AIS 轨迹数据处理与预测

摘要

AIS(Automatic Identification System)船舶自动识别系统，是一种安装于船舶上的无线电通信导航避碰设备，可适用于内河、港口及外海等各类船舶。其可以为海事系统及船务公司提供大量与船舶有关的静态和动态信息。然而，想要利用好这些信息，需要对数据进行合适的处理以及分析。本文通过建立合适的模型，对题目提供的 AIS 轨迹数据进行了数据重建，数据压缩和数据预测，以便这些数据能更好的被利用。

针对问题一，本文将其分为两个子问题，首先进行噪声点的去除再进行缺失点的补充。对于噪声点的去除，本文将噪声点分为两类，根据两类噪声点不同的特征，本文分别进行了去除。一类是重复点，其特征是经纬坐标与上一时刻完全相同，但速度却较大，二者产生矛盾。另一类是漂移点，其特征是这一时刻船舶的经纬度坐标与相邻时刻经纬度坐标差值极大，坐标数据明显发生错误。同样，本文也将缺失点分为两类。第一类缺失点的前后时刻相差不是特别大，对于这种缺失点，采用简单的线性插值的方式即可得到很好的效果。第二类缺失点的前后时刻相差较大，此时如果强行对缺失点进行补充反而会造成更大的误差，补充的点可能在陆地上，因此不对这类缺失点进行处理。

针对问题二，本文采用了经典的 Douglas-Peucker 算法对船舶行驶轨迹数据进行压缩，并选定了两个指标用来评价压缩质量，包括压缩率和误差。其中压缩率指压缩后的数据条数与压缩前的数据条数的比值。误差采用平均垂直欧式距离进行计算，垂直欧氏距离即压缩前的每个点的位置与压缩后的轨迹的最短垂直距离。针对其中需要人为选定的阈值，本文通过遍历的方式找到了合适的取值为 20m，数据压缩后，压缩率达到了 0.213，误差仅有 3.057m，且通过绘制压缩前后轨迹的方式可以看出，压缩效果较好。

针对问题三，首先进行了已知数据的统计，得出 10 月 17 号到 10 月 25 号每天各时间段经过以长江大桥为观测断面的船流量。考虑到样本极少，每个时间段仅有 9 个样本，不应采用较复杂的模型如 ARIMA 模型，神经网络等，因此选择了一次指数平滑模型进行数据预测，最后每个时间段的预测结果均符合大致趋势，预测结果较好。

关键词：　线性插值　Douglas-Peucker 算法　指数平滑模型

目录

1 问题重述

1.1 问题背景

随着国家经济实力的迅速发展，我国水上经济活动也日益频繁，其促进经济发展的同时也加大了国家和地区对水上安全监管与调度的难度，而随着船舶自动识别系统(AIS)的使用，船务公司以及海事系统对水上交通管理有了大量的数据支撑，然而，这些数据相当庞大，且在传输过程中易受噪声的影响，造成数据异常以及数据丢失，因此我们需要寻找合适的模型对这些数据进行处理以及压缩，提高数据的质量以便其能得到更好的应用。

1.2 具体问题

现在我们有 2016 年 10 月 17 日至 25 日长江武汉段的船舶 AIS 轨迹数据，数据结构中包括采集数据时船舶当前的航行时间、船舶识别号(MMSI)、船舶当前时刻的经纬度、船舶上一时刻的经纬度、船舶航行的速度、船舶航行的船首向、数据解析校验码。我们需要对这些数据进行处理，完成以下问题:

问题一(数据重建)：AIS 轨迹数据在采集，传输和解析的过程中易受噪声的影响产生噪声点，另外也可能产生数据丢失，导致船舶 AIS 轨迹数据采集质量下降，应用价值不高。要求根据给定数据的数据结构，建立合适的模型对数据进行预处理，实现噪声消除以及数据重建，提高数据质量。

问题二(数据压缩)：航载 AIS 设备一般 2*s*-6*min* 发布一条消息，这使船舶 AIS 轨迹数据相当庞大，一方面这给海事局以及船务公司存储数据的能力带来了很大的挑战，另一方面电子海图显示与信息系统(ECDIS)在导入如此庞大的船舶 AIS 轨迹数据时效率非常低，海图响应的速度也非常缓慢，极大影响了船舶轨迹显示的时效性，为了降低存储成本以及提高 ECDIS 平台回放和再现船舶轨迹的效率，有必要对船舶 AIS 轨迹数据进行压缩处理。要求建立合适的数学模型实现船舶 AIS 轨迹数据的高效压缩并对压缩质量进行评价。

问题三(数据预测)：船舶 AIS 轨迹数据蕴含着丰富的船舶航行信息，有助于海事局以及船务公司统计分析特定水域的船舶交通流量，对于海事部门实现精细化管理起着重要作用。在对数据进行预处理完毕之后，数据质量已经较好，可以利用其做一些统计预测工作。要求利用给定的船舶 AIS 轨迹数据，以长江大桥为观测断面，统计船舶到达规律，并预测 2016 年 10 月 26 日在不同时间段内通过该观测断面的船舶交通流量。

2 问题分析

2.1 问题一的分析

问题一实际上包括了两个小问，噪声点的消除以及缺失点的补充。首先需要考虑的是这两个小问的完成顺序，由于如果我们先进行缺失点的补充，那么在这个过程中实际上我们会把所有点都当成正常点来看待，这样显然会补充进更多错误的点，反而会造成数据质量的下降。因此我们要先进行数据噪声点的消除再进行缺失数据的补充。

噪声点的消除：通过对数据文件的分析，我们可以归纳出两种噪声点。第一种是重复点，即在前后时刻船的位置坐标相同，但速度较大，不可能坐标没有任何变化，这种噪声点的出现是因为船舶自带定位装置在复杂环境下可能无法实时获取位置数据，从而导致连续一段报文中位置数据相同[1]，如果不对这种点进行消除，那么会误判为船舶处于停止状态，因此需要对这种数据进行消除。第二种是漂移点，即船的位置坐标与上一时刻和下一时刻船的位置坐标发生明显漂移，这种异常点的出现是因为数据传输过程中受到干扰，位置坐标被改变。对于这一种噪声点的判断只需判别这一时刻到上一时刻的平均速度是否过大即可。

缺失数据的补充：首先我们对缺失数据的判断只需计算前后两时刻的时间差是否大于 360 秒即可。之后通过分析数据发现缺失数据也分为两种情况。第一种缺失的数据前后两点的时间差不是特别大，这种情况仅需做简单的线性插值即可。第二种缺失的数据前后两点的时间差极大，这种情况即使运用复杂的方法对缺失数据进行预测也会产生较大误差，因此不对这种缺失数据做处理。

