28 7:00 | 467 |
| 2022/2/28 8:00 | 668 |
| 2022/2/28 9:00 | 732 |
| 2022/2/28 10:00 | 820 |
| 2022/2/28 11:00 | 698 |
| 2022/2/28 12:00 | 592 |
| 2022/2/28 13:00 | 855 |
| 2022/2/28 14:00 | 868 |
| 2022/2/28 15:00 | 1,139 |
| 2022/2/28 16:00 | 1,182 |
| 2022/2/28 17:00 | 1,384 |
| 2022/2/28 18:00 | 1,280 |
| 2022/2/28 19:00 | 709 |
| 2022/2/28 20:00 | 512 |
| 2022/2/28 21:00 | 466 |
| 2022/2/28 22:00 | 318 |
| 2022/2/28 23:00 | 212 |
|  |  |
|  |  |

**附表4 2月28日路段5各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 112 |
| 2022/2/28 1:00 | 49 |
| 2022/2/28 2:00 | 66 |
| 2022/2/28 3:00 | 44 |
| 2022/2/28 4:00 | 52 |
| 2022/2/28 5:00 | 48 |
| 2022/2/28 6:00 | 312 |
| 2022/2/28 7:00 | 589 |
| 2022/2/28 8:00 | 865 |
| 2022/2/28 9:00 | 1,184 |
| 2022/2/28 10:00 | 1,259 |
| 2022/2/28 11:00 | 932 |
| 2022/2/28 12:00 | 648 |
| 2022/2/28 13:00 | 868 |
| 2022/2/28 14:00 | 1,097 |
| 2022/2/28 15:00 | 1,045 |
| 2022/2/28 16:00 | 1,145 |
| 2022/2/28 17:00 | 1,192 |
| 2022/2/28 18:00 | 1,038 |
| 2022/2/28 19:00 | 754 |
| 2022/2/28 20:00 | 430 |
| 2022/2/28 21:00 | 350 |
| 2022/2/28 22:00 | 215 |
| 2022/2/28 23:00 | 135 |
|  |  |
|  |  |

**附表5 2月28日路段6各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 194 |
| 2022/2/28 1:00 | 161 |
| 2022/2/28 2:00 | 110 |
| 2022/2/28 3:00 | 98 |
| 2022/2/28 4:00 | 150 |
| 2022/2/28 5:00 | 186 |
| 2022/2/28 6:00 | 452 |
| 2022/2/28 7:00 | 793 |
| 2022/2/28 8:00 | 981 |
| 2022/2/28 9:00 | 1,308 |
| 2022/2/28 10:00 | 1,474 |
| 2022/2/28 11:00 | 1,076 |
| 2022/2/28 12:00 | 788 |
| 2022/2/28 13:00 | 1,007 |
| 2022/2/28 14:00 | 1,282 |
| 2022/2/28 15:00 | 1,191 |
| 2022/2/28 16:00 | 1,295 |
| 2022/2/28 17:00 | 1,280 |
| 2022/2/28 18:00 | 1,158 |
| 2022/2/28 19:00 | 872 |
| 2022/2/28 20:00 | 547 |
| 2022/2/28 21:00 | 428 |
| 2022/2/28 22:00 | 280 |
| 2022/2/28 23:00 | 216 |
|  |  |
|  |  |

**附表6 2月28日路段7各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 211 |
| 2022/2/28 1:00 | 189 |
| 2022/2/28 2:00 | 185 |
| 2022/2/28 3:00 | 174 |
| 2022/2/28 4:00 | 218 |
| 2022/2/28 5:00 | 291 |
| 2022/2/28 6:00 | 628 |
| 2022/2/28 7:00 | 1,300 |
| 2022/2/28 8:00 | 1,708 |
| 2022/2/28 9:00 | 2,431 |
| 2022/2/28 10:00 | 2,220 |
| 2022/2/28 11:00 | 1,677 |
| 2022/2/28 12:00 | 1,430 |
| 2022/2/28 13:00 | 1,484 |
| 2022/2/28 14:00 | 2,061 |
| 2022/2/28 15:00 | 1,907 |
| 2022/2/28 16:00 | 1,963 |
| 2022/2/28 17:00 | 1,946 |
| 2022/2/28 18:00 | 1,686 |
| 2022/2/28 19:00 | 1,348 |
| 2022/2/28 20:00 | 956 |
| 2022/2/28 21:00 | 788 |
| 2022/2/28 22:00 | 545 |
| 2022/2/28 23:00 | 405 |
|  |  |
|  |  |

**附表7 2月28日路段8各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 224 |
| 2022/2/28 1:00 | 251 |
| 2022/2/28 2:00 | 177 |
| 2022/2/28 3:00 | 213 |
| 2022/2/28 4:00 | 220 |
| 2022/2/28 5:00 | 291 |
| 2022/2/28 6:00 | 572 |
| 2022/2/28 7:00 | 1,160 |
| 2022/2/28 8:00 | 1,863 |
| 2022/2/28 9:00 | 2,494 |
| 2022/2/28 10:00 | 2,470 |
| 2022/2/28 11:00 | 1,698 |
| 2022/2/28 12:00 | 1,540 |
| 2022/2/28 13:00 | 1,543 |
| 2022/2/28 14:00 | 2,062 |
| 2022/2/28 15:00 | 1,977 |
| 2022/2/28 16:00 | 1,958 |
| 2022/2/28 17:00 | 1,871 |
| 2022/2/28 18:00 | 1,784 |
| 2022/2/28 19:00 | 1,414 |
| 2022/2/28 20:00 | 1,098 |
| 2022/2/28 21:00 | 845 |
| 2022/2/28 22:00 | 617 |
| 2022/2/28 23:00 | 515 |
|  |  |
|  |  |

**附表8 2月28日路段9各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 290 |
| 2022/2/28 1:00 | 244 |
| 2022/2/28 2:00 | 206 |
| 2022/2/28 3:00 | 191 |
| 2022/2/28 4:00 | 237 |
| 2022/2/28 5:00 | 251 |
| 2022/2/28 6:00 | 514 |
| 2022/2/28 7:00 | 1,087 |
| 2022/2/28 8:00 | 1,795 |
| 2022/2/28 9:00 | 2,489 |
| 2022/2/28 10:00 | 2,561 |
| 2022/2/28 11:00 | 1,739 |
| 2022/2/28 12:00 | 1,510 |
| 2022/2/28 13:00 | 1,475 |
| 2022/2/28 14:00 | 1,979 |
| 2022/2/28 15:00 | 1,984 |
| 2022/2/28 16:00 | 1,914 |
| 2022/2/28 17:00 | 1,817 |
| 2022/2/28 18:00 | 1,860 |
| 2022/2/28 19:00 | 1,371 |
| 2022/2/28 20:00 | 1,068 |
| 2022/2/28 21:00 | 766 |
| 2022/2/28 22:00 | 640 |
| 2022/2/28 23:00 | 434 |
|  |  |
|  |  |

