

2016 年同济大学数学建模校内竞赛论文

队员 1
队员 2
队员 3

80

基于智能导览系统的游乐园客流疏导方案

摘要

随着近年来国内大型游乐园的快速发展，如何有效解决园区内客流拥堵问题并同步完善园区周边配套设施已成为人们关注的热点。传统的游乐园导览方式，通过向游客发放园区指南，为其提供多条游园路线以供自行选择，从而达到全路线控制分流的目的；但由于游客选择的随机性，固化的路线无法适应动态变化的客流，效果并不理想。而后来兴起的“时空分流导航”系统，借助于信息技术实现对园区内客流分布的实时监测，以各项目负荷均衡为目标，制定游客分流方案；虽然客流“扎堆”得以缓解，但游客满意度却不尽如人意。基于以上情况，本文通过建立数学模型解决 Youth 游乐园相关问题，使游客游园体验最佳。

针对问题（一）中万人大客流的疏导，本文从游客角度出发解决问题。将最优调度思想建立在游客的游园体验上，联系实际引入“游览所需代价”将游园体验量化，并构建“基于最小游览代价”的决策模型；借助于智能导览系统，根据每位游客游玩的实际情况，获取园内实时客流分布以及游乐项目信息，经决策模型计算得到最优的游园路径，并通过电子导航器将最优路径信息反馈给游客。模型逐步产生的推荐路线即构成了个性化的动态实时调度方案，能让游客自动散布到不太拥挤的游乐项目上，在保障游园体验的前提下解决了客流拥堵问题。为了检验模型的灵敏度，本文在 Visual C++ 平台上对于该方案进行仿真分析。实验结果表明，模型在路径的多样性、游客等待时间和路遇排队人数三个指标上均展现出良好性能。

针对问题（二）中 16 年酒店预定量预测问题，本文首先对皇冠假日酒店 2015 年全年的预定信息进行分析处理，联系实际提出预定量的可能影响因素，并通过查找相关资料，利用 EXCEL 做出预定量和可能影响因素随时间的分布图，通过初步分析，得到预定量的影响因素是季节、经济状况和法定节假日及双休；由于酒店预定量影响因素多，且这些因素对预定量的作用均表现出明显的灰色性，本文采用灰色关联度分析方法对酒店影响因素进行剖析，确定三者的影响程度基本相当；最后，根据传统时间序列预测方法，并结合前文所得主要影响因素，本文采用指数平滑方法对数据进行拟合，得到 16 年 1 月至 3 月每天房间预定量。

关键词：最优动态调度 决策支持 灰色关联度分析法 时间序列预测方法

一. 问题重述

近年来，迪士尼、欢乐谷等众多大型游乐园纷纷入驻中国，因其精彩纷呈的娱乐盛宴而受到人们的热情追捧。随着大型游乐园的快速发展，成千上万的游客同时入园游玩，也使得园区内负载不均衡等问题日益明显：如游乐园某些热门项目，游客们往往需要长时间排队等待；而与此同时有些项目因为地理位置等原因却无人问津——这既降低了游客的游园体验，也限制了园区总体容量得到充分利用。此外，游乐园周边相关配套设施的同步完善，如地铁、住宿、餐饮等等，也是近年来游乐园管理者和游客共同关注的热点。因此，制定高效的游客分流方案、结合已有数据对未来园区内酒店房间预定量进行预测具有重要意义。

Youth 游乐园即将盛大开园，届时园区将迎接每天一万人的大客流。基于以上信息，为了给游客提供全方位最优的服务，我们需要解决以下两方面问题：

（一）保障游客体验游乐设施的前提下，制定客流分流方案，使游园体验最优

1. 分析决定游客游园体验的影响因素，量化游园体验。
2. 以游园体验最佳为目标，建立相应的决策模型。利用以该模型为核心的智能导览系统，为每个游客生成个性化的疏导方案，以解决园区内客流集中的问题。
3. 通过计算机仿真模拟，分析决策模型在实际运行中情况。

（二）分析皇冠假日酒店 2015 年预定数据，预测 16 年 1 月至 3 月每日预定量

1. 利用 EXCEL 处理 2015 年该酒店预定量数据，分析可能对其产生影响的因素，并绘制相应图表做出相应分析。
2. 运用灰色关联分析方法研究可能的影响因素，找出重要影响因素。
3. 结合重要影响因素，建立数学模型，借助 SPSS，预计 16 年 1 月-3 月每天预定量。

二. 问题分析

问题（一）游园客流疏导方案制定

面对园中每天 1 万的大客流，游乐园管理者希望客流能在园中各处娱乐设施相对均匀分布，将客流压力分散到个点，从而避免由于园内某个别设施人数过多而造成的人员拥挤、排队时间过长、游乐设施故障频发等问题的发生。这一般是借助于“时空分流导航”管理来完成的：借助信息监控技术，设计出一条或若干条优化游览路线，利用随时间移动形成的相对“空置”、“空闲”的空间对客流进行分流导航。其重点在于考察园区内客流的分布情况和超负荷项目，以园区内各项目的负荷均衡为目标对项目负荷进行动态预测。在此基础上制定一定的方案实现各项目之间的客流的调度。该决策模型虽然能较为有效的使项目负荷达到最优，但由于模型是建立与项目负荷上的，并不一定能使游客满意程度，即游园体验最优。

问题（二）酒店房间预定量的影响因素探究及未来预定量预测

三. 模型假设

1. 游乐园内各游乐项目周围步行道路的容量足够。
2. 不同游客在园内步行速度相同，均为 1m/s 。
3. 每个游客在游园过程中，每个项目都玩到且均只游玩一次。
4. 不同游客在相同游览项目处所花时间相同，均为游乐项目一次持续时间。
5. 该游乐园采用智能导览系统，并为每个游客提供游园导航器。

1. 不考虑 2015 年 1 月 1 日之前下的订单，对于 2015 年全年预定量的影响。
2. 假设主要影响因素，对 2015 年和 2016 年酒店房间预定量的影响基本相同。
3. 假设 Youth 游乐园位于上海。

四. 符号说明

S_i 项目 $i(i=A,B,C,D,E,F,G,H,I,J)$

p_{ij}	从项目 i 到项目 j 所需的游览代价
b_i	项目 i 处每场容纳游客数
W_i	某时刻项目 i 处排队等待的游客数
d_{ij}	项目 i 到项目 j 的步行距离
$T_{walk_{ij}}$	从项目 i 步行到项目 j 所需的时间
T_{f_i}	项目 i 下一次空闲的时刻
T_{p_i}	游客在项目 i 处游玩所需时间
T_{w_i}	游客在项目 i 处排队等待所花时间
T	当前时间
$xo(t)$	参考序列
$xi(t)$	比较序列
$\xi_{oi}(t)$	$xo(t)$ 和 $xi(t)$ 在时刻 t 的关联系数
r_{oi}	$xo(t)$ 和 $xi(t)$ 的灰色关联度

五. 模型建立与求解

5.1 问题（一）游园客流疏导方案制定

5.1.1 确定影响游客游园体验的影响因素并量化游园体验。

如前所述，一个大型游乐场的客流疏导方案不能只考虑游乐园管理者最关心的“如何解决游乐园各游乐项目客流负荷不均匀问题”，而更应当站在游客的角度上，去考虑如何提高游客的游园体验。在现实中，一个对游客至关重要的问题是：如何在最短的时间内游玩所有自己感兴趣的游乐项目？在大型游乐园中，由于游客时常碰到因客流拥堵而造成的排队现象，而在每个游乐项目游玩所花时间 T_{p_i} 又相对固定——等于每个游乐项目每场所持续的时间，最短的游玩时间主要