2.2 问题二的分析

问题二要求对 AIS 轨迹数据进行高效压缩并对压缩质量进行评价。那么我们既要寻找一个合适的压缩算法也需要找到合适的指标对压缩效果进行评价。在压缩算法方面，事实上已经有了非常成熟的轨迹压缩算法，即 Douglas-Peucker 算法[2]，我们将其编程实现即可。对于压缩效果评价方面，我们不能仅仅考虑压缩后的数据是否足够少，还需要考虑到压缩后的数据的精度怎么样，即应当在保证压缩后数据量尽可能少的同时还要保证产生的误差较小。然而，这两个方面是相悖的，压缩后数据越少，其带来的误差必然也越大，因此我们需要在两者中进行权衡，将轨迹数据压缩到合适的大小。

2.3 问题三的分析

问题三要求对 26 日各时间段船流量数据进行预测，首先需对已知数据进行统计，要注意的是不能统计数据压缩后的数据，因为压缩过程中时间数据已经丢失，而是要统计数据压缩前的数据。另外考虑到已知数据极少，只有 9 天的数据，因此不能采用很复杂的模型，而是采用相对简单的指数平滑模型进行预测比较合适。

3 模型假设

1. 速度大于 2 节时，船舶的位置坐标应发生明显变化。

2. 由于船舶在两时刻之间的速度数据未知，我们假设船舶在相邻两时刻之间的加速度的绝对值不会特别大，即船舶在两时刻之间不会加速到特别大的速度，也不会减速到特别小的速度。

3. 假设地球是个球体，其半径为 6371 千米。

4. 正常情况下 AIS 设备发布数据的时间间隔不会超过 6 分钟

4 符号说明

表 1: 符号说明

符号	符号描述
t_i	每一个文件中第 i 个数据的时间戳，即 1970 年 1 月 1 日到当前时间的秒数，单位为秒
lon_i	第 i 个时刻船舶的经度，单位为度
lat_i	第 i 个时刻船舶的纬度，单位为度
v_i	第 i 个时刻船舶的速度，单位为节
r	地球半径，单位为米
$D_{\max}$	Douglas-Peucker 算法中的阈值
Cr	压缩率
N_{before}	数据压缩前的数据条数
N_{after}	数据压缩后的数据条数
$error$	数据压缩产生的误差，单位为米
d_i	数据压缩前第 i 条坐标数据与数据压缩后的轨迹最短垂直距离，单位为米
a	一次指数平滑模型中的平滑系数

5 问题一的求解

5.1 噪声点的去除

通过对提供数据的分析，共归纳出两种噪声点。我们称第一种噪声点为重复点，重复点即前后两时刻船舶的位置坐标完全相同，但速度大小不为零，船舶不可能停留在原地，这种点会造成误判船舶处于停止状态，因此需要将这种点去除。我们称第二种噪声点为漂移点，漂移点即某一时刻船舶的坐标与其相邻时刻船舶位置坐标相差过大，这种点会对船舶轨迹的描绘产生极其大的影响，因此这种噪声点也应该去除。

5.1.1 重复点的去除

由对重复点的定义以及模型假设 1，速度大于 2 节时，船舶的位置坐标应发生明显变化，我们很容易得出消除重复点的方法，即遍历所有数据，若当前时刻与上一时刻船舶位置坐标相同且速度大于 2 节，那么就将此点去除。即每一时刻的数据判断为重复点的条件如下：

$$\begin{cases} lon_i = lon_{i-1} \\ lat_i = lat_{i\text{-}1} \\ v_i > 2 \end{cases}$$

经过计算，总共有 28372 个重复点，重复点去除之前，共有 1361363 条数据，重复点去除之后，还剩 1332991 条数据。

5.1.2 漂移点的去除

由对漂移点的定义，某一时刻船舶的坐标与其相邻时刻船舶位置坐标相差过大，那么对于这种噪声点的判断只需判断这一时刻与上一时刻之间的平均速度是否大于某个值即可，对于这个值的选择，采取的策略如下：由模型假设 2，船舶在两时刻之间速度较稳定，那么此值应该取 $\max(v_i, v_{i\text{-}1})$ 即可，为了减少将正常的点删除的可能性，使模型更具有鲁棒性，这里将这个值定为 $2\times\max(v_i, v_{i\text{-}1})$。

为了得到这一时刻与上一时刻之间的平均速度，我们首先需要计算这一时刻与上一时刻之间的距离。由经纬度坐标计算距离的 Haversine 公式如下(其建立在假设地球为球体的基础上):

$$d = 2\times r\arcsin(\sqrt{\sin^2((lat_i - lat_{i-1})/2) + \cos(lat_i)\cos(lat_{i-1})\sin^2((lon_i - lon_{i-1})/2)})$$

计算出距离后，我们用距离除以两个时刻的时间差得到速度(*m*/*s*)，再将其乘以 1.94 即可得到单位为节的速度。即：

$$v = \frac{d}{t_i - t_{i-1}} \times 1.94$$

因此，每一时刻的数据判断为漂移点的条件如下:

$$v > 2 \times \max(v_i, v_{i-1})$$

经过计算，共有 7044 个漂移点，将漂移点去除之后，还剩 1325947 条数据。

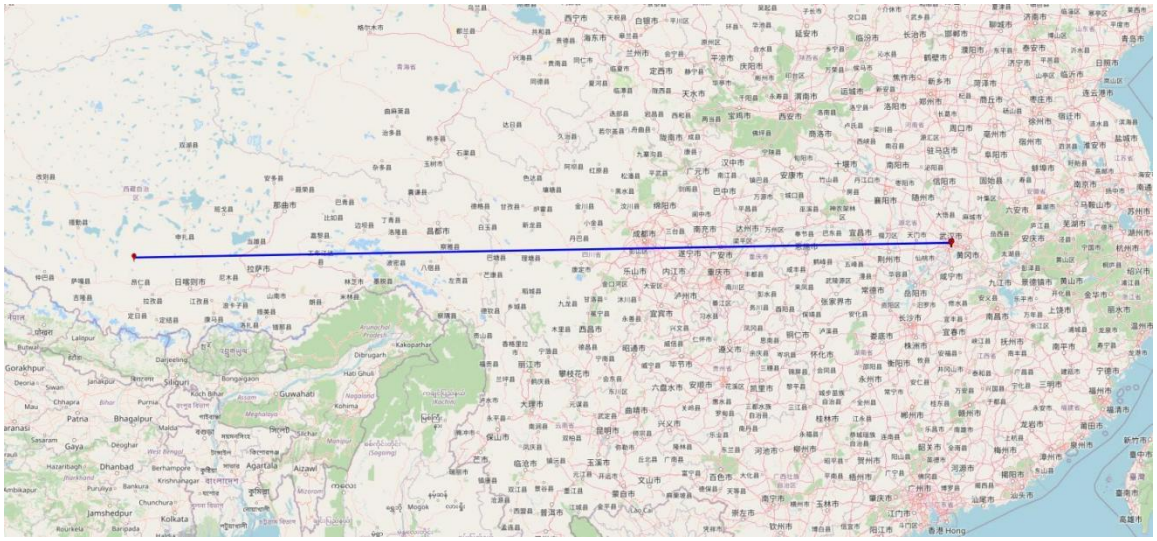

图 1: 去除漂移点前的轨迹

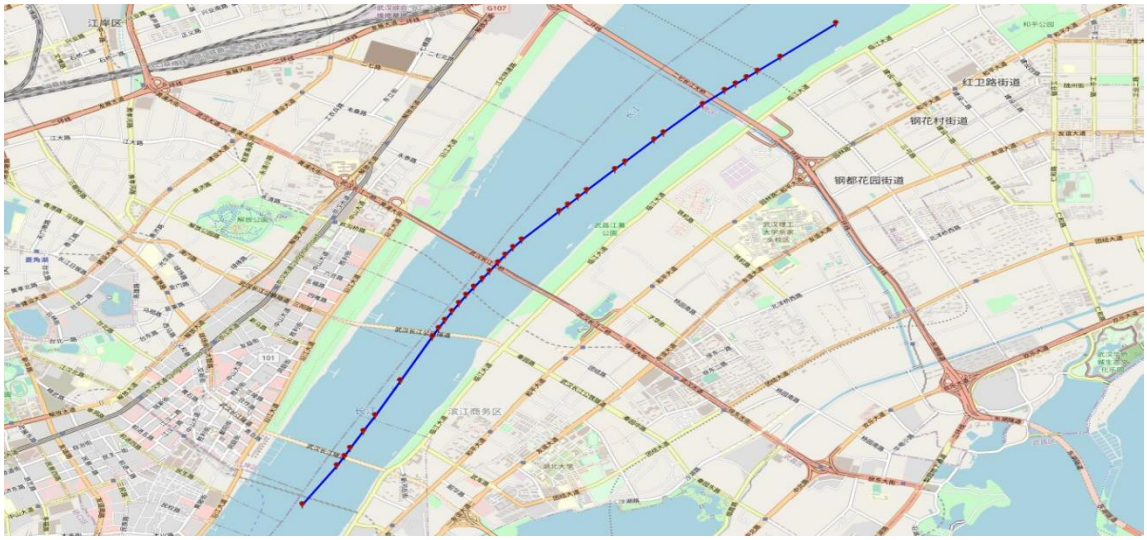

图 2: 去除漂移点后的轨迹

图 1，图 2 分别为 20161017 (287).txt 文件中去除漂移点前和去除漂移点后的轨迹。我们可以看到，在去除漂移点前，漂移点对船舶航行轨迹产生了极大的影响，而在去除漂移点后，船舶航行轨迹恢复正常。

5.2 缺失点的补充

首先我们由题目知，船载 AIS 设备一般 2*s*-6*min* 发布一条信息，因此我们对

缺失数据的判断只需计算前后两时刻的时间差是否大于 360 秒即可。但我们不能简单的对这些缺失数据做相同的处理，通过分析数据发现缺失数据也应分为两种情况。第一种缺失的数据前后两点的时间差不是特别大，这种情况下进行缺失点的补充不需要用复杂的算法，仅需进行线性插值即可，这是因为通过分析数据发现，船舶的行驶轨迹大都近似直线，即使有弯曲，曲率半径也非常大，且在两个时刻之间船舶行驶距离也较短。第二种缺失的数据前后两点的时间差极大，这说明中间缺失的数据极其的多，这种情况即使运用复杂的方法对缺失数据进行预测也会产生较大误差，因此不对这种缺失数据做处理。在第一种缺失的数据和第二种缺失的数据的边界时间差选择上，这里选取为 1 小时，这样既能保证补充足够多的缺失点，也防止了因补充的点太多而产生较大误差。

5.2.1 第一类缺失点的补充

对第一类缺失点的判断条件如下:

$$360 < t_i - t_{i\text{-}1} <= 3600$$

当上述判断条件成立时，第 i-1 个时刻和第 i 个时刻之间为第一类缺失点，我们采取线性插值的方式进行补充。首先计算需要补充的点的个数，为了保证相邻两个时刻之间时间戳的插值不超过 360 秒，我们从第 i-1 个时刻开始，每 360 秒插入一个点即可，那么共需插入 $(t_i - t_{i\text{-}1})/360$ 个点，其中第 j 个点的经纬度坐标计算公式如下:

$$lon_{i-1+j} = (lon_i\text{-}lon_{i\text{-}1})/(t_i - t_{i-1}) \times 360 \times j + lon_{i-1}$$

$$lat_{i-1+j} = (lat_i\text{-}lat_{i\text{-}1})/(t_i - t_{i-1}) \times 360 \times j + lat_{i-1}$$

注意当缺失点插入完毕之后，原先第 i 个时刻的数据变为第 $(t_i - t_{i\text{-}1})/360 + i$ 个时刻的数据。

补充完第一类缺失点后，数据扩展到 1450649 条。

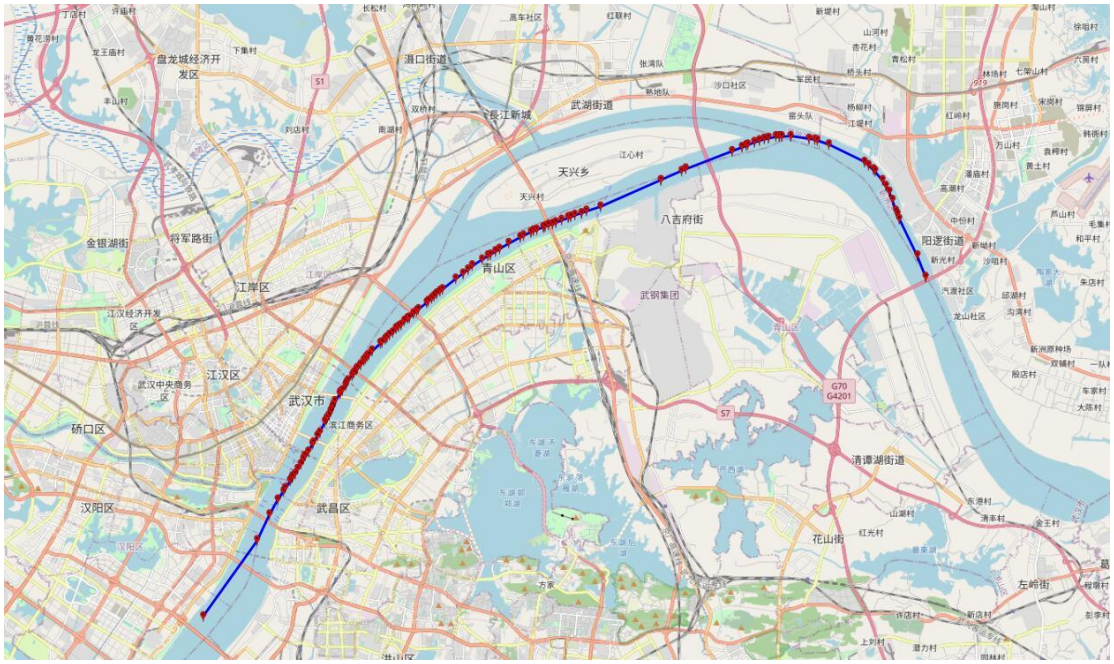

图 3: 补充缺失点之前的轨迹

图 4: 补充缺失点之后的轨迹

图 3，图 4 分别为 20161017 (133).txt 文件中补充缺失点之前和补充缺失点之后的轨迹，从图中可以看出，经过线性插值，在补充缺失值的同时，几乎没有改变船舶行驶的轨迹。