**附表9 2月28日路段10各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 319 |
| 2022/2/28 1:00 | 230 |
| 2022/2/28 2:00 | 227 |
| 2022/2/28 3:00 | 209 |
| 2022/2/28 4:00 | 280 |
| 2022/2/28 5:00 | 285 |
| 2022/2/28 6:00 | 497 |
| 2022/2/28 7:00 | 1,047 |
| 2022/2/28 8:00 | 1,826 |
| 2022/2/28 9:00 | 2,449 |
| 2022/2/28 10:00 | 2,506 |
| 2022/2/28 11:00 | 1,753 |
| 2022/2/28 12:00 | 1,559 |
| 2022/2/28 13:00 | 1,470 |
| 2022/2/28 14:00 | 1,932 |
| 2022/2/28 15:00 | 2,025 |
| 2022/2/28 16:00 | 1,956 |
| 2022/2/28 17:00 | 1,865 |
| 2022/2/28 18:00 | 1,813 |
| 2022/2/28 19:00 | 1,377 |
| 2022/2/28 20:00 | 1,078 |
| 2022/2/28 21:00 | 756 |
| 2022/2/28 22:00 | 642 |
| 2022/2/28 23:00 | 463 |
|  |  |
|  |  |

**附表10 2月28日路段11各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 382 |
| 2022/2/28 1:00 | 120 |
| 2022/2/28 2:00 | 252 |
| 2022/2/28 3:00 | 173 |
| 2022/2/28 4:00 | 270 |
| 2022/2/28 5:00 | 186 |
| 2022/2/28 6:00 | 355 |
| 2022/2/28 7:00 | 666 |
| 2022/2/28 8:00 | 1,113 |
| 2022/2/28 9:00 | 1,496 |
| 2022/2/28 10:00 | 1,677 |
| 2022/2/28 11:00 | 1,288 |
| 2022/2/28 12:00 | 1,168 |
| 2022/2/28 13:00 | 1,034 |
| 2022/2/28 14:00 | 1,413 |
| 2022/2/28 15:00 | 1,450 |
| 2022/2/28 16:00 | 1,531 |
| 2022/2/28 17:00 | 1,495 |
| 2022/2/28 18:00 | 1,693 |
| 2022/2/28 19:00 | 1,145 |
| 2022/2/28 20:00 | 901 |
| 2022/2/28 21:00 | 611 |
| 2022/2/28 22:00 | 573 |
| 2022/2/28 23:00 | 397 |
|  |  |
|  |  |

**附表11 2月28日路段12各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 202 |
| 2022/2/28 1:00 | 186 |
| 2022/2/28 2:00 | 165 |
| 2022/2/28 3:00 | 195 |
| 2022/2/28 4:00 | 188 |
| 2022/2/28 5:00 | 216 |
| 2022/2/28 6:00 | 301 |
| 2022/2/28 7:00 | 573 |
| 2022/2/28 8:00 | 854 |
| 2022/2/28 9:00 | 1,296 |
| 2022/2/28 10:00 | 1,362 |
| 2022/2/28 11:00 | 1,144 |
| 2022/2/28 12:00 | 895 |
| 2022/2/28 13:00 | 900 |
| 2022/2/28 14:00 | 1,168 |
| 2022/2/28 15:00 | 1,200 |
| 2022/2/28 16:00 | 1,196 |
| 2022/2/28 17:00 | 1,208 |
| 2022/2/28 18:00 | 1,176 |
| 2022/2/28 19:00 | 973 |
| 2022/2/28 20:00 | 647 |
| 2022/2/28 21:00 | 561 |
| 2022/2/28 22:00 | 492 |
| 2022/2/28 23:00 | 376 |
|  |  |
|  |  |

**附表12 2月28日路段13各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 166 |
| 2022/2/28 1:00 | 151 |
| 2022/2/28 2:00 | 132 |
| 2022/2/28 3:00 | 169 |
| 2022/2/28 4:00 | 164 |
| 2022/2/28 5:00 | 177 |
| 2022/2/28 6:00 | 281 |
| 2022/2/28 7:00 | 531 |
| 2022/2/28 8:00 | 798 |
| 2022/2/28 9:00 | 1,145 |
| 2022/2/28 10:00 | 1,187 |
| 2022/2/28 11:00 | 1,021 |
| 2022/2/28 12:00 | 704 |
| 2022/2/28 13:00 | 803 |
| 2022/2/28 14:00 | 1,046 |
| 2022/2/28 15:00 | 1,090 |
| 2022/2/28 16:00 | 1,020 |
| 2022/2/28 17:00 | 1,032 |
| 2022/2/28 18:00 | 954 |
| 2022/2/28 19:00 | 850 |
| 2022/2/28 20:00 | 539 |
| 2022/2/28 21:00 | 453 |
| 2022/2/28 22:00 | 426 |
| 2022/2/28 23:00 | 308 |
|  |  |
|  |  |

**附表13 2月28日路段14各小时车流量预测**

|  |  |
| --- | --- |
| 时段 | 车流量预测 |
| 2022/2/28 0:00 | 161 |
| 2022/2/28 1:00 | 115 |
| 2022/2/28 2:00 | 116 |
| 2022/2/28 3:00 | 148 |
| 2022/2/28 4:00 | 124 |
| 2022/2/28 5:00 | 197 |
| 2022/2/28 6:00 | 236 |
| 2022/2/28 7:00 | 479 |
| 2022/2/28 8:00 | 706 |
| 2022/2/28 9:00 | 922 |
| 2022/2/28 10:00 | 1,060 |
| 2022/2/28 11:00 | 897 |
| 2022/2/28 12:00 | 647 |
| 2022/2/28 13:00 | 746 |
| 2022/2/28 14:00 | 932 |
| 2022/2/28 15:00 | 995 |
| 2022/2/28 16:00 | 991 |
| 2022/2/28 17:00 | 955 |
| 2022/2/28 18:00 | 909 |
| 2022/2/28 19:00 | 838 |
| 2022/2/28 20:00 | 487 |
| 2022/2/28 21:00 | 394 |
| 2022/2/28 22:00 | 377 |
| 2022/2/28 23:00 | 247 |
|  |  |
|  |  |

问题四中其余各个路段未来24小时的限速值如下所示。

**附表14 2月28日各路段限速值（一）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 时间 | 路段1 | 路段2 | 路段3 | 路段4 | 路段5 | 路段6 | 路段7 |
| 2022/2/28 0:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 1:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 2:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 3:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 4:00 | 80 | 80 | 80 | 80 | 80 | 80 | 90 |
| 2022/2/28 5:00 | 80 | 80 | 80 | 80 | 80 | 90 | 100 |
| 2022/2/28 6:00 | 80 | 80 | 80 | 80 | 90 | 100 | 110 |
| 2022/2/28 7:00 | 80 | 80 | 80 | 90 | 100 | 110 | 120 |
| 2022/2/28 8:00 | 80 | 80 | 90 | 100 | 110 | 120 | 120 |
| 2022/2/28 9:00 | 80 | 90 | 100 | 110 | 120 | 120 | 120 |
| 2022/2/28 10:00 | 80 | 80 | 90 | 100 | 110 | 120 | 120 |
| 2022/2/28 11:00 | 80 | 80 | 80 | 90 | 100 | 110 | 120 |
| 2022/2/28 12:00 | 80 | 80 | 80 | 80 | 90 | 100 | 110 |
| 2022/2/28 13:00 | 80 | 80 | 80 | 85 | 95 | 105 | 115 |
| 2022/2/28 14:00 | 80 | 80 | 85 | 95 | 105 | 115 | 120 |
| 2022/2/28 15:00 | 80 | 80 | 80 | 85 | 95 | 105 | 115 |
| 2022/2/28 16:00 | 80 | 80 | 80 | 90 | 100 | 110 | 120 |
| 2022/2/28 17:00 | 80 | 80 | 80 | 85 | 95 | 105 | 115 |
| 2022/2/28 18:00 | 80 | 80 | 80 | 80 | 85 | 95 | 105 |
| 2022/2/28 19:00 | 80 | 80 | 80 | 80 | 80 | 85 | 95 |
| 2022/2/28 20:00 | 80 | 80 | 80 | 80 | 80 | 80 | 85 |
| 2022/2/28 21:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 22:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 23:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |

**附表15 2月28日各路段限速值（二）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 时间 | 路段8 | 路段9 | 路段10 | 路段11 | 路段12 | 路段13 | 路段14 |
| 2022/2/28 0:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 1:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 2:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 3:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 4:00 | 85 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 5:00 | 95 | 90 | 90 | 80 | 80 | 80 | 80 |
| 2022/2/28 6:00 | 105 | 100 | 100 | 90 | 80 | 80 | 80 |
| 2022/2/28 7:00 | 115 | 110 | 110 | 100 | 90 | 80 | 80 |
| 2022/2/28 8:00 | 120 | 120 | 120 | 110 | 100 | 90 | 80 |
| 2022/2/28 9:00 | 120 | 120 | 120 | 120 | 110 | 100 | 90 |
| 2022/2/28 10:00 | 120 | 120 | 120 | 120 | 115 | 105 | 95 |
| 2022/2/28 11:00 | 120 | 120 | 120 | 115 | 105 | 95 | 85 |
| 2022/2/28 12:00 | 115 | 115 | 115 | 105 | 95 | 85 | 80 |
| 2022/2/28 13:00 | 105 | 105 | 105 | 95 | 85 | 80 | 80 |
| 2022/2/28 14:00 | 115 | 105 | 105 | 95 | 85 | 80 | 80 |
| 2022/2/28 15:00 | 110 | 105 | 110 | 100 | 90 | 80 | 80 |
| 2022/2/28 16:00 | 110 | 100 | 105 | 95 | 85 | 80 | 80 |
| 2022/2/28 17:00 | 105 | 95 | 100 | 90 | 80 | 80 | 80 |
| 2022/2/28 18:00 | 100 | 100 | 100 | 90 | 80 | 80 | 80 |
| 2022/2/28 19:00 | 90 | 90 | 90 | 80 | 80 | 80 | 80 |
| 2022/2/28 20:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 21:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 22:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 2022/2/28 23:00 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
|  |  |  |  |  |  |  |  |

代码源码如下。

**数据预读取**

*"""*  
*数据加载模块*  
*"""*  
  
import pandas as pd  
from datetime import datetime  
import json  
import os  
import sys  
  
PATH = os.path.dirname(os.path.abspath(\_\_file\_\_))  
CACHE = os.path.join(PATH, 'cache')  
DATA = os.path.join(PATH, 'data')  
TRACE\_TABLE = os.path.join(DATA, 'track\_table')  
  
**def** read\_raw\_data(file\_name: str, chache=True) -> pd.DataFrame:  
 *"""*  
 *读取原始数据文件，返回 DataFrame。如果已经存在缓存文件，则直接读取缓存文件。*  
  
 *param*  
 *- file\_name: xlsx文件名*  
 *- chache: 是否使用缓存，默认为True*  
  
 *return*  
 *- DataFrame*  
 *"""*  
 print(f"Loading data from {file\_name}", end='...')  
  
 file\_path = os.path.join(TRACE\_TABLE, file\_name)  
  
 **if** **not** os.path.exists(file\_path):  
 **raise** FileNotFoundError(f"File {file\_path} not found")  
 **if** **not** file\_path.endswith('.xlsx'):  
 **raise** ValueError(f"File {file\_path} is not a xlsx file")  
  
 *# 判断缓存文件*  
 cache\_file\_name = ".".join([os.path.splitext(file\_name)[0], 'csv'])  
 cache\_file = os.path.join(CACHE, cache\_file\_name)  
 **if** **not** os.path.exists(CACHE):  
 os.makedirs(CACHE)  
 **if** os.path.exists(cache\_file) **and** chache: *# 有缓存文件，直接读取*  
 print("Cache found, using cache.")  
 **return** pd.read\_csv(cache\_file)  
  
 print("Reading data", end='...')  
 df = pd.read\_excel(file\_path)  
  
 *# 初始化结果列表*  
 results = []  
 track\_temp = []  
  
 *# 临时变量存储当前车辆信息，用于解析数据块*  
 current\_vehicle = None  
 current\_model = None  
 current\_track = None  
  
 cnt = 0  
  
 *# 遍历每一行*  
 **for** index, row **in** df.iterrows():  
 **if** pd.notna(row['车辆编号']): *# 数据块首行，获取车辆信息*  
 *# 处理数据块的第一行*  
 current\_vehicle = row['车辆编号']  
 current\_model = row['收费站/门架编号'].split('：')[1] *# 提取车型*  
 current\_track = None  
 **continue** *# 跳过数据块的第一行*  
 **if** pd.notna(row['轨迹编号']): *# 刷新轨迹编号*  
 current\_track = row['轨迹编号']  
  
 time\_items = [item[4] **for** item **in** track\_temp **if** item[4] **is** **not** None]  
 time\_max = max(time\_items) **if** len(time\_items) > 0 **else** 0  
 time\_min = min(time\_items) **if** len(time\_items) > 0 **else** 0  
 **if** len(track\_temp) > 0 **and**\  
 time\_max <= datetime.strptime("2022-02-27 23:59:59", "%Y-%m-%d %H:%M:%S").timestamp() **and**\  
 time\_min >= datetime.strptime("2022-02-22 0:0:0", "%Y-%m-%d %H:%M:%S").timestamp(): *# 有效数据*  
 results.extend(track\_temp)  
 **else**: cnt += 1  
 track\_temp = []  
   
 **if** (current\_track **is** **not** None): *# 只有读取到了轨迹信息才会进行数据处理，规避了轨迹编号缺失的情况（否则会受先前的影响）*  
 *# 读取数据行信息*  
 event = row['信息类型'].replace('门架信息', '门架')  
 location = row['收费站/门架编号'] **if** row['收费站/门架编号'] != '其他' **else** None  
 time = row['记录时间'] **if** row['记录时间'] != '——' **else** None  
 **if** time: *# 转换时间格式为时间戳*  
 time = datetime.strptime(time, '%d/%m/%Y %H:%M:%S').timestamp()  
 track\_temp.append([current\_vehicle, current\_model, current\_track, event, time, location])  
  
 time\_items = [item[4] **for** item **in** track\_temp **if** item[4] **is** **not** None]  
 time\_max = max(time\_items) **if** len(time\_items) > 0 **else** 0  
 **if** len(track\_temp) > 0 **and**\  
 time\_max <= datetime.strptime("2022-02-27 23:59:59", "%Y-%m-%d %H:%M:%S").timestamp() **and**\  
 time\_min >= datetime.strptime("2022-02-22 0:0:0", "%Y-%m-%d %H:%M:%S").timestamp(): *# 有效数据*  
 results.extend(track\_temp)  
 **else**: cnt += 1  
  
 track\_temp = []  
  