取决于排队等待所花时间 T_{w_i} ($i=A,B,C,D,E,F,G,H,I,J$) 和在各项目之间步行所花费时间 $T_{walk_{ij}}$ ($i,j=A,B,C,D,E,F,G,H,I,J$)。我们将从当前所处项目 S_i 到完成下一步目标项目 S_j 的游玩所需要的时间, 定义为从项目 i 到项目 j 所需的游览代价 p_{ij} , 有

$$P_{ij} = f(T_{walk_{ij}}, T_{w_j}, T_{p_j}) \quad (1)$$

显而易见, 当游乐园为游客提供适当的导览方案, 如地图等, 将有效地降低从项目 i 到项目 j 所需的游览代价 p_{ij} 。传统的导览方式是通过发放游客指南, 采用全路线控制协调方法对游客进行分流, 游乐园为游客提供多类游览路线, 让游客自行选择合适的方案, 并按照这个行程参观。但是这样固化的路线一方面难以适应动态变化的客流分布, 另一方面由于游客路线选取的随机性, 仍然可能造成客流扎堆现象的发生, 效果并不理想。为弥补这个不足, 本文采用目前较为先进的“智能导览系统”进行讨论。智能导览系统是指一种利用现代信息技术, 实现对园区内客流分布、工作人员分布、游乐项目运行状况等信息的实时感知, 通过智能运算获得游客游园最佳路线, 并最终利用电子设备反馈给游客的系统。基于该系统, 每个项目实时的排队人数 W_i 和某游乐项目下一次空闲时间 T_{f_i} 可以通过技术手段获取。我们通过下表, 对游览代价进行分析。

表格 1 对于某个未游览项目的游览代价分析

对某个未游玩项目 S_j 的游览代价作如下判断:

如果此刻该项目空闲, 则所需代价= $T_{walk_{ij}}$

否则, 所需代价= $\text{Max}(\text{景点下一次空闲时间 } T_{f_j} - \text{当前时间 } T + \text{景点游玩用时 } T_{p_j} \times \text{排队等候人数 } W_j / \text{每轮所能容纳人数 } b_j, T_{walk_{ij}})$

把上面两种情况综合起来, 则每一步的游览代价可以被量化, (1)式具体表达为:

$$p_{ij} = \max(T_{f_j} - T + T_{p_j} \cdot W_j / b_j, T_{walk_{ij}}) \quad (2)$$

当每一步的游览代价越小时, 游客的游园满意度就越高。借助于对游览代价 p_{ij} 高低的分析, 我们可以对不同路线下的游园体验进行定量分析比较。

5.1.2 建立最小游览代价决策模型并制定动态分流方案

构建基于最小游览代价的决策模型的关键在于: 当游客入园以后, 或者游客游览完当前项目后, 该决策模型可以根据此刻客流在各未游览项目的分布情况, 结合行走到达该候选项目的排队人数、排队等候人数、候选项目游玩耗时、候选项目容量等信息, 分别计算出该游客对各个候选游览项目所需要付出的游览代价, 进行

排序,最终选择其中一个游览代价最小的项目,作为下一个游览项目推荐给游客。基于最小游览路线的决策模型旨在通过每一步推荐的路线,将游客自动地分配到相对不太拥挤的设施上,从而达到客流疏导、智能导览的目的。决策模型的算法框图如下:

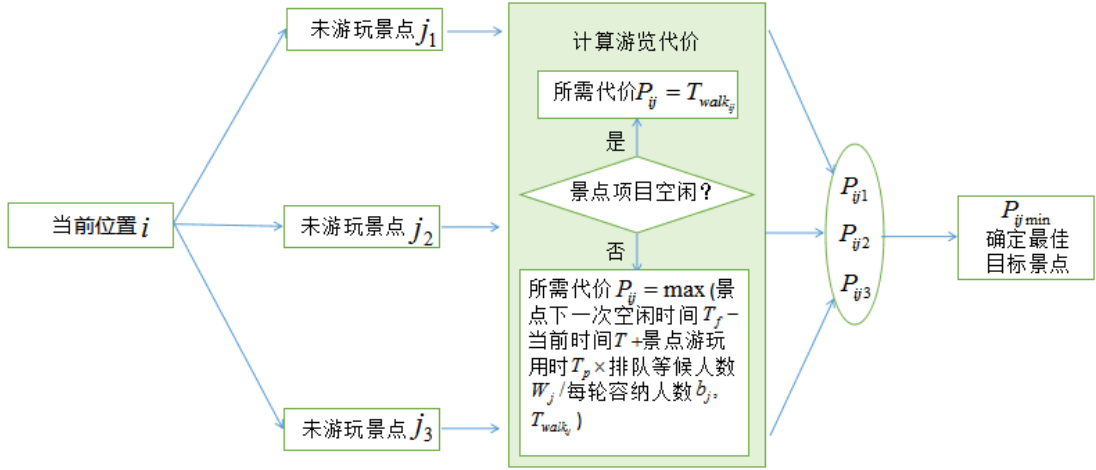


图 1 基于最小游览代价的决策模型算法

不同游客到达游乐园的时间是随机的,而不同的到达时间对应着游乐园内不同的客流分布情况。因此该智能导览系统可以为各个时刻到达游乐园的游客,提供动态变化的最佳游园路线。具体的动态分流方案算法如表 2 及其流程图如图 2。

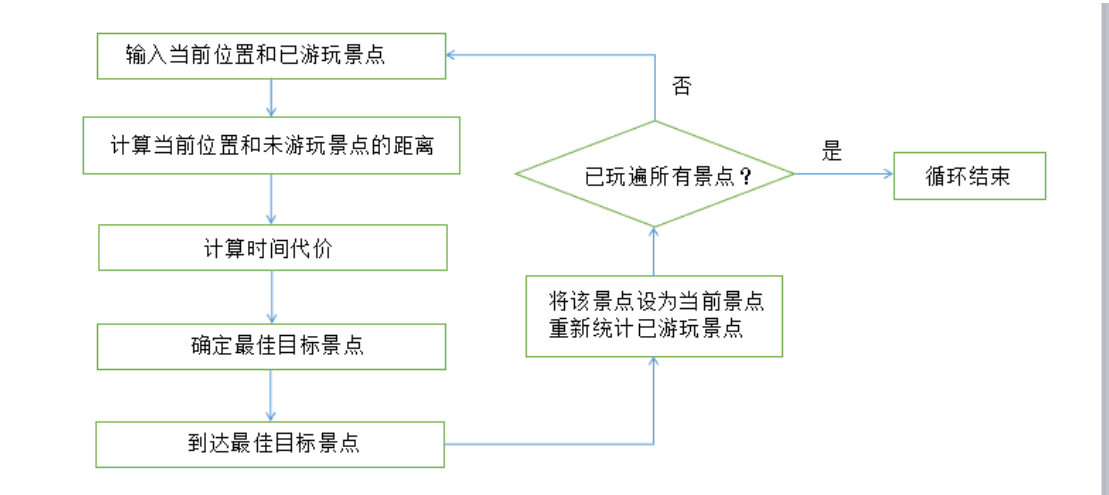


图 2 基于最小游览代价决策模型的动态分流方案算法流程图

表格 2 基于最小游览代价决策模型的动态分流方案

◆	当游客入园时,生成“到达事件”,并以该事件发生的时间作为标记,对一系列的到达时间排序。
◆	做以下循环,直到所有的项目都被游玩。
●	如果是“到达事件”

-
- 记录当前位置，设为 A。
 - 利用上述“基于最小游览代价”的决策模型，计算出当前最快可游玩的项目，记为 B。
 - 生成“待游玩事件”，其中事件发生时间=当前时间 T +行走时间 $T_{walk_{AB}}$ 。
 - 如果是“待游玩事件”（游客刚到达景点或者正在排队等待进入）
 - 如果该项目此时空闲（游客可以进入游玩）， $T_{wait}=0$ ；
 - 否则，等待时间=项目下一次完成服务时间-当前时间+排队等待所需时间，即： $T_{wait} = T_f - T + T_p \cdot W_i / b_i$
 - 设置完成游玩时间=景点游玩用时 T_f +等待时间 T_{wait}
 - 生成“完成游玩事件”，其中事件发生时间=完成游玩时间。
 - 如果是“完成游玩事件”
 - 设置景点完成服务时间 T_f ：如果没有游客等候，则设置为 0；否则，将其设置为当前时间+景点游玩用时： $T_f = T + T_p$ 。
 - 记录当前位置，设置为 A。
 - 利用上述“基于最小游览代价”的决策模型，计算出当前最快可游玩的项目，记为 B。
 - 生成“待游玩事件”，其中事件发生时间=当前时间 T +行走时间 $T_{walk_{AB}}$ 。
-

注：表中 W_i 为项目 S_i 实时排队人数，在实际的中我们通过 RFID（无线射频识别技术获取），但是在后文的模拟中，我们利用计算机科学领域中的“队列”结构来模拟，通过队列元素个数即可获知排队等待人数。