5.2.2 第二类缺失点的补充

对第二类缺失点的判断条件如下:

$$t_i - t_{i-1} > 3600$$

当判断为第二类缺失点时，为了放止产生较多的误差，我们不对这类缺失点进行补充。

图 5: 第二类缺失点

图 5 为 20161017 (1).txt 文件中的第二类缺失点，如图所示，两个时刻相差时间太大，导致这两个时刻的经纬度坐标相差很大，中间缺失了数量较多的点，采用任何方法进行缺失点的补充都会产生较大误差，因此不对这类缺失点进行补充。

6 问题二的求解

6.1 数据压缩算法

采用经典的轨迹压缩算法 Douglas-Peucker 算法进行船舶航行轨迹的压缩。设人为定义的阈值为 $D_{\max}$，那么 Douglas-Peucker 算法的步骤如图 6 所示。即首先在起点和终点之间连一条直线，计算其余所有的点到这条直线的距离，并保留其中距离最大的点和距离的最大值，若距离的最大值小于 $D_{\max}$，那么直接将起点和终点加入特征点集，忽略掉中间的所有点，反之将这个距离最大的点加入特征点集，并用这个最大的点将原始点集分为左右两部分，对两部分重复进行以上步骤，直到所有点被遍历完为止。算法执行完毕后，特征点集中的点即为轨迹数据压缩后剩余的点。

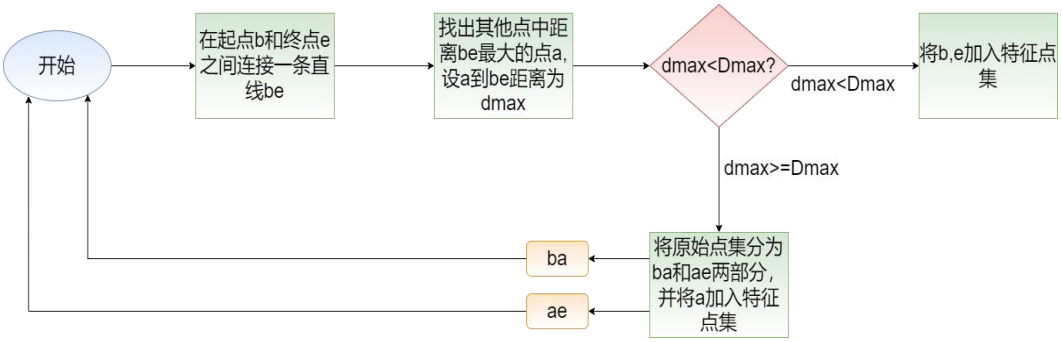

图 6: Douglas-Peucker 算法流程图

由算法流程可以看出，$D_{\max}$ 的选取至关重要，其关系着压缩后数据的多少以及压缩后产生误差的大小。对于 $D_{\max}$ 的选取在下一节压缩质量评价中介绍。

6.2 压缩质量评价

在轨迹数据压缩的质量评价中，我们显然不能简单认为压缩后数据量越小，压缩效果就越好。这是因为数据量减少的同时，其带来的后果是数据精度的丢失，即数据压缩后的船舶运行轨迹可能与数据压缩前的船舶运行轨迹有较大差别，由此，我们引入两个指标对数据压缩质量进行评价，即压缩率与误差。

设压缩率用 Cr 表示，数据压缩前共有 N_{before} 条数据，数据压缩后共有 N_{after} 条数据，那么 Cr 的计算公式如下

$$Cr = \frac{N_{after}}{N_{before}}:$$

由计算公式可以看出，Cr 越小证明数据压缩后数据量越小，即压缩质量与 Cr 呈负相关。需要注意的是这里的 N_{before} 并不是原始数据条数，而是经过噪声点的去除和缺失点的补充之后的数据条数。即 $N_{before}=1450649$ 。

我们对误差采用平均垂直欧式距离进行计算。垂直欧氏距离即压缩前的每个点的位置与压缩后的轨迹的最短垂直距离，计算完之后再取平均值即可。设数据压缩前第 i 条坐标数据与数据压缩后的轨迹最短垂直距离为 d_i ,用 $error$ 表示误差，那么 $error$ 的计算公式如下:

$$error = \frac{\sum_{i=1}^{N_{before}} d_i}{N_{before}}$$

在确定好评价指标之后，我们需要决定如何权衡好这两个指标之间的关系。从算法流程中可以看出， $D_{\max}$ 的选取同时影响着这两个指标， $D_{\max}$ 越大，意味

着被选入特征点集的点越少，那么数据压缩率就越小，误差也就越大。为了寻找合适的 $D_{\max}$，本文通过将 $D_{\max}$ 从 $1m$ 遍历到 $100m$ 并求出相应的 Cr 和 $error$ 寻找最合适的 $D_{\max}$，结果如图 7 所示。图 7 中左图为 Cr 随 $D_{\max}$ 的变化曲线，右图为 $error$ 随 $D_{\max}$ 的变化曲线。

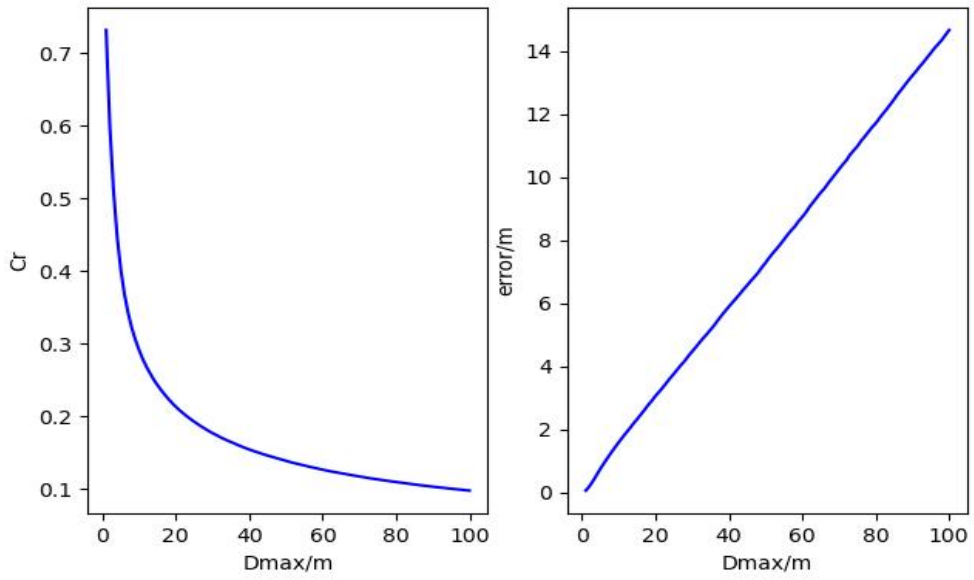

图 7: $D_{\max}$ 的选取对压缩质量的影响

从上图中可以看出，$D_{\max}$ 从 $1m$ 增长到 $20m$ 左右时压缩率急剧下降，从 $20m$ 增长到 $100m$ 时压缩率下降速度较缓慢，而误差随 $D_{\max}$ 的增长几乎是线性增长的。那么我们可以取 $D_{\max}$ 为 $20m$，此时数据压缩率较低，误差也不会太高，而且如果再增加 $D_{\max}$，压缩率并不会有显著的降低，而误差依然在快速增长。