 *# 创建 DataFrame*  
 result\_df = pd.DataFrame(results, columns=['车辆编号', '车型', '轨迹编号', '事件', '时间', '发生地点'])  
 result\_df.to\_csv(cache\_file, index=False)  
 print("Done.")  
 **return** result\_df  
  
**def** load\_cache() -> None:  
 **for** i **in** range(1, 31):  
 read\_raw\_data(f"轨迹表{i}.xlsx", chache=False)  
  
**def** read\_location\_data() -> dict:  
 *"""*  
 *读取位置数据文件，返回字典*  
  
 *return*  
 *- dict*  
 *- gate: 门架编号和位置的字典*  
 *- sequence: 门架和收费站的顺序列表*  
 *"""*  
 print("Loading location data", end='...')  
 location\_data\_path = os.path.join(DATA, "location.json")  
 **if** **not** os.path.exists(location\_data\_path):  
 **raise** FileNotFoundError(f"File {location\_data\_path} not found")  
 **with** open(location\_data\_path, 'r', encoding="utf-8") as f:  
 location\_data = json.load(f)  
 print("Done.")  
  
 **return** location\_data

**数据输出**

*"""*  
*输出结果文件*  
*"""*  
import pandas as pd  
import os  
  
PATH = os.path.dirname(os.path.abspath(\_\_file\_\_))  
OUTPUT = os.path.join(PATH, 'output')  
  
**def** dump(df: pd.DataFrame, file\_name: str) -> None:  
 *"""*  
 *将 DataFrame 输出到文件*  
  
 *param*  
 *- df: DataFrame*  
 *- file\_name: 输出文件名*  
 *"""*  
 output\_path = os.path.join(OUTPUT, file\_name)  
 df.to\_csv(output\_path, index=False)  
 print(f"Data dumped to {output\_path}")

**统计门架漏失/正常统计结果的各项数据**

*"""*  
*统计门架漏失/正常统计结果的各项数据*  
*"""*  
import pandas as pd  
import dataLoader  
  
dfs = []  
**for** i **in** range(1, 31):  
 dfs.append(dataLoader.read\_raw\_data(f"轨迹表{i}.xlsx"))  
df = pd.concat(dfs, ignore\_index=True)  
  
*# 记录所有门架事件的列表*  
events = []  
  
*# 按照车辆编号和轨迹编号分组*  
grouped = df.groupby(['车辆编号', '轨迹编号'])  
  
**for** (vehicle\_id, track\_id), group **in** grouped:  
 *# 过滤出门架事件*  
 gantry\_events = group[group['事件'] == '门架'].sort\_values(by='时间')  
   
 *# 获取所有的门架名称*  
 gantry\_names = gantry\_events['发生地点'].tolist()  
   
 *# 检查门架是否连续*  
 **for** i **in** range(len(gantry\_names) - 1):  
 current\_gantry = gantry\_names[i]  
 next\_gantry = gantry\_names[i + 1]  
   
 *# 提取门架编号*  
 current\_gantry\_num = int(current\_gantry[2:])  
 next\_gantry\_num = int(next\_gantry[2:])  
   
 *# 如果门架编号不连续，记录缺失的门架*  
 **if** next\_gantry\_num != current\_gantry\_num + 1:  
 **for** missing\_gantry\_num **in** range(current\_gantry\_num + 1, next\_gantry\_num):  
 missing\_gantry\_name = f"门架{missing\_gantry\_num}"  
   
 *# 推断缺失门架的时间*  
 missing\_time = (gantry\_events.iloc[i]['时间'] + gantry\_events.iloc[i + 1]['时间']) / 2  
   
 events.append({  
 '门架名称': missing\_gantry\_name,  
 '状态': '漏过',  
 '车辆编号': vehicle\_id,  
 '车型': group.iloc[0]['车型'],  
 '轨迹编号': track\_id,  
 '时间戳': missing\_time  
 })  
   
 *# 记录正常的门架事件*  
 **for** \_, row **in** gantry\_events.iterrows():  
 events.append({  
 '门架名称': row['发生地点'],  
 '状态': '正常',  
 '车辆编号': vehicle\_id,  
 '车型': row['车型'],  
 '轨迹编号': track\_id,  
 '时间戳': row['时间']  
 })  
  
events\_df = pd.DataFrame(events)  
  
print(events\_df)  
events\_df.to\_csv('gate\_condition.csv', index=False)

**对每个收费站的常规车道数量和应急系统数量进行关于总成本的优化**

*"""*  
*对每个收费站的常规车道数量和应急系统数量进行关于总成本的优化*  
*> 数据输出中，应急系统关闭时间 指的是应急系统开启状态的最后一个小时。例如开启关闭分别为8和9，则实际开启了2小时。*  
*"""*  
  
import pandas as pd  
import pulp  
  
import dataOutput  
  
*# 定义常量*  
C\_x = 24 \* 6 \* 25.8 *# 每条常规车道的成本*  
C\_y = 414.8 *# 每套应急收费系统每小时的成本*  
P = 3600 / (0.25 \* 3.5 + 0.75 \* 19) *# 每条车道每小时能处理的车流量*  
P\_e = P *# 每套应急系统每小时能处理的车流量*  
  
**def** optimize\_cost(D, L, R):  
 *# 创建问题实例*  
 problem = pulp.LpProblem("Minimize\_Cost", pulp.LpMinimize)  
  
 *# 定义变量*  
 x = pulp.LpVariable("x", lowBound=1, cat='Integer') *# 常规车道数量*  
 y = pulp.LpVariable("y", lowBound=0, cat='Integer') *# 应急系统数量*  
  
 *# 目标函数（总费用）*  
 problem += C\_x \* x + C\_y \* y \* (R - L)  
  
 *# 约束条件*  
 **for** t **in** range(24 \* 6):  
 hour\_of\_day = t % 24  
 **if** L <= hour\_of\_day <= R: *# 开启应急系统，应急系统和常规车道同时处理车流*  
 problem += (x \* P + y \* P\_e >= D[t])  
 **else**:  
 problem += (x \* P >= D[t])  
  
 *# 求解问题*  
 problem.solve()  
  
 *# 返回结果*  
 **return** pulp.LpStatus[problem.status], x.varValue, y.varValue, pulp.value(problem.objective)  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 df = pd.read\_csv("./doc/表1表2\_各收费站逐小时进出车流.csv")  
 stations = df["收费站"].unique()  
 results = []  
  
 **for** station **in** stations:  
 D = list(map(**lambda** x: x \* 281689 / 17542, (df[df["收费站"] == station]["入站车流"] + df[df["收费站"] == station]["出站车流"]).values))  
 min\_cost = 999999  
 rec = []  
  
 **for** L **in** range(24): *# 应急系统开启*  
 **for** R **in** range(L, 24): *# 应急系统关闭*  
 status, x, y, cost = optimize\_cost(D, L, R) *# 求解最优*  
  