5.2 问题（二）酒店房间预定量的影响因素探究及未来预定量预测

5.2.1 影响房间预定量的主要因素初探

皇冠假日酒店位于游乐园内，开园后往来游客中需要住宿的人群将成为其主要顾客群。而 Youth 游乐园作为大型休闲娱乐场所，因其过山车种类繁多而特别受到青少年的追捧，不难想到其客流量与人们消费水平、季节天气、学生寒暑假及法定节假日与双休息息息相关。联系生活实际，我们初步推测影响该酒店房间预定量的可能因素有 4 个：季节、当地经济状况、暑假和法定节假日与双休。根据附件 2 中提供的 2015 年酒店预订数据，利用 EXCEL 对数据进行处理，分析全年预订量随时间的分布情况；再查找有关资料，利用 EXCEL 绘制各影响因素在 2015 年中随时间的分布图；比较房间预定量和影响因素随时间的变化趋势，初

步确定影响房间预定量的有关因素。

第一步，对 2015 年酒店预订信息进行处理。顾客预订酒店一般都早于实际入住日期，最晚发生在入住当天；而预定量又直接反映了顾客对于入住酒店那段时间可能发生事件的一种心理预估。因此，想要分析影响酒店预定量的因素，必须将每个游客的预定量与实际入住酒店的那段时间（到店日期与离店日期之间的时间段）结合起来进行分析，即分析“被预约占用的房间数”与“预计入住时间”的关系。简而言之，就是分析房间预定量与实际入住时间之间的关系。为避免歧义发生，本文在此阐明：本节（5.2.1）中所使用的“房间预定量”指某时刻酒店“被预约占据的房间数”。由于每名顾客从入住开始到离店都占据酒店房间，故均计入预定量。按照此原则，统计出 2015 年全年酒店每天房间预定量，绘出预定量全年分布图，并计算出同期房间预定量的增长率。绘制下图。

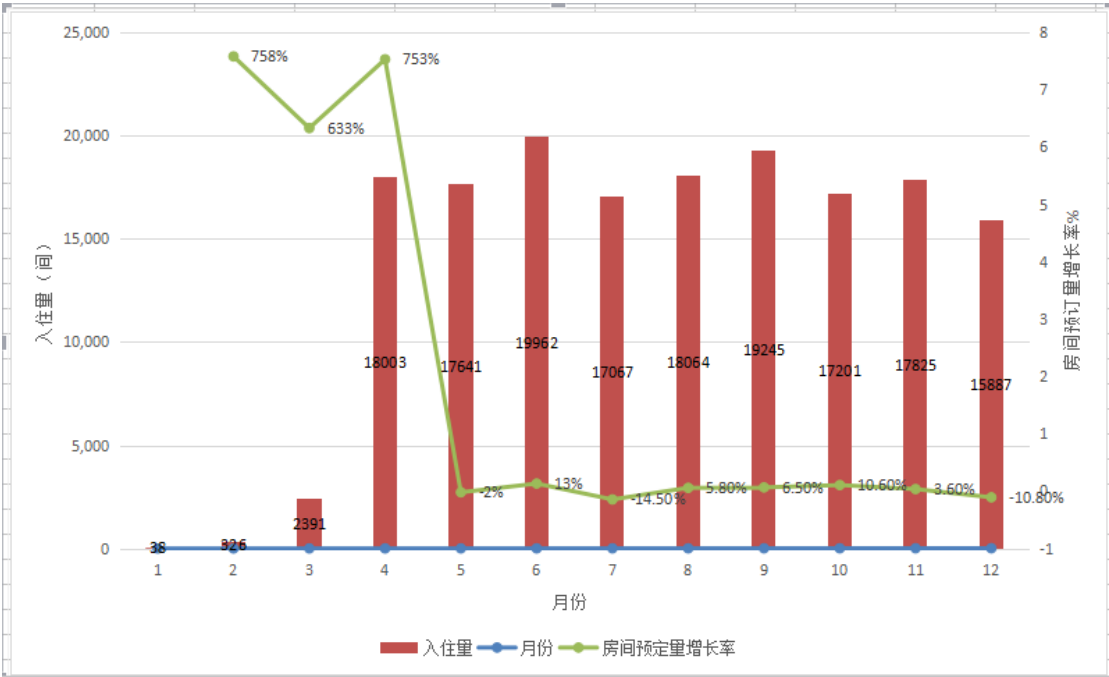


图 3 2015 年皇冠假日酒店月房间预定量及其增长率分布图

从图 1 中可以明显看出，四月以后房间预定量趋于稳定，处于轻微波动状态；可前三个月的房间预定量却显著低于后面几个月，每月大于 600%的预定量增长率（最高达到 758%）令人不解，并引起了我们的注意。

第二步，查找相关影响因素的资料。假设 Youth 游乐园位于上海，我们查找了 2015 年上海市全年温度信息、上海市居民消费价格指数情况、大学生暑假时间段以及 2015 年法定节假日及双休分布时间。分别作季节（月平均温度）/房间预定量---月份分布图、法定节假日与双休（每月放假天数）/房间预定量---月份分布图、暑假/房间预定量---日期分布图，以及经济水平（居民消费价格指数）/房间预定量---月份分布图，已进行进一步的分析。

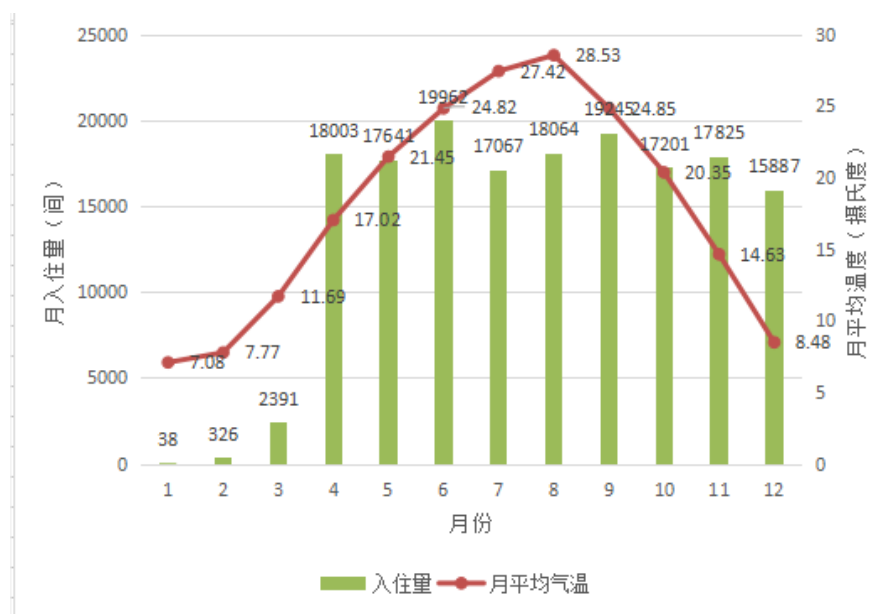


图 4 季节（月平均温度）/房间预定量---月份分布图

从上图可以看出，当温度特别低（冬季 1 月、2 月，3 月）酒店房间预定量明显低于其他时期的水平；随着春天的到来，气温逐渐回暖（4 月、5 月、6 月），酒店的房间预定量有逐步上升的趋势；而在酷夏时节（7 月、8 月、9 月），酒店的房间预定量与前期比较有所回落，可以解释为气温居高（特别是平均气温高于 25℃ 的 7 月、8 月），人们减少了外出游玩；而进入秋季（10 月、11 月、12 月）之后，气温逐渐回落，当气温由高温降到较为宜人的（15℃-20℃）之间时，酒店的房间预定量有所回升，但随着气温进一步下降（15℃ 之下时），酒店预定量又开始下降。通过对数据的分析，我们可以看出，季节对于酒店的房间预定量存在一定影响。当温度在 15℃ 至 25℃ 之间时，酒店的房间预定量处于高峰期。