6.3 最终压缩结果

选取 $D_{\max}=20m$ 对轨迹数据进行压缩后，还剩 308635 条数据(数据压缩前有 1450649 条数据)，数据压缩率为 0.213，误差为 $3.057m$。压缩效果如图 8 和图 9 所示:

图 8: 数据压缩前船舶行驶轨迹

图 9: 数据压缩后船舶行驶轨迹

其中图 8 为 20161017 (8).txt 文件中经过数据重建后的船舶行驶轨迹，图 9 为其经过数据压缩之后的船舶行驶轨迹。对比两图可以看出，在进行轨迹数据压缩之后，坐标数据数量明显减少且轨迹与数据压缩前几乎吻合，由此可见，数据压缩质量较高。

7 问题三的求解

7.1 已知数据的统计

在进行 2016 年 10 月 26 日的数据预测之前，我们需要对已知数据进行处理，

获得 17 日到 25 日每天各时间段通过长江大桥观测断面的船流量，注意这里要用的是数据压缩前的数据而不是数据压缩后的数据，因为数据压缩仅仅是对位置坐标数据进行压缩，在压缩过程中时间数据丢失，因此需采用数据压缩前的数据进行预测。

首先选取观测面，选定四个点(114.28354°,30.55238°)，(114.28341°,30.55218°)，(114.29277°,30.54731°)，(114.29265°,30.54711°)，这四个点框住的矩形区域即为观测断面，如图 10 所示:

图 10: 观测断面示意图

选定观测面后，将一天划分为 8 个时间段，分别为 0-3 时，3-6 时，6-9 时，9-12 时，12-15 时，15-18 时，18-21 时，21-24 时，然后遍历每一天中的每一艘船舶的 AIS 轨迹数据，若其出现在矩形区域内，那么对应时间段的船流量加一即可。

这一部分的关键步骤就在于如何判断一个坐标点是否在矩形区域内，设矩形观测断面四个点分别为 A，B，C，D，四条边表示为向量分别为 ***AB***，***CD***，***AC***，***BD***，需要判断的点为 E。那么(这里×表示叉乘)

$$(\boldsymbol{AB} \times \boldsymbol{AE}) \cdot (\boldsymbol{CD} \times \boldsymbol{CE}) >= 0$$

表示 E 点在 ***AB*** 和 ***CD*** 之间。同理，

$$(\boldsymbol{AC} \times \boldsymbol{AE}) \cdot (\boldsymbol{BD} \times \boldsymbol{BE}) >= 0$$

表示 E 点在 ***AC*** 和 ***BD*** 之间。因此，判断 E 点是否在矩形区域内只需判断上述两式是否同时成立即可。

遍历完毕后，得到的结果如表 2 和图 11 所示:

表 2: 每日各时间段船流量

日期	0-3 时	3-6 时	6-9 时	9-12 时	12-15 时	15-18 时	18-21 时	21-24 时
17 号	1	4	9	4	10	10	6	0
18 号	0	0	0	6	15	14	6	6
19 号	6	3	9	3	5	17	9	5
20 号	7	5	10	12	11	11	10	3
21 号	5	3	8	7	10	12	1	5
22 号	5	4	12	7	6	10	7	5
23 号	4	7	8	6	8	13	5	6
24 号	6	2	7	9	10	14	8	2
25 号	7	2	15	11	10	13	5	1

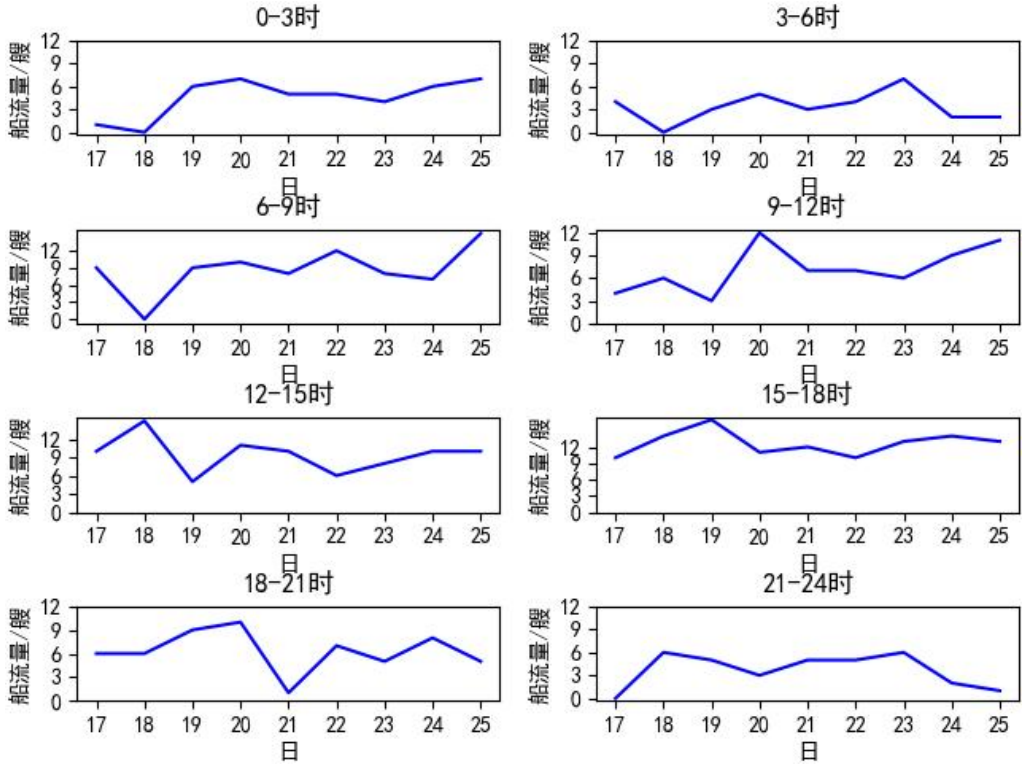

图 11: 各时间段船流量随日期变化趋势

7.2 预测模型选择

对于时间序列数据的预测，通常选择的模型有神经网络，ARIMA 模型，指数平滑模型等，然而本问题中已知数据量太少，每个时间段中已知数据只有 9 个(即 17 号到 25 号的每日各时间段船流量)，在这种情况下，应选用较简单的模型，因为在样本数量极少的情况下模型越复杂并不会带来预测效果的提升，反而效果会比简单的模型还差。首先排除使用神经网络的方法进行预测，这是因为神经网络需要大样本进行训练，另外，9 个已知数据对于 ARIMA 模型来说也远远不够，因此采用最简单的指数平滑模型。