 **if** status == 'Optimal' **and** cost < min\_cost: *# 如果找到更优解*  
 min\_cost = cost  
 rec = [x, y, L, R]  
  
 results.append([station, rec[2], rec[3], rec[0], rec[1], min\_cost])  
  
 *# 保存结果到CSV文件*  
 results\_df = pd.DataFrame(results, columns=["收费站", "应急系统开启时间", "应急系统关闭时间", "常规车道数量", "应急系统数量", "总费用"])  
 dataOutput.dump(results\_df, "优化结果.csv")

**对各路段逐小时的限速优化求解**

import pandas as pd  
import numpy as np  
from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpStatus  
  
import dataLoader  
import dataOutput  
  
df = pd.read\_csv("./doc/预测的各收费站24小时车流.csv")  
distance\_gate = dataLoader.read\_location\_data()["gate"]  
distance = {}  
**for** i **in** range(1,15):  
 distance[f"路段{i}"] = distance\_gate[f"门架{i + 1}"] - distance\_gate[f"门架{i}"]  
  
print(distance)  
print(df.head())  
  
*# 参数定义*  
N = 3 *# 车道数*  
M = 140 *# 最大限速*  
R = 150 *# 车行间距*  
E = 10 *# 限速差值*  
  
A = 1  
B = 0.1 \* A  
  
*# 计算车道车流*  
density = df.iloc[:, 1:].div([distance[f"路段{i}"] **for** i **in** range(1, 15)], axis=1) / N  
  
*# 定义问题*  
prob = LpProblem("SpeedOptimization", LpMinimize)  
  
  
*# 定义变量*  
num\_hours = 24  
num\_segments = 14  
speeds = LpVariable.dicts("Speed", (range(num\_hours), range(num\_segments)), lowBound=80, upBound=M)  
ranges = LpVariable.dicts("Range", (range(num\_hours), range(num\_segments)), lowBound=R)  
  
*# 目标函数：*  
  
prob += lpSum(A \* (speeds[hour][segment])+\  
 B \* ranges[hour][segment] / R  
 **for** hour **in** range(num\_hours) **for** segment **in** range(num\_segments))  
  
*# 约束条件*  
*# 相邻路段限速差值约束*  
**for** hour **in** range(num\_hours):  
 **for** segment **in** range(num\_segments - 1):  
 prob += speeds[hour][segment] - speeds[hour][segment + 1] <= E  
 prob += speeds[hour][segment + 1] - speeds[hour][segment] <= E  
  
 *# 同一路段相邻时间限速差值约束*  
**for** hour **in** range(num\_hours - 1):  
 **for** segment **in** range(num\_segments):  
 prob += speeds[hour][segment] - speeds[hour + 1][segment] <= E  
 prob += speeds[hour + 1][segment] - speeds[hour][segment] <= E  
  
*# 车行间距约束*  
**for** hour **in** range(num\_hours):  
 **for** segment **in** range(num\_segments):  
 flow = df.iloc[hour, segment + 1] *# 获取车流量*  
 prob += speeds[hour][segment] >= (flow / N \* ranges[hour][segment] + distance[f"路段{segment + 1}"]) / 1000  
  
*# 求解问题*  
prob.solve()  
  
*# 结果解析*  
**if** LpStatus[prob.status] == 'Optimal':  
 speed\_results = np.zeros((num\_hours, num\_segments))  
 **for** hour **in** range(num\_hours):  
 **for** segment **in** range(num\_segments):  
 speed\_results[hour, segment] = speeds[hour][segment].varValue  
   
 *# 将所有大于120的值改为120*  
 speed\_results[speed\_results > 120] = 120  
   
 *# 将结果向上取整到最接近的5的倍数*  
 speed\_results = np.ceil(speed\_results / 5) \* 5  
   
 *# 将结果转换为 DataFrame*  
 speed\_df = pd.DataFrame(speed\_results, columns=[f"路段{i+1}" **for** i **in** range(num\_segments)])  
 speed\_df.index.name = '小时'  
   
 *# 保存到 CSV 文件*  
 dataOutput.dump(speed\_df, "optimized\_speeds.csv")  
**else**:  
 print("优化问题无解")

**绘制车辆通过门架的时间、位置、数量信息的三维分布图**

*"""*  
*绘制车辆通过门架的时间、位置、数量信息的三维分布图*  
*"""*  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from datetime import datetime  
  
import dataLoader  
  
plt.rcParams['figure.figsize'] = (8, 6)  
plt.rcParams['axes.grid'] = False  
plt.rcParams['font.sans-serif'] = ['SimHei'] *# 使用黑体显示中文*  
plt.rcParams['font.family'] = 'sans-serif'  
plt.rcParams['axes.unicode\_minus'] = False *# 正确显示负号*  
  
*# 读取数据*  
dfs = []  
**for** i **in** range(1, 31):  
 dfs.append(dataLoader.read\_raw\_data(f"轨迹表{i}.xlsx"))  
df = pd.concat(dfs, ignore\_index=True)  
location = dataLoader.read\_location\_data()  
gates = location["gate"].keys()  
  
*# 统计记录车辆通过门架的时间戳*  
print("statisticting", end='...')  
pass\_gate\_count = {}  
**for** i **in** range(len(df)):  
 row = df.iloc[i]  
 **if** row["事件"] == "门架":  
 gate = row["发生地点"]  
 **if** gate **in** gates:  
 **if** gate **not** **in** pass\_gate\_count:  
 pass\_gate\_count[gate] = []  
 pass\_gate\_count[gate].append(row['时间'])  
print("Done.")  
  
*# 获取所有时间戳的最大值和最小值*  
*# 生成时间区间序列*  
start\_time = datetime(2022, 2, 22)  
end\_time = datetime(2022, 2, 28)  
  
*# 将时间戳均分区间*  
bins = np.linspace(start\_time.timestamp(), end\_time.timestamp(), num=24 \* 6 + 1)  
  
*# 创建三维图形对象*  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
  
*# 初始化 x, y, z 数据列表*  
x\_data = np.arange(1, len(gates) + 1)  
y\_data = (bins[:-1] + bins[1:]) / 2  
z\_data = np.zeros((len(x\_data), len(y\_data)))  
  
*# 遍历 gates 和对应的时间戳列表，统计每个区间内的时间戳数量*  
print("counting", end='...')  
**for** i, gate **in** enumerate(gates):  
 times = pass\_gate\_count[gate]  
 counts, \_ = np.histogram(times, bins)  
 z\_data[i, :] = counts  
print("Done.")  
  