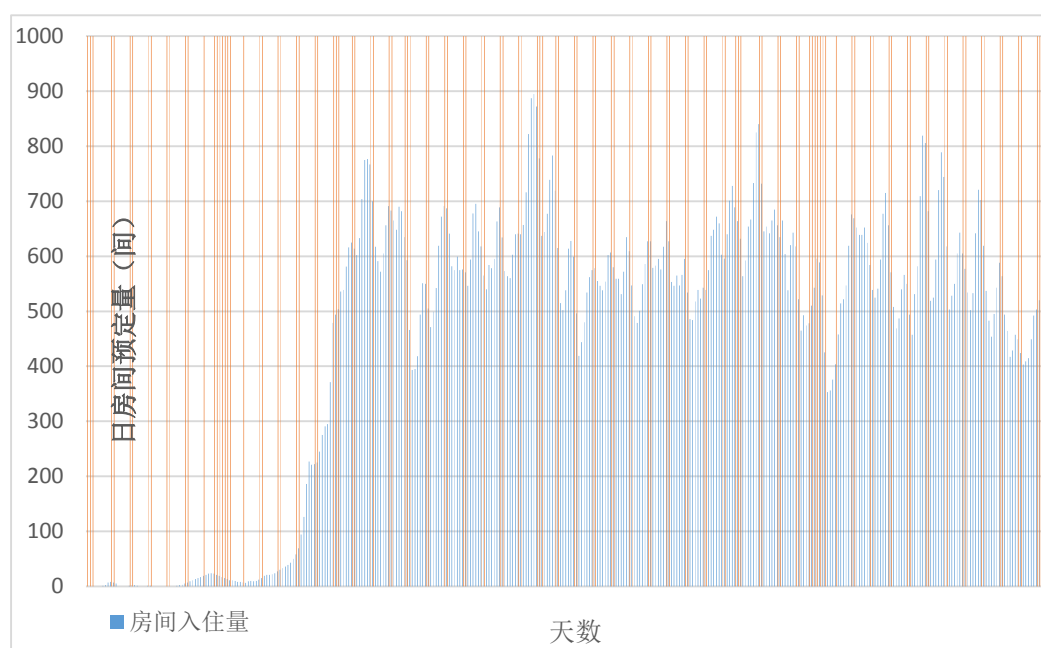


图 5 法定节假日与双休/房间预定量---日期分布图

图 5 中，橙色线条表示法定节假日及双休，蓝色线条表示酒店每天的房间预定量。从图中我们不难发现，日预定量的峰值（蓝色）与节假日（橙色）十分接近甚至重合。为了更加清晰分析节假日与非节假日对于房间预定量的影响，我们对数据进行了进一步的处理，作“假期中日平均房间预定量/非假期平均房间预定量---月份分布图”（如下），从图中我们可以清楚的看出绝大多数情况下，假期里的平均房间预定量要大于非假期里的。由此，我们得到结论，法定节假日与双休会对酒店的房间预定量产生影响。

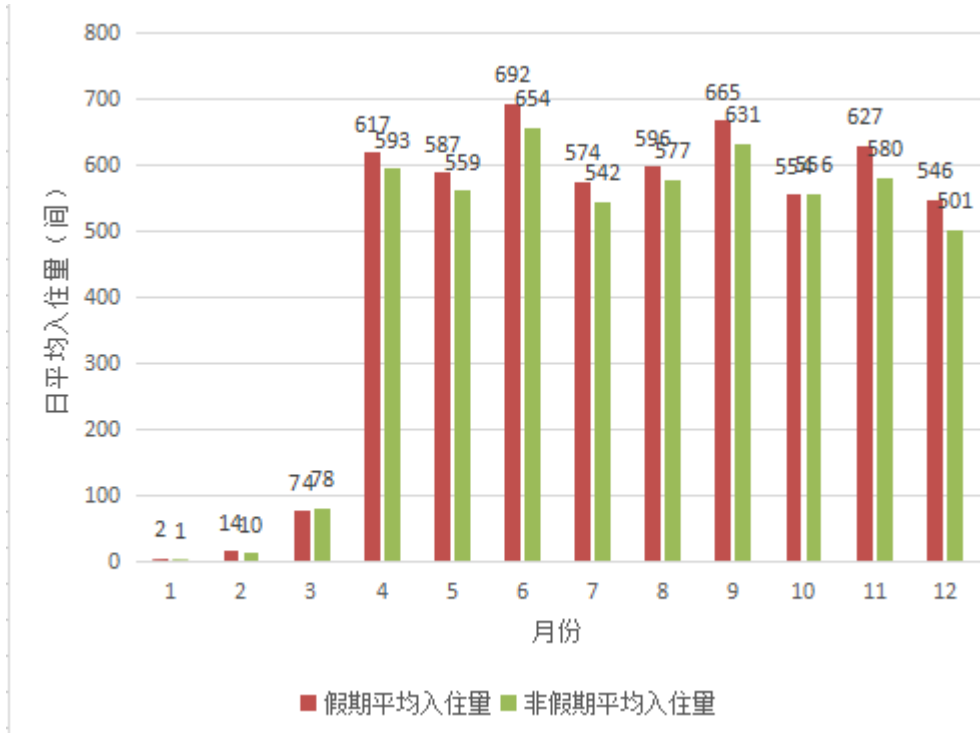


图 6 假期中日平均房间预定量/非假期平均房间预定量---月份分布图



图 7 暑假/房间预定量日期分布图

转观“暑假/房间预定量日期分布图”，在暑假的两个月中并没有迎来一个房

间预定量的高峰。据此，我们推断：与暑假联系紧密的学生教师群体并不是构成酒店顾客的主要人群，故可以视该因素对于酒店预定量没有明显影响。

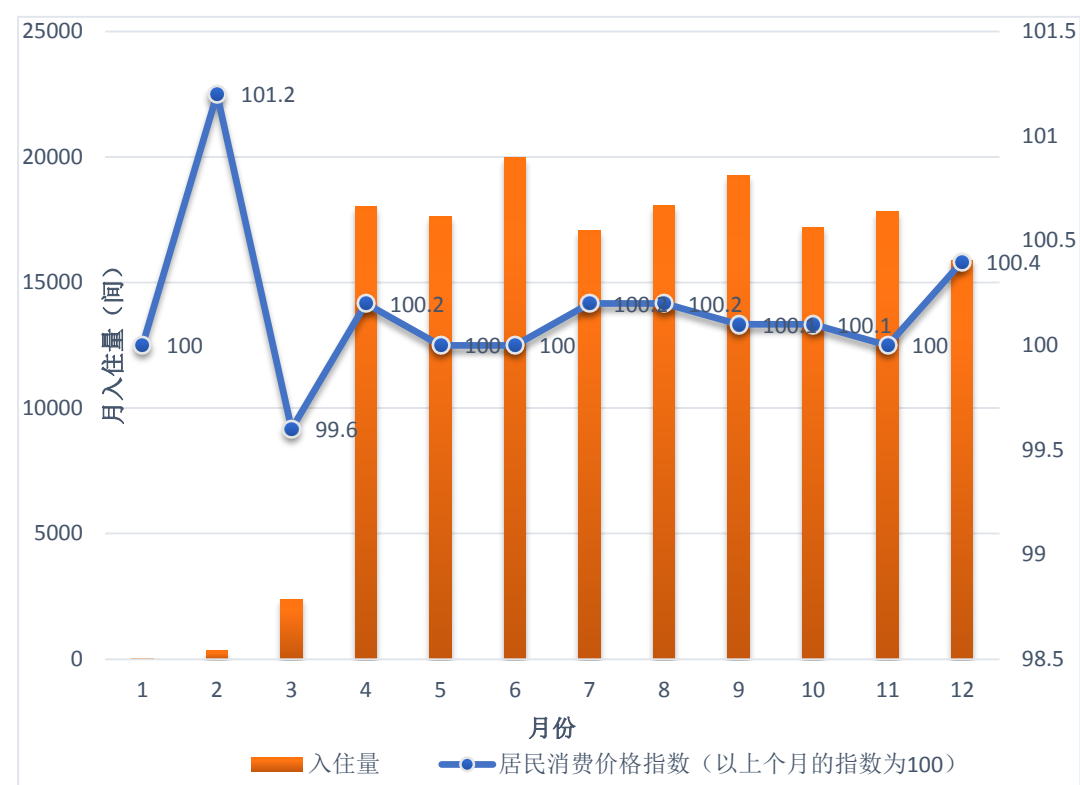


图 8 经济水平（居民消费价格指数）/房间预定量---月份分布图

居民消费价格指数（**Consumer Price Index**），是一个反映居民家庭一般所购买的消费商品和服务价格水平变动情况的宏观经济指标。它是度量一组代表性消费商品及服务项目的价格水平随时间而变动的相对数，是用来反映居民家庭购买消费商品及服务的价格水平的变动情况。经济学家认为，消费者物价指数上涨，货币购买力则下降；反之则上升。从一定程度上，CPI 反映了物价的高低情况。在 2015 年上海市居民月平均工资不大的情况下，CPI 升高，说明物价升高，人们愿意花费在娱乐（如酒店住宿和逛游乐园）上的消费自然下降，因而预定量下降；当 CPI 上升时，则人们有能力负担更多的娱乐消费，因此酒店预订量上升。比较房间预定量和 CPI 随时间的趋势图，基本符合这个规律（除去 4 月）。因此，我们认为经济水平也从一定程度上影响了酒店房间的预定量。