指数平滑模型包括一次指数平滑法，二次指数平滑法，三次指数平滑法，其中一次指数平滑法针对没有趋势和季节性的序列，二次指数平滑法针对有趋势但没有季节性的序列，三次指数平滑法针对有趋势也有季节性的序列。考虑模型尽可能简单的原则，以及 9 个样本不可能看出来样本是否有季节性，因此选用最简单的一次指数平滑模型。需要注意的是虽然一次指数平滑模型只能对近期进行预测，但我们在本题中不需要对长期进行预测，只需对后一天进行预测，因此一次指数平滑模型完全适用于本题。

一次指数平滑模型主要思想是对不同时刻的观察值赋予不同的权重，即新信息比旧信息的权重要大，其递推关系如下:

$$s_i = \alpha x_i + (1-\alpha) s_{i-1}$$

其中 s_i 是第 i 个时刻经过平滑后的值，x_i 是第 i 个时刻的真实值，α是平滑系数，范围在 0 到 1 之间，其决定着新旧信息的占比情况。将递推关系展开后得到:

$$s_i = \alpha \sum_{j=0}^{i} (1-\alpha)^j x_{i-j}$$

由上面两式可以看出，共有两个参数需要人为调整，一个是平滑系数α，其值越大，则新信息占比越大，旧信息占比越小。另外一个参数是 s_0，由于 s_0 之前没有数据，因此 s_0 需要人为给定初始值，一般取前面几条数据的平均值。最后，设共有 n 条已知数据，因为此时没有第 n+1 时刻的真实值，那么对第 n+1 时刻数据的预测值取第 n 条数据的平滑值即可。设对第 n+1 时刻的预测值为 y_{n+1}，那么预测公式如下:

$$y_{n+1} = s_n$$

7.3 预测结果

一次指数平滑模型中共有两个参数需要人为选定，而我们有八个时间段，对每个时间段的预测需要分别选定参数，那么一共有 16 个参数需要人为设定，操作起来较困难，因此采用 python 自带的一次指数平滑模型库进行自动选择最优参数。

最终的预测结果如表 3 和图 12 所示:

表 3: 预测各时间段船流量

日期	0-3 时	3-6 时	6-9 时	9-12 时	12-15 时	15-18 时	18-21 时	21-24 时
26 号	7	3	9	7	9	13	6	4

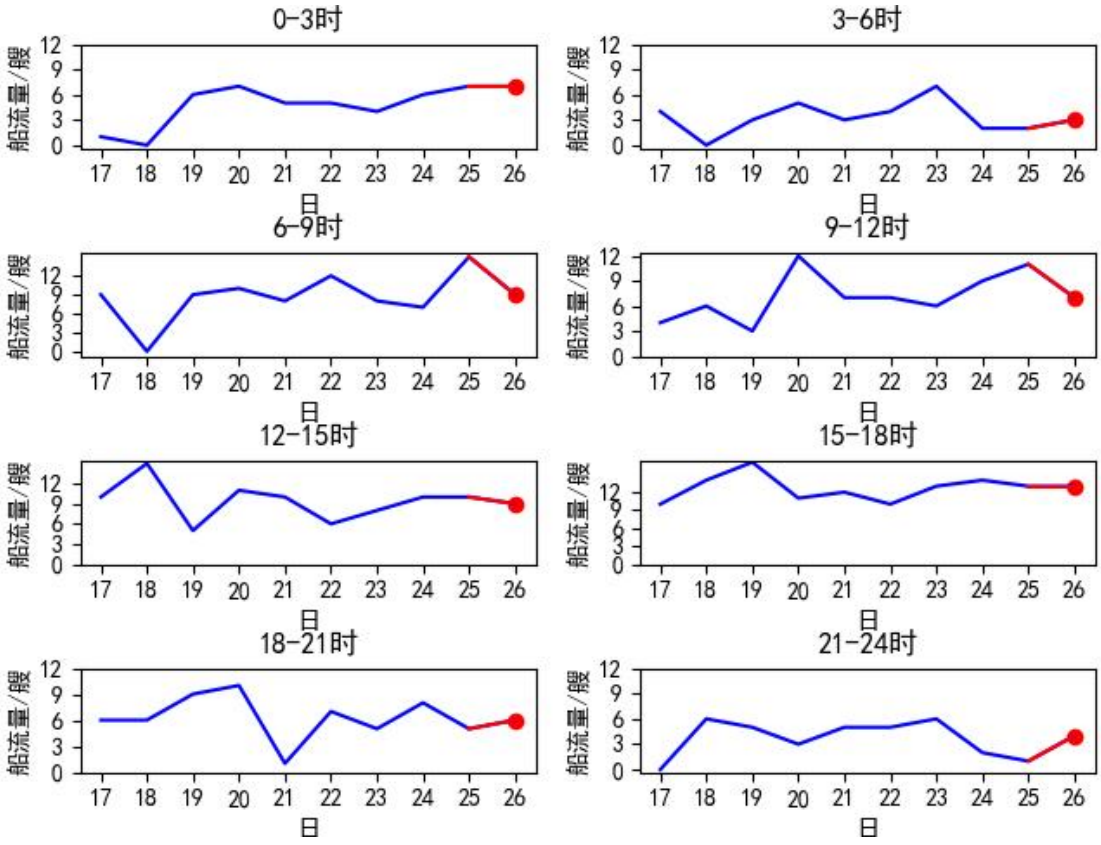

图 12: 各时间段船流量随时间变化趋势

由图 12 可以看出，预测结果基本符合大致趋势，预测结果较好。

8 模型总结

8.1 灵敏度分析

1. 第二问中数据压缩质量和结果对 $D_{\max}$ 的选取灵敏度较高，因此通过分析

关系曲线的方式选取了较合适的$D_{\max}$。

2. 第三问中数据预测对α和s_0的选取灵敏度较高，选取的值不同，最后的结果差别可能很大，解决办法是采用 python 自带的库进行自动选择最优参数。

8.2 模型评价

8.2.1 模型优点

1. 在问题一的求解中对异常点进行处理时将异常点分为两类，尽可能去除所有的异常点。

2. 在问题一的求解中对缺失点进行补充时也将缺失点分为了两类，对不同的缺失点做不同的处理，减少了误差。

3. 在问题二的求解中采用了经典的 Douglas-Peucker 算法，此算法已在广泛的应用中被证实压缩效率高，数据产生误差较小，因此能在此题中对船舶航行轨迹数据进行质量较高的压缩。