*# 将时间戳转换为可读的时间格式*  
y\_data\_readable = [datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S') **for** ts **in** y\_data]  
  
*# 创建网格*  
X, Y = np.meshgrid(x\_data, y\_data)  
  
*# 绘制三维曲面图*  
ax.plot\_surface(X, Y, z\_data.T, cmap='viridis')  
  
*# 设置图形标签和标题*  
ax.set\_xlabel('位置索引')  
ax.set\_ylabel('时间区间中点')  
ax.set\_zlabel('数量')  
ax.set\_title('时间、位置、数量信息的三维分布')  
  
*# 设置 y 轴刻度标签为可读的时间信息*  
ax.set\_yticks(y\_data)  
ax.set\_yticklabels(y\_data\_readable, rotation=45, ha='right')  
  
*# 显示图形*  
plt.show()

**从SPASS输出文件读取数据**

import pandas as pd  
import matplotlib.pyplot as plt  
from mpl\_toolkits.mplot3d import Axes3D  
import numpy as np  
import dataOutput  
  
plt.rcParams['font.sans-serif'] = ['SimHei'] *# 使用黑体显示中文*  
plt.rcParams['font.family'] = 'sans-serif'  
plt.rcParams['axes.unicode\_minus'] = False *# 正确显示负号*  
  
*# 读取数据*  
data = {}  
**for** i **in** range(1, 15):  
 df\_one = pd.read\_excel(f"./doc/各路段预测24h/路段{i}.xls")  
 data[f"路段{i}"] = list(df\_one["预测\_车流量\_模型\_1"][-24:])  
  
*# 转换数据格式*  
hours = list(range(24))  
df = pd.DataFrame(data, index=hours)  
df.index.name = '小时数'  
  
  
dataOutput.dump(df, 'traffic\_data.csv')  
  
*# # 准备数据*  
*# sections = list(range(1, 15))*  
*# hours = list(range(24))*  
*# X, Y = np.meshgrid(sections, hours)*  
*# Z = np.array([data[f"路段{section}"] for section in sections]).T*  
  
*# # 绘制三维图表*  
*# fig = plt.figure()*  
*# ax = fig.add\_subplot(111, projection='3d')*  
*# ax.plot\_surface(X, Y, Z, cmap='viridis')*  
  
*# # 设置标签*  
*# ax.set\_xlabel('路段')*  
*# ax.set\_ylabel('小时数')*  
*# ax.set\_zlabel('车流量')*  
  
*# plt.show()*

**搜索确认进出收费站，并修补或删除行程的“补录入站”和“补录出站”的“发生地点”**

*"""*  
*搜索确认进出收费站，并修补或删除行程的“补录入站”和“补录出站”的“发生地点”*  
*"""*  
  
import pandas as pd  
import numpy as np  
  
import dataLoader  
import dataOutput  
  
dfs = []  
**for** i **in** range(1, 3):  
 dfs.append(dataLoader.read\_raw\_data(f"轨迹表{i}.xlsx"))  
df = pd.concat(dfs, ignore\_index=True)  
  
location = dataLoader.read\_location\_data()  
  
*# 提取所有“补录入站”的记录*  
bu\_lu\_ru\_zhan\_records = df[df['事件'] == '补录入站']  
  
*# 速度（米/秒）*  
speed = 120 / 3.6  
  
entry\_results = []  
  
print("begin to search missing enrty station...")  
**for** index, record **in** bu\_lu\_ru\_zhan\_records.iterrows():  
 vehicle\_id = record['车辆编号']  
 track\_id = record['轨迹编号']  
  
 *# 找到相同车辆编号和轨迹编号的第一个门架事件*  
 gate\_events = df[(df['车辆编号'] == vehicle\_id) & (df['轨迹编号'] == track\_id) & (df['事件'] == '门架')]  
   
 *# 如果没有门架事件，跳过该记录*  
 **if** gate\_events.empty:  
 **continue**  
   
 first\_gate\_event = gate\_events.iloc[0]  
  
 *# 计算时间差（秒）*  
 time\_diff = first\_gate\_event['时间'] - record['时间']  
   
 *# 计算最大行进距离（米）*  
 max\_distance = time\_diff \* speed  
   
 *# 从第一个门架往前遍历，找到可能的收费站*  
 first\_gate\_location = first\_gate\_event['发生地点']  
 first\_gate\_index = location['sequence'].index(first\_gate\_location)  
   
 distance\_covered = 0  
 possible\_stations = []  
   
 **for** i **in** range(first\_gate\_index, -1, -1):  
 loc = location['sequence'][i]  
 **if** '收费站' **in** loc:  
 possible\_stations.append(loc)  
 **if** '门架' **in** loc:  
 distance\_covered = location["gate"][first\_gate\_location] - location["gate"][loc]  
 **if** distance\_covered > max\_distance:  
 **break**  
  
 **if** len(possible\_stations) == 0:  
 **continue**  
 **else**:  
 possible\_station = possible\_stations[-1]  
   
 entry\_results.append({  
 '车辆编号': vehicle\_id,  
 '轨迹编号': track\_id,  
 '可能的收费站': possible\_station  
 })  
print("search missing enrty station done.")  
df\_gatesearch = pd.DataFrame(entry\_results)  
dataOutput.dump(df\_gatesearch, 'enrty\_search.csv')  
  
*# 确定“补录出站”的收费站*  
exit\_results = []  
  
print("begin to search missing exit station...")  
**for** vehicle\_id, track\_id **in** df[['车辆编号', '轨迹编号']].drop\_duplicates().values:  
 *# 获取该行程的所有记录*  
 trip\_records = df[(df['车辆编号'] == vehicle\_id) & (df['轨迹编号'] == track\_id)]  
   
 *# 找到最后一个门架*  
 gate\_events = trip\_records[trip\_records['事件'] == '门架']['发生地点']  
  
 *# 如果没有门架事件，跳过该记录*  
 **if** gate\_events.empty:  
 **continue**  
   
 last\_gate = gate\_events.iloc[-1]  
   
 *# 定位最后一个门架的位置*  
 sequence = location['sequence']  
 last\_gate\_index = sequence.index(last\_gate)  
   
 *# 找到下一个收费站*  
 next\_station = None  
 **for** i **in** range(last\_gate\_index + 1, len(sequence)):  
 **if** sequence[i].startswith('收费站'):  
 next\_station = sequence[i]  
 **break**  
 exit\_results.append({  
 '车辆编号': vehicle\_id,  
 '轨迹编号': track\_id,  
 '可能的收费站': next\_station,  
 '填补的时间': df[(df['车辆编号'] == vehicle\_id) & (df['轨迹编号'] == track\_id) & (df['事件'] == '门架')].iloc[-1]['时间']  
 })  
print("search missing exit station done.")  
  