综合上述分析，我们得到如下结论：

1. 初步确定了房间预定量的影响因素有季节（月平均气温）、法定节假日及双休和经济水平。
2. 暑假对于该酒店预定量的影响并不明显，可能是由于与暑假联系紧密的学生教师群体并不是构成酒店顾客的主要人群。
3. 人们更倾向于在温度较宜人的时候（15℃-25℃）外出游玩并入住酒店，春季和秋季是酒店房间预订量的高峰期，冬季是该酒店预定量的低谷期。
4. 对于房间预定量分布图中 1~3 月预定量明显低于其他时期，我们认为有 4 个可能的原因：①受春节的影响巨大传统的中国家庭都会选择在春节

及前后回到家中和家人团聚，而不是外出游玩入住酒店②该段时间天气寒冷，较大的减少了人们外出游玩的概率③该段时间 CPI 较高，偏高的物价水平导致人们用于娱乐消费的钱减少④合理推测该酒店那时刚开业不久。

根据确定的影响因素，我们接下来采用灰色关联分析，进一步确定其关联程度，对影响因素按照重要性排序。

5.2.2 对房间预定量影响因子的灰色关联分析

影响酒店房间预定量的因素多种多样，比如经济状况、季节变化、假期分布等等，而且这些因素对于房间预定量变化均表现出明显不确定性。针对不确定性系统，我国知名学者邓聚龙教授于 1982 年提出了“灰色系统理论”，通过对该系统“已知”部分的信息进行有效的提取并进行新的开发、生成，以此来寻找系统运行的内部规律，并进行科学地分析预测。因此，本文将酒店房间的预定量作为一个灰色系统来研究，采用灰色理论系统中的灰色关联度分析方法，来进行房间预定量影响因素的剖析。

灰色关联分析方法，具体的来讲，就是：在不完整的信息数据中，对我们所需要研究的各种因素，通过进行一定的数据分析和整理，在随机的因素序列间，寻找它们的关联性，进而找到重要的影响因素。计算方法与步骤如下：

1. 对原始数据进行无量纲化处理

由于原始数据中的量纲不同，比较起来不方便。因此首先对数据进行无量纲化处理。

设酒店房间月预定量为参考序列 $x_0(t)$ ，影响因素的每月分布情况为比较序列 $x_i(t)$ 。

$$x_0(t)=[38,326,2391,18003,17641,19962,17067,18064,19245,17201,17825,15887]$$

影响因素为经济水平（CPI）时：

$$x_i(t)=[100,101.2,99.6,100.2,100,100,100.2,100.2,100.1,100.1,100,100.4]$$

影响因素为季节（月平均气温时）：

$$x_i(t)=[7.08\ 7.77\ 11.69\ 17.02\ 21.45\ 24.82\ 27.42\ 28.53\ 24.85\ 20.35\ 14.63\ 8.48]$$

影响因素为法定节假日及双休（每月假期天数）时：

$$x_i(t)=[9\ 12\ 8\ 9\ 10\ 10\ 8\ 8\ 11\ 13\ 8\ 10]$$

2. 求差序列

$$x_0(t) \text{ 和 } x_i(t) \text{ 两序列在 } t \text{ 时刻的绝对差为: } \Delta_{oi}(t)=|x_0(t)-x_i(t)|, (t=1,2,3...12)$$

$$\text{二级最小差为: } \Delta_{\min} = \min_i \min_t |x_0(t) - x_i(t)|$$

二级最大差为： $\Delta \max = \max_i \max_t |xo(t) - xi(t)|$

3. 计算灰色关联度系数

在 t 时刻， $xi(t)$ 两序列的关联系数为：

$$\xi_{oi}(t) = \frac{\min_i \min_t |xo(t) - xi(t)| + \rho \max_i \max_t |xo(t) - xi(t)|}{|xo(t) - xi(t)| + \rho \max_i \max_t |xo(t) - xi(t)|}$$

其中 ρ 为分变系数，按照惯性 $\rho=0.5$

4. 求灰色关联度 r_{oi}

$$r_{oi} = \frac{1}{n} \sum_{i=1}^n \xi_{oi}(t)$$

r_{oi} 表示参考序列和比较序列之间的相互关联程度， r_{oi} 越大表示两序列之间的关联系数越高。

按照上述方法，我们依次求得了房间预定量与它的三个影响因素（季节、法定节假日与双休、经济状况）之间的关联度，结果如下表。

表格 3 房间预定量影响因素的原始数据及其灰色关联度

月份	季节	经济状况	法定节假日与双休
1	7.08	100	9
2	7.77	101.2	12
3	11.69	99.6	8
4	17.02	100.2	9
5	21.45	100	10
6	24.82	100	10
7	27.42	100.2	8
8	28.53	100.2	8
9	24.85	100.1	11
10	20.35	100.1	13
11	14.63	100	8
12	8.48	100.4	10
与预定量的关联度	0.502791421	0.5009264950	0.501066

通过上表我们可以看出，季节、经济状况和法定节假日及双休对于房间预订量的影响程度大体上相当。其中季节对于预定量的影响稍大，法定节假日影响次之，国民经济状况的影响最小。

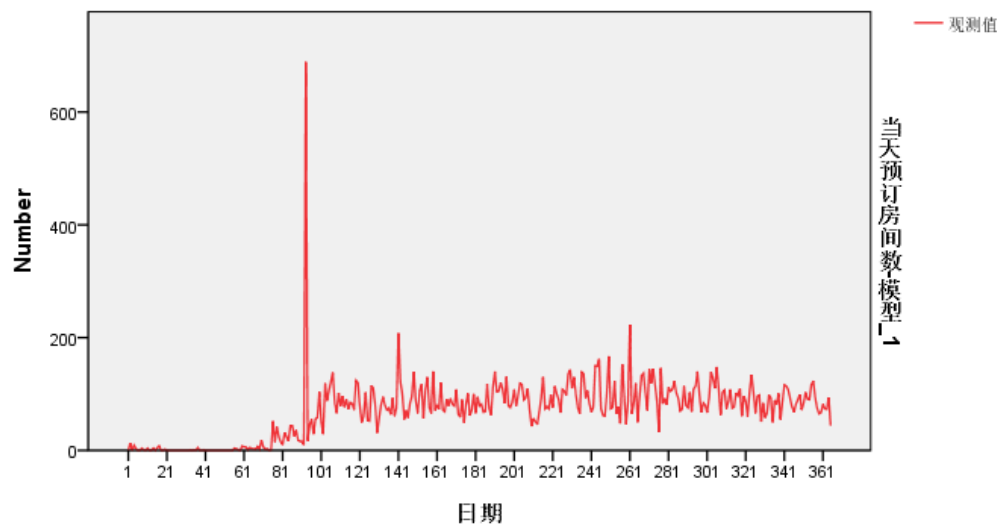
5.2.3 对 2016 年 1 月至 3 月每天预约房间数的预测

根据皇冠假日酒店 2015 年全年的预订信息，我们需要对 2016 年 1 月至 3 月每天的预定量做出预测。考虑到本题中房间预定量随时间进行变化，我们又仅有一年的数据，并受到季节、经济、节假日等不确定因素的影响，本文采用了时间

序列预测方法进行分析预测。

时间序列（time series）即指在一段时间内，通过对某一变量定期等间隔测量而获得的一组观察值的集合。因此，一个典型时间序列的数据，就是把定期获得的观察值按某种顺序或按列表方式排列后的数据集合。显然，2015 年每一天的房间预定量的数据集合满足这个定义。本文利用 SPSS 软件，建立数学模型来定量描述序列的特征，找到隐藏在序列中的基本变化规律，并预测序列未来的值。