4. 在问题二的求解中对于$D_{\max}$的选取较为合理，在保证数据压缩率低的同时产生的误差也在合理范围内。

5. 在问题三中根据已知数据过少的问题选取了较简单的模型对未来值进行预测，可信度较高。

8.2.2 模型缺点

1. 在问题一的求解中对异常点进行处理时，未考虑船舶的航向，而实际上船舶的航向数据也有可能出错，但这对船舶轨迹影响较小，因此未考虑。

2. 在问题一的求解中未对第二类缺失点进行补充，数据重建后在数据集中仍存在缺失点。

3. 在问题二的求解中对数据压缩质量进行评价时未考虑压缩时间以及解压缩的难度，其实这两个指标也影响压缩质量，但由于难以评估，因此未考虑。

9 参考文献

[1] 张黎翔,朱怡安,陆伟,文捷,崔俊云. 基于 AIS 数据的船舶轨迹修复方法研究[J]. 西北工业大学学报, 2021,39(01):119-125.

[2] 郑宇. computing with spatial trajectories[M]. New York: Springer-Verlag, 2011.

10 附录

10.1 问题一主要代码

```
import pandas as pd
import numpy as np
from pandas import DataFrame
import os
from datetime import datetime, date
from math import radians, cos, sin, asin, sqrt
import os
import openpyxl
import codecs
from openpyxl.utils import get_column_letter
from openpyxl import Workbook


# 计算第一类噪声点
def stopPoint(file):
    data = pd.read_excel(file, header=None, index_col=None)
    # print(data)
    data_array = np.array(data)
    data_list = data_array.tolist()
    # print(data_list)
    neg_num = 0
    neg_List = []
    for i in range(len(data_list)):
        if i == 0:
            continue
        else:
            if (data_list[i][3] == data_list[i - 1][3]
                    and data_list[i][4] == data_list[i - 1][4]
                    and data_list[i][7] == data_list[i - 1][7]
                    and data_list[i][8] == data_list[i - 1][8]
                    and data_list[i - 1][7] > 2):
                neg_List.append(i)
                neg_num = neg_num + 1

    return neg_List, neg_num


# 计算时间差
def timeCalculate(t1, t2):
    t1_struct = (datetime.strptime(t1, "%Y-%m-%d %H:%M:%S"))
```

```
    t2_struct = (datetime.strptime(t2, "%Y-%m-%d %H:%M:%S"))
    total_seconds = (t2_struct - t1_struct).total_seconds()
    return total_seconds


# 计算距离
def distanceCalculate(x1, y1, x2, y2):
    x1, y1, x2, y2 = map(radians, [float(x1), float(y1), float(x2),
                                   float(y2)])
    dx = x2 - x1
    dy = y2 - y1
    a = sin(dy / 2) ** 2 + cos(y1) * cos(y2) * sin(dx / 2) ** 2
    c = 2 * asin(sqrt(a))
    r = 6371
    return c * r * 1000


# 寻找第二类噪声点
def neg2Point(file):
    data = pd.read_excel(file, header=None, index_col=None)
    data_array = np.array(data)
    data_list = data_array.tolist()
    neg_num = 0
    neg_List = []
    check = []

    for i in range(len(data_list)):
        if i == 0:
            check.append(True)
            continue
        else:
            j = i - 1
            while not check[j]:
                j = j - 1
            t1 = data_list[j][0] + ' ' + data_list[j][1]
            t2 = data_list[i][0] + ' ' + data_list[i][1]
            t = timeCalculate(t1, t2)
            dis = distanceCalculate(data_list[j][3], data_list[j][4],
                                    data_list[i][3], data_list[i][4])
            v = dis / t
            if v*1.94 <= 2*max(data_list[i][7], data_list[j][7]):
                check.append(True)
            else:
                check.append(False)
                neg_num = neg_num + 1
```

```
                neg_List.append(i)
    # print(file)
    # print(neg_List)
    print(neg_num)
    return neg_List, neg_num
def timeCalculate(t1, t2):
    t1_struct = (datetime.datetime.strptime(t1, "%Y-%m-%d %H:%M:%S"))
    t2_struct = (datetime.datetime.strptime(t2, "%Y-%m-%d %H:%M:%S"))
    total_seconds = (t2_struct - t1_struct).total_seconds()
    return total_seconds

#缺失点
def processLose(filename, outfile, sheetname):
    df = pd.read_csv(filename, header=None, sep=" ")
    data_array = np.array(df)
    data_list = data_array.tolist()
    fr = codecs.open(filename, 'r')
    wb = openpyxl.Workbook()
    ws = wb.active
    # ws = wb.create_sheet()
    ws.title = sheetname
    row = 0
    real_row = 0
    for line in fr:
        row += 1
        if row == 1:
            real_row += 1
            line = line.strip()
            line = line.split(' ')
            col = 0
            for j in range(len(line)):
                col += 1
                # print (line[j])
                if (col != 1) and (col != 2) and (col != 10):
                    ws.cell(column=col, row=real_row, value=line[j]\
                        .format(get_column_letter(col)))\
                        .data_type = "int"
                else:
                    ws.cell(column=col, row=real_row, value=line[j]\
                        .format(get_column_letter(col)))
        else:
            t1 = data_list[row - 1][0] + ' ' + data_list[row - 1][1]
            t2 = data_list[row - 2][0] + ' ' + data_list[row - 2][1]
            t = timeCalculate(t2, t1)
            if t <= 360 or t >= 7200:
```

```
            real_row += 1
            line = line.strip()
            line = line.split(' ')
            col = 0
            for j in range(len(line)):
                col += 1
                # print (line[j])
                if (col != 1) and (col != 2) and (col != 10):
                    ws.cell(column=col, row=real_row,
                            value=line[j]\
                            .format(get_column_letter(col))) \
                        .data_type = "int"
                else:
                    ws.cell(column=col, row=real_row, value=line[j]\
                    .format(get_column_letter(col)))
        else:
            n = (int)(t / 360)
            for i in range(1, n + 1):
                real_row += 1
                ws.cell(column=1, row=real_row,
                        value=data_list[row - 2][0])
                ws.cell(column=2, row=real_row,
                        value=((datetime.datetime\
                        .strptime(data_list[row - 2][1], "%H:%M:%S")
                         + datetime.timedelta(seconds=i * 360))\
                         .strftime("%H:%M:%S")))
                ws.cell(column=3, row=real_row,
                        value=data_list[row - 2][2])
                ws.cell(column=4, row=real_row,
                        value=(data_list[row - 1][3] -
                               data_list[row - 2][3])
                        / t * 360 * i + data_list[row - 2][3])
                ws.cell(column=5, row=real_row,
                        value=(data_list[row - 1][4] -
                               data_list[row - 2][4])
                        / t * 360 * i + data_list[row - 2][4])
            col = 0
            line = line.strip()
            line = line.split(' ')
            real_row += 1
            for j in range(len(line)):
                col += 1
                # print (line[j])
                if (col != 1) and (col != 2) and (col != 10):
                    ws.cell(column=col, row=real_row,
```

```
value=line[j].format(get_column_letter(col))) \
                            .data_type = "int"
                    else:
                        ws.cell(column=col, row=real_row, value=line[j]\
                             .format(get_column_letter(col)))
        wb.save(outfile)
```