*# 将 exit\_results 列表保存到 exit\_search.csv 文件中*  
df\_exitsearch = pd.DataFrame(exit\_results)  
dataOutput.dump(df\_exitsearch, 'exit\_search.csv')  
  
*# 将 entry\_results 和 exit\_results 转换为字典，方便查找*  
entry\_dict = {(result['车辆编号'], result['轨迹编号']): result['可能的收费站'] **for** result **in** entry\_results}  
exit\_dict = {(result['车辆编号'], result['轨迹编号']): (result['可能的收费站'], result['填补的时间']) **for** result **in** exit\_results}  
  
*# 记录删除的行程数*  
deleted\_trips\_count = 0  
  
*# 遍历每一个行程，修补“发生地点”*  
print("begin to fill missing station...")  
**for** vehicle\_id, track\_id **in** df[['车辆编号', '轨迹编号']].drop\_duplicates().values:  
 trip\_key = (vehicle\_id, track\_id)  
   
 *# 检查是否需要修补“补录入站”和“补录出站”*  
 needs\_entry\_fix = df[(df['车辆编号'] == vehicle\_id) & (df['轨迹编号'] == track\_id) & (df['事件'] == '补录入站')]['发生地点'].isnull().any()  
 needs\_exit\_fix = df[(df['车辆编号'] == vehicle\_id) & (df['轨迹编号'] == track\_id) & (df['事件'] == '补录出站')]['发生地点'].isnull().any()  
   
 *# 如果需要修补，且对应修补方案中不存在，则删除该行程*  
 **if** (needs\_entry\_fix **and** trip\_key **not** **in** entry\_dict) **or** (needs\_exit\_fix **and** trip\_key **not** **in** exit\_dict):  
 df = df[~((df['车辆编号'] == vehicle\_id) & (df['轨迹编号'] == track\_id))]  
 deleted\_trips\_count += 1  
 **else**:  
 *# 修补“补录入站”的“发生地点”*  
 **if** needs\_entry\_fix **and** trip\_key **in** entry\_dict:  
 df.loc[(df['车辆编号'] == vehicle\_id) & (df['轨迹编号'] == track\_id) & (df['事件'] == '补录入站'), '发生地点'] = entry\_dict[trip\_key]  
   
 *# 修补“补录出站”的“发生地点”*  
 **if** needs\_exit\_fix **and** trip\_key **in** exit\_dict:  
 df.loc[(df['车辆编号'] == vehicle\_id) & (df['轨迹编号'] == track\_id) & (df['事件'] == '补录出站'), '发生地点'] = exit\_dict[trip\_key][0]  
 df.loc[(df['车辆编号'] == vehicle\_id) & (df['轨迹编号'] == track\_id) & (df['事件'] == '补录出站'), '时间'] = exit\_dict[trip\_key][1]  
print("fill missing station done.")  
  
dataOutput.dump(df, 'filled\_data.csv')  
remained\_cnt = len(df[['车辆编号', '轨迹编号']].drop\_duplicates())  
print(f"删除了 {deleted\_trips\_count} 条行程，剩余有效行程 {remained\_cnt} 条，删除占比 {deleted\_trips\_count / (deleted\_trips\_count + remained\_cnt) \* 100:.2f}%")

**统计门架忽略与正常计数的数据**

*"""*  
*统计门架忽略与正常计数的数据*  
*"""*  
import pandas as pd  
  
df = pd.read\_csv('./doc/总\_门架忽略与正常计数.csv')  
  
*# 计算“状态”为漏过的占比*  
total\_count = len(df)  
missed\_count = len(df[df['状态'] == '漏过'])  
missed\_ratio = missed\_count / total\_count  
  
*# 计算“状态”为漏过时，“门架名称”中各项的占比*  
missed\_df = df[df['状态'] == '漏过']  
missed\_by\_gate = missed\_df['门架名称'].value\_counts(normalize=True)  
  
*# 计算“状态”为漏过时，“门架名称”为“门架6”且时间戳范围在1645599600 ~ 1645747200（维修期间）之间的占比*  
missed\_gate6\_df = missed\_df[(missed\_df['门架名称'] == '门架6') &   
 (missed\_df['时间戳'] >= 1645599600) &   
 (missed\_df['时间戳'] <= 1645747200)]  
missed\_gate6\_ratio = len(missed\_gate6\_df) / missed\_count  
  
*# 计算不同的“车型”，“状态”为“漏过”的占比*  
missed\_by\_vehicle\_type = missed\_df['车型'].value\_counts(normalize=True)  
  
*# 计算各门架的漏过率，排除门架6在维修期间的漏过记录*  
filtered\_missed\_df = missed\_df[~((missed\_df['门架名称'] == '门架6') &   
 (missed\_df['时间戳'] >= 1645599600) &   
 (missed\_df['时间戳'] <= 1645747200))]  
total\_by\_gate = df['门架名称'].value\_counts()  
missed\_by\_gate\_count = filtered\_missed\_df['门架名称'].value\_counts()  
missed\_rate\_by\_gate = (missed\_by\_gate\_count / total\_by\_gate).fillna(0)  
  
  
print(f"状态为漏过的占比: {missed\_ratio:.2%}")  
print("\n状态为漏过时，门架名称中各项的占比:")  
print(missed\_by\_gate)  
print(f"\n状态为漏过时，门架名称为门架6且时间戳范围在1645599600 ~ 1645747200之间的占比: {missed\_gate6\_ratio:.2%}")  
print("\n不同的车型，状态为漏过的占比:")  
print(missed\_by\_vehicle\_type)  
print("\n各门架的漏过率:")  
print(missed\_rate\_by\_gate)  
print("\n各门架的漏过率（排除门架6在维修期间的漏过记录）:")  
print(missed\_rate\_by\_gate)

**收费站与区间车流量的统计分析**

*"""*  
*收费站与区间车流量的统计分析*  
*"""*  
  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from datetime import datetime  
  
import dataLoader  
  
plt.rcParams['figure.figsize'] = (8, 6)  
plt.rcParams['axes.grid'] = False  
plt.rcParams['font.sans-serif'] = ['SimHei'] *# 使用黑体显示中文*  
plt.rcParams['font.family'] = 'sans-serif'  
plt.rcParams['axes.unicode\_minus'] = False *# 正确显示负号*  
  
df = pd.read\_csv("./doc/表1表2-出入站补全清洗数据.csv")  
location = dataLoader.read\_location\_data()  
gates = list(location["gate"].keys())  
stations = [one **for** one **in** location["sequence"] **if** one **not** **in** gates]  
  
  
traffic\_stats = {station: {"入": 0, "出": 0} **for** station **in** stations}  
  
segments = {f"路段{i}": {"入": 0, "出": 0} **for** i **in** range(1, len(gates))}  
  
*# 遍历数据*  
**for** index, row **in** df.iterrows():  
 event = row["事件"]  
 location = row["发生地点"]  
   
 **if** event **in** ["补录入站", "MTC入站", "ETC入站"]:  
 **if** location **in** traffic\_stats:  
 traffic\_stats[location]["入"] += 1  
 **elif** event **in** ["补录出站", "MTC出站"]:  
 **if** location **in** traffic\_stats:  
 traffic\_stats[location]["出"] += 1  
 **elif** event == "门架":  
 *# 找到当前门架的索引*  
 current\_gate\_index = gates.index(location)  
 *# 更新路段统计*  
 **if** current\_gate\_index < len(gates) - 1:  
 segment\_name = f"路段{current\_gate\_index + 1}"  
 segments[segment\_name]["入"] += 1  
 **if** current\_gate\_index > 0:  
 segment\_name = f"路段{current\_gate\_index}"  
 segments[segment\_name]["出"] += 1  
  
*# 输出结果*  
print("收费站车流量统计:")  
**for** location, stats **in** traffic\_stats.items():  
 print(f"{location}: 入车流量={stats['入']}, 出车流量={stats['出']}")  
  
print("\n路段车流量统计:")  
**for** segment, stats **in** segments.items():  
 print(f"{segment}: 入车流量={stats['入']}, 出车流量={stats['出']}")