表格 4 房间预定量（实际值）日分布图



从房间预定量日分布图中可以看出，2015 年前三个月以及后九个月的数据在均值水平上没有任何明显的趋势，而是一种随机的波动，且序列的周期性未知。在预测下一年前三个月的预定量时，不能通过普通线性回归方法预测环比数据，只能通过指数平滑法预测同比数据。基于指数平滑法的假设：序列具有“记忆”功能，即每一个值能“记住”它的先前值，并趋近于它，我们根据 2016 和 2015 年天气、节假日、居民收入关系，通过同期数据同比“正自相关性”，预测 16 年前三月数据。

拟合结果如下图所示，从图中可以看出，整体上拟合情况良好，拟合值的波动性要小于实际值。

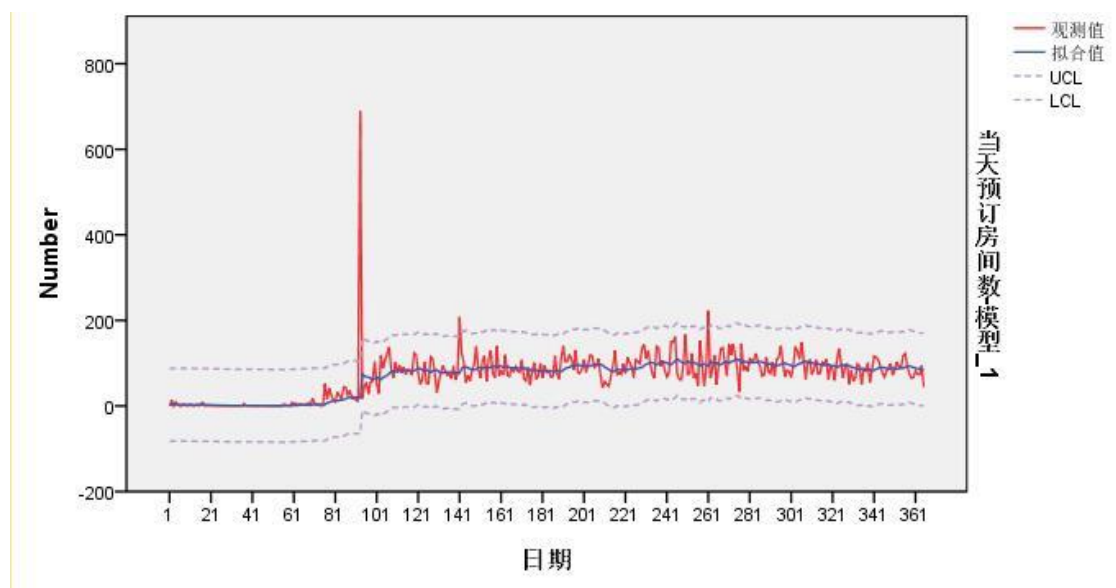


图 9 指数平滑法拟合 2015 年房间预定量

将 2016 年 1 月至 3 月的拟合值输出，就得到了预测值，如下表：

表格 52016 年 1 月至 3 月日酒店房间预定量预测表

日期	1 月 1 日	1 月 2 日	1 月 3 日	1 月 4 日	1 月 5 日	1 月 6 日	1 月 7 日
预定量 (间)	3	2	3	3	4	3	3
日期	1 月 8 日	1 月 9 日	1 月 10 日	1 月 11 日	1 月 12 日	1 月 13 日	1 月 14 日
预定量 (间)	3	3	3	3	3	3	2
日期	1 月 15 日	1 月 16 日	1 月 17 日	1 月 18 日	1 月 19 日	1 月 20 日	1 月 21 日
预定量 (间)	2	2	2	3	3	2	2
日期	1 月 22 日	1 月 23 日	1 月 24 日	1 月 25 日	1 月 26 日	1 月 27 日	1 月 28 日
预定量 (间)	2	2	2	2	2	1	1
日期	1 月 29 日	1 月 30 日	1 月 31 日	2 月 1 日	2 月 2 日	2 月 3 日	2 月 4 日
预定量 (间)	1	1	1	1	1	1	1
日期	2 月 5 日	2 月 6 日	2 月 7 日	2 月 8 日	2 月 9 日	2 月 10 日	2 月 11 日
预定量 (间)	1	1	1	1	1	1	1
日期	2 月 12 日	2 月 13 日	2 月 14 日	2 月 15 日	2 月 16 日	2 月 17 日	2 月 18 日
预定量 (间)	1	1	1	1	1	1	1
日期	2 月 19 日	2 月 20 日	2 月 21 日	2 月 22 日	2 月 23 日	2 月 24 日	2 月 25 日
预定量 (间)	0	0	0	0	0	0	0
日期	2 月 26 日	2 月 27 日	2 月 28 日	2 月 29 日	3 月 1 日	3 月 2 日	3 月 3 日
预定量 (间)	0	1	1	1	1	1	2
日期	3 月 4 日	3 月 5 日	3 月 6 日	3 月 7 日	3 月 8 日	3 月 9 日	3 月 10 日
预定量 (间)	2	2	2	2	2	2	3
日期	3 月 11 日	3 月 12 日	3 月 13 日	3 月 14 日	3 月 15 日	3 月 16 日	3 月 17 日
预定量 (间)	3	4	4	4	4	4	3
日期	3 月 18 日	3 月 19 日	3 月 20 日	3 月 21 日	3 月 22 日	3 月 23 日	3 月 24 日
预定量 (间)	7	8	11	12	12	12	14
日期	3 月 25 日	3 月 26 日	3 月 27 日	3 月 28 日	3 月 29 日	3 月 30 日	3 月 31 日

预定量（间）	15	15	17	19	20	21	21
--------	----	----	----	----	----	----	----

六. 模型分析与评价

6.1 问题（一）游园客流疏导方案制定

6.1.1 仿真模拟

以虚拟的游乐园为研究对象，以 Microsoft Visual C++6.0 软件平台构建模型及其运行环境。采用模拟的游客行为数据进行模型的校核，分析研究基于最小游览代价的决策模型的灵敏度。

根据题中所给的“YOUTH 游乐园规划图”、相关游乐项目的每场持续时间和每场容纳人数，设置模型参数 d_{ij} 、 T_{p_i} 和 b_i 。设游客样本容量为 50 人。由于篇幅限制，随机取其中 10 个人的数据。运行结果如下表（具体程序代码详见附件）：

表格 6 基于最小游览代价决策模型的仿真模拟（方案 1）

游客 编号	决策模型所提供的优化路线	途遇排队 等待人数	总等待 时间 (min)	平均等待 时间 (min)
1	B-E-C-D-F-G-H-I-J-A	2	7.5	8.1
2	B-C-D-F-G-H-I-J-A-E	0	5	
3	A-J-I-H-G-F-E-B-C-D	0	5	
4	A-J-I-H-G-F-E-B-C-D	4	17.5	
5	B-C-D-F-G-H-I-J-A-E	3	11.25	
6	A-J-I-H-F-E-B-C-D-G	1	8	
7	A-J-I-H-G-E-B-C-D-F	0	5.5	
8	B-C-D-F-G-H-I-J-A-E	0	11.25	
9	B-C-D-F-G-H-I-J-A-E	0	3	
10	A-J-I-H-G-F-E-B-C-D	0	7	

6.1.2 结果分析

为了更好地说明模型的有效性，我们对传统的游客分流模型也进行了模拟，称为方案 2。在方案 2 中，游客只通过查看游乐园的地图，选择距离自己最近的项目依次游玩，并游玩所有的游乐项目。我们将样本容量也设置为 50 人，选取与方案 1（基于最小游览代价决策模型）入园时间相同的十个人的相关模拟结果，数据如下：