10.2 问题二主要代码

```
    def distanceCalculate(x1, y1, x2, y2):
        x1, y1, x2, y2 = map(radians, [float(x1), float(y1)
                        , float(x2), float(y2)])
        dx = x2 - x1
        dy = y2 - y1
        a = sin(dy / 2) ** 2 + cos(y1) * cos(y2) * sin(dx / 2) ** 2
        c = 2 * asin(sqrt(a))
        r = 6371
        return c * r * 1000


    def Geodist(point1, point2):
        return distanceCalculate(point1[1], point1[2]
        , point2[1], point2[2])


    def get_vertical_dist(pointA, pointB, pointX):
        a = math.fabs(Geodist(pointA, pointB))
        if a == 0:
            return math.fabs(Geodist(pointA, pointX))
        b = math.fabs(Geodist(pointA, pointX))
        c = math.fabs(Geodist(pointB, pointX))
        p = (a + b + c) / 2
        S = math.sqrt(math.fabs(p * (p - a) * (p - b) * (p - c)))
        vertical_dist = S * 2 / a
        return vertical_dist


    def DP_compress(point_list, output_point_list, Dmax):
        start_index = 0
        end_index = len(point_list) - 1
        output_point_list.append(point_list[start_index])
        output_point_list.append(point_list[end_index])

        if start_index < end_index:
            index = start_index + 1
```

```
        max_vertical_dist = 0
        key_point_index = 0

        while (index < end_index):
            cur_vertical_dist = get_vertical_dist(point_list[start_index]
            , point_list[end_index], point_list[index])
            if cur_vertical_dist > max_vertical_dist:
                max_vertical_dist = cur_vertical_dist
                key_point_index = index
            index += 1

        if max_vertical_dist >= Dmax:
            DP_compress(point_list[start_index:key_point_index]
            , output_point_list, Dmax)
            DP_compress(point_list[key_point_index:end_index]
            , output_point_list, Dmax)


def cal_Err(point_list, output_point_list):
    Err = 0

    start_index = 0
    end_index = len(output_point_list) - 1

    while (start_index < end_index):
        pointA_id = int(output_point_list[start_index][0])
        pointB_id = int(output_point_list[start_index + 1][0])

        id = pointA_id + 1
        while (id < pointB_id):
            Err += get_vertical_dist(output_point_list[start_index],
                     output_point_list[start_index + 1], point_list[id])
            id += 1

        start_index += 1

    return Err


def work(infile, outfile, Dmax):
    point_list = []
    output_point_list = []

    fd = open(infile, 'r')
    id = 0
```

```
    for line in fd:
        line = line.strip()
        id += 1
        longitude = float(line.split(" ")[3])
        latitude = float(line.split(" ")[4])
        point_list.append((id, longitude, latitude))
    fd.close()

    DP_compress(point_list, output_point_list, Dmax)

    output_point_list = list(set(output_point_list))
    output_point_list = sorted(output_point_list, key=lambda x: x[0])
    fd = open(outfile, 'w')
    for point in output_point_list:
         fd.write("{},{},{}\n".format(point[0], point[1], point[2]))
    fd.close()
    return len(point_list), len(output_point_list),
           cal_Err(point_list, output_point_list)
```

10.3 问题三主要代码

```
import os
import numpy as np
from datetime import *
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
import matplotlib.pyplot as plt
from pylab import *
def gethour(s):
    time = datetime.strptime(s,"%H:%M:%S")
    return time.hour


def GetCross(x1,y1,x2,y2,x,y):
    a=(x2-x1,y2-y1)
    b=(x-x1,y-y1)
    return a[0]*b[1]-a[1]*b[0]
def isInSide(x1,y1,x2,y2,x3,y3,x4,y4,x,y):
    return GetCross(x1,y1,x2,y2,x,y)*GetCross(x3,y3,x4,y4,x,y)>=0
      and GetCross(x2,y2,x3,y3,x,y)*GetCross(x4,y4,x1,y1,x,y)>=0

def cal(inflie):
    f=open(inflie,'r')
    lis=[0,0,0,0,0,0,0,0]
    for line in f:
        line=line.strip()
```

```
        time=line.split(" ")[1]
        longitude = float(line.split(" ")[3])
        latitude = float(line.split(" ")[4])
        if isInSide(114.28354, 30.55238, 114.28341, 30.55218,
          114.29277, 30.54731, 114.29265, 30.54711
          ,longitude, latitude):
            hour=gethour(time)
            id=hour//3
            if lis[id]==0:
                lis[id]+=1
    return lis
mpl.rcParams['font.sans-serif'] = ['SimHei']
import pandas as pd
res=[[1, 4, 9, 4, 10, 10, 6, 0], [0, 0, 0, 6, 15, 14, 6, 6],
     [6, 3, 9, 3, 5, 17, 9, 5], [7, 5, 10, 12, 11, 11, 10, 3],
     [5, 3, 8, 7, 10, 12, 1, 5], [5, 4, 12, 7, 6, 10, 7, 5],
     [4, 7, 8, 6, 8, 13, 5, 6], [6, 2, 7, 9, 10, 14, 8, 2],
     [7, 2, 15, 11, 10, 13, 5, 1]]
change=[]
for i in range(0,8):
    lit=[]
    for j in range(0,9):
       lit.append(res[j][i])
    change.append(lit)
for item in change:
    fit1 = SimpleExpSmoothing(item).fit()
    print(fit1.summary())
    item.append(int(np.round(fit1.forecast(1))))
date = ['17','18','19','20','21','22','23','24','25','26']
tit=['0-3 时','3-6 时','6-9 时','9-12 时','12-15 时',
     '15-18 时','18-21 时','21-24 时']
id = [1,2,3,4,5,6,7,8,9,10]
y=range(0,15,3)
fig=plt.figure(1)
for i in range(0,8):
    temx=[[id[8],id[9]]]
    temy=[[change[i][8],change[i][9]]]
    print(change[i][9])
    ax1=plt.subplot(4,2,i+1)
    plt.plot(id,change[i],color='b')
    plt.plot(temx[0], temy[0], color='r')
    plt.scatter(id[9], change[i][9], color='r')
    plt.xlabel('日')
    plt.ylabel('船流量/艘')
    plt.yticks(y)
```

```
    plt.xticks(id, date)
    plt.title(tit[i])
plt.show()
```