**统计逐小时门架/路段车流**

*"""*  
*统计逐小时门架/路段车流*  
*"""*  
import pandas as pd  
import numpy as np  
from datetime import datetime  
  
import dataLoader  
import dataOutput  
  
  
df = pd.read\_csv("./doc/表1表2\_出入站补全清洗数据.csv")  
location = dataLoader.read\_location\_data()  
gates = list(location["gate"].keys())  
  
*# 生成时间区间序列*  
start\_time = datetime(2022, 2, 22)  
end\_time = datetime(2022, 2, 28)  
time\_intervals = np.linspace(start\_time.timestamp(), end\_time.timestamp(), num=24 \* 6 + 1) *# 每小时一个区间*  
  
  
*# 初始化结果列表*  
results = []  
  
*# 按照收费站筛选数据*  
**for** gate **in** gates:  
 gate\_data = df[df['发生地点'] == gate]  
  
 pass\_hist, \_ = np.histogram(gate\_data['时间'].astype(np.int64), bins=time\_intervals)  
   
 *# 保存结果*  
 **for** i **in** range(len(time\_intervals) - 1):  
 results.append([gate, datetime.fromtimestamp(time\_intervals[i]), pass\_hist[i]])  
  
*# 转换为DataFrame并保存为CSV*  
result\_df = pd.DataFrame(results, columns=['门架', '时间区间起点', '车流'])  
dataOutput.dump(result\_df, 'gate\_traffic\_hourly.csv')

**统计逐小时收费站车流**

*"""*  
*统计逐小时收费站车流*  
*"""*  
import pandas as pd  
import numpy as np  
from datetime import datetime  
  
import dataLoader  
import dataOutput  
  
  
df = pd.read\_csv("./doc/表1表2\_出入站补全清洗数据.csv")  
location = dataLoader.read\_location\_data()  
gates = list(location["gate"].keys())  
stations = [one **for** one **in** location["sequence"] **if** one **not** **in** gates]  
  
*# 生成时间区间序列*  
start\_time = datetime(2022, 2, 22)  
end\_time = datetime(2022, 2, 28)  
time\_intervals = np.linspace(start\_time.timestamp(), end\_time.timestamp(), num=24 \* 6 + 1) *# 每小时一个区间*  
  
  
*# 初始化结果列表*  
results = []  
  
*# 按照收费站筛选数据*  
**for** station **in** stations:  
 station\_data = df[df['发生地点'] == station]  
   
 *# 筛选入站和出站事件*  
 entry\_data = station\_data[station\_data['事件'].str.contains('入站')]  
 exit\_data = station\_data[station\_data['事件'].str.contains('出站')]  
  
 entry\_hist, exit\_hist = [], []  
 **if** len(entry\_data['时间']) > 0:  
 *# 统计入站车流*  
 entry\_hist, \_ = np.histogram(entry\_data['时间'].astype(np.int64), bins=time\_intervals)  
 **if** len(exit\_data['时间']) > 0:  
 *# 统计出站车流*  
 exit\_hist, \_ = np.histogram(exit\_data['时间'].astype(np.int64), bins=time\_intervals)  
   
 *# 保存结果*  
 **for** i **in** range(len(time\_intervals) - 1):  
 results.append([station, datetime.fromtimestamp(time\_intervals[i]), entry\_hist[i], exit\_hist[i]])  
  
*# 转换为DataFrame并保存为CSV*  
result\_df = pd.DataFrame(results, columns=['收费站', '时间区间起点', '入站车流', '出站车流'])  
dataOutput.dump(result\_df, 'station\_traffic\_hourly.csv')

**检查数据中，同一轨迹内，通过两门的时间间隔是否合理，以及通过的顺序是否合理**

*"""*  
*检查数据中，同一轨迹内，通过两门的时间间隔是否合理，以及通过的顺序是否合理*  
*"""*  
import pandas as pd  
import dataLoader  
import dataOutput  
  
source = "轨迹表5.xlsx"  
  
df = dataLoader.read\_raw\_data(source)  
  
*# 筛选出包含“门架”的记录*  
df['门架序号'] = df['发生地点'].str.extract(r'门架(\d+)').astype(float)  
  
*# 按车辆编号和轨迹编号分组（即按照“行程”分组）*  
grouped = df.groupby(['车辆编号', '轨迹编号'])  
  
*# 存储异常行程*  
abnormal\_trips = []  
  
*# 遍历每个分组，检查数据*  
**for** (vehicle\_id, track\_id), group **in** grouped:  
 *# 筛选出门架记录并按时间排序*  
 gantry\_records = group.dropna(subset=['门架序号']).sort\_values(by='时间')  
   
 *# 初始化异常类型列表*  
 anomalies = []  
   
 *# 检查门架序号是否按升序排列*  
 **if** **not** gantry\_records['门架序号'].is\_monotonic\_increasing:  
 anomalies.append('门架顺序异常')  
   
 *# 检查整个分组的时间是否按升序排列*  
 **if** **not** group['时间'].dropna().is\_monotonic\_increasing:  
 anomalies.append('时间顺序异常')  
   
 **if** anomalies:  
 abnormal\_trips.append((vehicle\_id, track\_id, ', '.join(anomalies)))  
  
abnormal\_df = pd.DataFrame(abnormal\_trips, columns=['车辆编号', '轨迹编号', '异常类型'])  
  
abnormal\_counts = abnormal\_df['异常类型'].value\_counts()  
  
print()  
print(f"there's {len(abnormal\_df)} abnormal trips in {source}, the rate is {len(abnormal\_df) / len(grouped):.2%}(total {len(grouped)})")  
print("abnormal type:")  
print(abnormal\_counts)  
print()  
  
dataOutput.dump(abnormal\_df, '异常行程.csv')

**用于检查车辆行程，探究数目不匹配的原因**

*"""*  
*用于检查车辆行程，探究数目不匹配的原因*  
*结果：发现了存在轨迹编号缺失的情况，需要处理，进而修改了dataLoader.py中的read\_raw\_data函数*  
*"""*  
import pandas as pd  
import dataLoader  
import dataOutput  
  
df = dataLoader.read\_raw\_data("轨迹表1.xlsx")  
  
*# 填充时间列中的空数据*  
df['时间'] = df['时间'].fillna(0)  
  
*# 新建一列，将“车辆编号”和“轨迹编号”合并为一个字符串，作为识别符*  
df['组合编号'] = df['车辆编号'] + '\_' + df['轨迹编号'].astype(str)  
  
*# 获取不同的组合编号列表*  
unique\_combinations = df['组合编号'].unique()  
  
  
unique\_combinations\_df = pd.DataFrame(unique\_combinations, columns=['组合编号'])  
dataOutput.dump(unique\_combinations\_df, 'unique\_combinations.csv')  
  
print("over.")

**Makefile用于清理目录**

PYTHON = python3.10  
DATA\_LOADER = ./dataLoader.py  
  
*# 默认目标*  
.PHONY: all  
all: cache  
  
*# 清理缓存*  
.PHONY: clean-cache  
clean-cache:  
 -rm -rf ./cache/\*\*  
  
*# 生成缓存*  
.PHONY: cache  
cache:  
 $(PYTHON) $(DATA\_LOADER) cache  
  
*# 清理输出*  
.PHONY: clean-output  
clean-output:  
 -rm -rf ./output/\*\*