表格 7 基于传统游客分流模型的仿真模拟（方案 2）

游客	决策模型所提供的优化路线	途遇排队	总等待	平均等待
----	--------------	------	-----	------

编号		等待人数	时间 (min)	时间 (min)
1	B-C-D-F-G-H-I-J-A-E	0	3.75	
2	B-C-D-F-G-H-I-J-A-E	0	10.5	
3	B-C-D-F-G-H-I-J-A-E	0	7.75	
4	B-C-D-F-G-H-I-J-A-E	3	7	
5	B-C-D-F-G-H-I-J-A-E	0	10.5	10.3
6	A-J-I-H-G-E-B-C-D-F	0	10	
7	A-J-I-H-G-E-B-C-D-F	2	21	
8	A-J-I-H-G-E-B-C-D-F	0	9	
9	A-J-I-H-G-E-B-C-D-F	14	13	
10	A-J-I-H-G-E-B-C-D-F	3	10.5	

1. 游玩路径选择的多样性分析

从表 3 我们可以看出，10 名游客中共有 5 种不同游览路径：**B-E-C-D-F-G-H-I-J-A**、**B-C-D-F-G-H-I-J-A-E**、**A-J-I-H-G-F-E-B-C-D**、**A-J-I-H-F-E-B-C-D-G**、**A-J-I-H-G-E-B-C-D-F**；从表 4 中我们看出，10 名游客中仅有 2 种不同游览路径：**B-C-D-F-G-H-I-J-A-E**和**A-J-I-H-G-E-B-C-D-F**。游玩路径越多，则游客越分散、每个项目的负荷压力越小，客流疏导效率越好。通过对比可见，基于最小游览代价的决策模型在游玩路径选择上的优越性。

2. 游客等待时间分析

比较表 3 和表 4 中游客的平均等待时间，我们不难发现按照方案 1 游览所有游乐项目的平均等待时间(8.1min)将比方案 2 的平均等待时间(10.3min)少。这反映了基于最小游览代价决策模型的动态分流方案可以较为有效地解决因为客流集中而造成的排队时间长问题。结合下图，我们对游客等待时间数据做进一步分析：从图中我们可以看出方案 2 中等待时间比方案 1 中等待时间长的点要略多（图中反映为红色的点在黑色点上面的个数略多），且方案 2 中数据点更加分散。通过计算我们得到：方案 1 中等待时间的标准差为 4.24min, 极差为 14.5min;方案 2 中等待时间的标准差为 4.52min, 极差为 17.25min。这说明方案 2 所得的等待时间较方案 1 所得波动更大。由此我们可以得到结论，基于最小游览代价决策模型的动态分流方案可以有效的降低排队等候时间，并使排队时间更加稳定、平均。

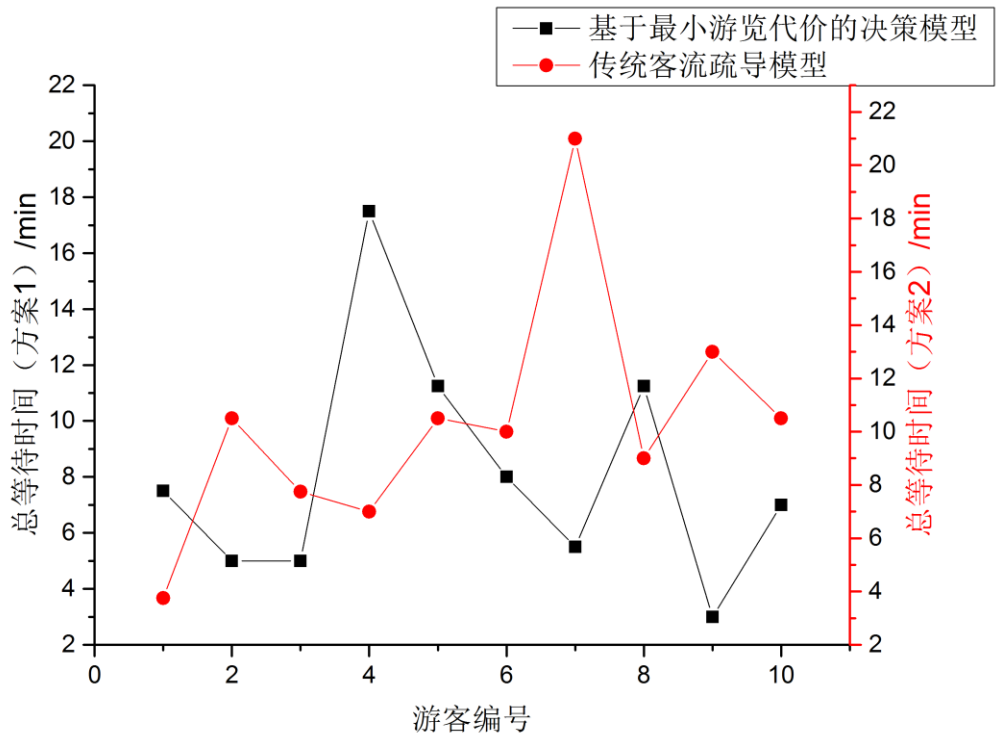


图 10 两种分流方案总等待时间对比图

3. 游客路遇排队人数分析

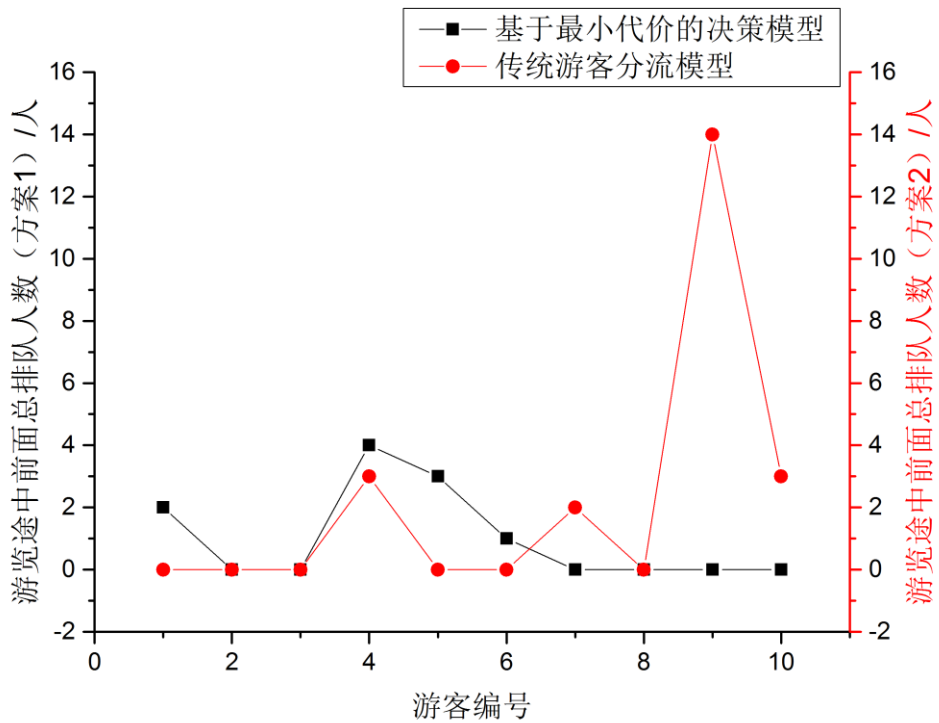
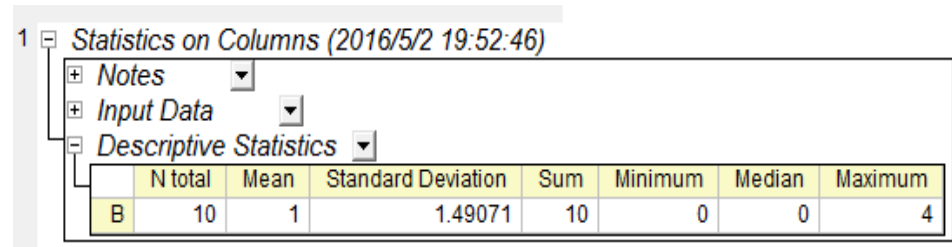


图 11 两种分流方案总排队人数对比图

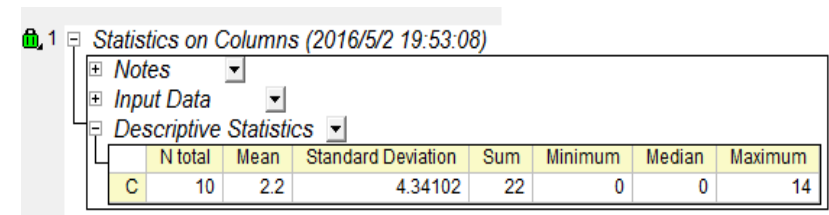
从上图我们可以清楚的看出：按照方案 1 进行游览，游客所遇到的总的

前面排队人数要少于按照方案 2 所遇到的情况，且方案 1 中每位游客遇到总排队人数分布更加平均，稳定。为了验证我们从图中得到的结论，我们对数据进行了分析，结果如下图。由于方案 1 中游客所遇到的总排队人数的均值和方差均远小于方案 2 中数值，这有力的说明了基于最小游览代价决策模型的动态分流方案能更加稳定地为每一位游客提供优化的游园体验，有效减少客流负荷集中的游园问题。



	N total	Mean	Standard Deviation	Sum	Minimum	Median	Maximum
B	10	1	1.49071	10	0	0	4

图 12 方案 1 总等待人数统计分析



	N total	Mean	Standard Deviation	Sum	Minimum	Median	Maximum
C	10	2.2	4.34102	22	0	0	14

图 13 方案 2 总等待人数的统计分析

6.1.3 模型评价

优点：

1. 从游客角度出发进行建模，优化游园路径，提高游客游园体验，思路新颖。
2. 模型综合考虑了排队时间、步行时间以及游玩时间等因素，与实际贴合。
3. 通过计算机仿真建模，对模型敏感性进行了分析检验，发现模型能有效的解决客流拥堵、游客负荷在项目间分布不均的问题。

缺点：

1. 计算机模拟时，样本容量偏小，入园时间是随机生成的，未考虑高峰时期的客流分布情况。

6.2 问题（二）酒店房间预定量的影响因素探究及未来预定量预测

6.2.1

运用 SPSS 软件，生成模型诊断报告，并借此对模型（二）进行分析。

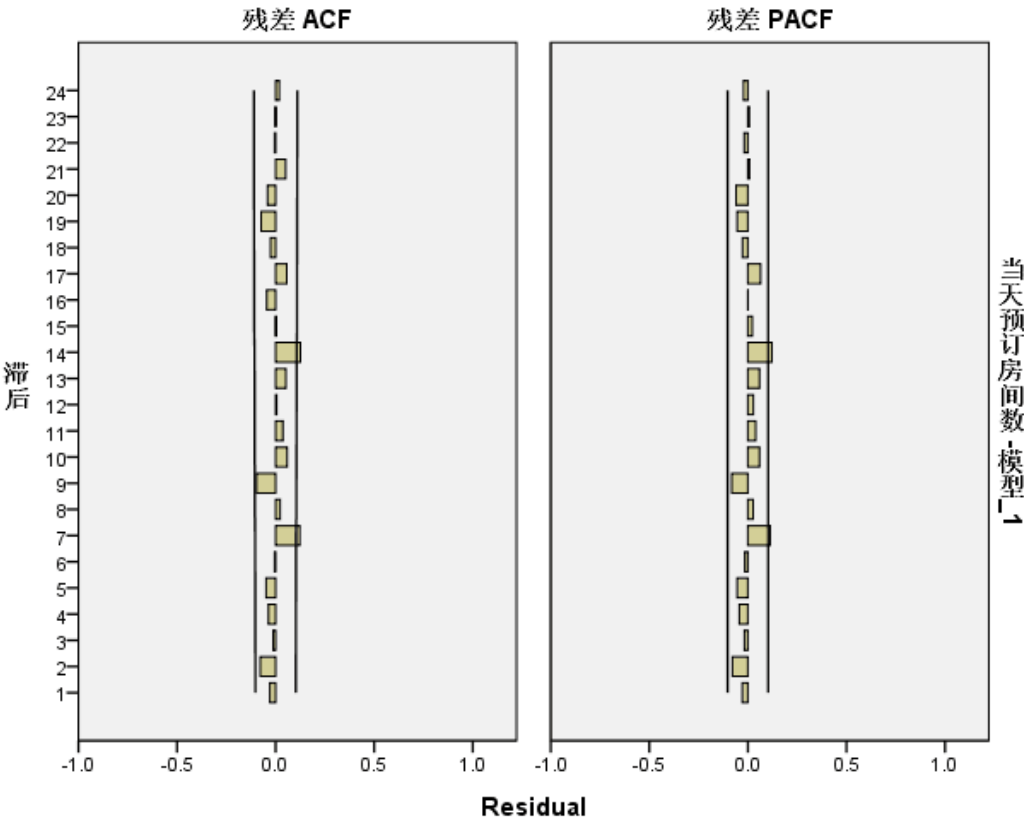
模型摘要

拟合统计量	模型拟合										
	均值	SE	最小值	最大值	百分位						
					5	10	25	50	75	90	95
平稳的 R 方	.469	.	.469	.469	.469	.469	.469	.469	.469	.469	.469
R 方	.388	.	.388	.388	.388	.388	.388	.388	.388	.388	.388
RMSE	43.151	.	43.151	43.151	43.151	43.151	43.151	43.151	43.151	43.151	43.151
MAPE	33.077	.	33.077	33.077	33.077	33.077	33.077	33.077	33.077	33.077	33.077
MaxAPE	341.783	.	341.783	341.783	341.783	341.783	341.783	341.783	341.783	341.783	341.783
MAE	19.424	.	19.424	19.424	19.424	19.424	19.424	19.424	19.424	19.424	19.424
MaxAE	669.684	.	669.684	669.684	669.684	669.684	669.684	669.684	669.684	669.684	669.684
正态化的 BIC	7.546	.	7.546	7.546	7.546	7.546	7.546	7.546	7.546	7.546	7.546

在“模型拟合”表中，拟合统计量“平稳的 R 方”的均值为 0.469，说明此模型拟合效果尚可，具有一定的参考性。

模型统计量													
模型	预测变量数	模型拟合统计量								Ljung-Box Q(18)			
		平稳的 R 方	R 方	RMSE	MAPE	MAE	MaxAPE	MaxAE	正态化的 BIC	统计量	DF	Sig	离群值数
当天预订房间数-模型_1	0	.469	.388	43.151	33.077	19.424	341.783	669.684	7.546	24.447	17	.108	0

在“模型统计量”表中，数据拟合的显著性值 Sig. 值为 0.108，大于 0.05 说明预定量的各影响因素之间无显著性差异，自变量方差均为齐性。这和模型 5.2.2 中结论相吻合。



残差图中，残差点在 0 线周围随机分布，没有任何可辨别的规律，说明此模型预测是合适的。

6.2.2 模型评价

优点：

1. 详细充分的对预定量的影响因素进行了分析。
2. 较为准确的预报了 2016 年 1 月至 3 月的预定量。

缺点：

1. 对于实际数据中一些突变的点未能准确拟合。

七. 模型优化

7.1 个性化游园路线制定

本文中基于最小游览代价的决策模型的算法中，是自动默认游客将游览所有的游乐设施。然而在实际生活中，一方面每个游客感兴趣的项目不尽相同，通常也未必对所有的项目都有兴趣；而另一方面，即使游客对所有的景点都向往，有时由于时间有限，也不大可能在一次游玩过程中玩遍所有的游乐项目。因此，如果能在现有算法的基础上，增加某种“偏好计算”机制（类似于网易云音乐中的“猜你喜欢”），模型所生成的推荐路线将更加个性化和智能。这样一来，凭借改进后的智能导览系统，当游客进入景区之后，系统会为其编号并收集个人偏好，从而生成个性化的动态调度方案。这样的调度方案将更加贴合实际，因此是我们考虑优化的方向之一。

7.2 高峰时期园内客流量远超预期时的限流方案

在旅游旺季，游乐园会迎来客流高峰，在园内人流量大大超过预期值的情况下，即使现有模型发挥最大作用，也不能保证游乐园内各景点的负荷处于适合的限度之下。人流量过大的景点可能会造成拥挤，甚至引发安全隐患，这是游乐园需要避免的。因此，模型下一步应当考虑园内人流量过大的情况下，采用怎样的限流机制，控制游乐园各景点人流量，使游乐园既能够在正常负荷状态下稳定运行，同时又兼顾游客体验。

八. 参考文献

- [1] 姜启源，谢金星，叶俊 编 数学模型（第四版） 北京：高等教育出版，2011
- [2] 周霓 著 基于数据分析的旅游市场预测及经济效应优化研究 北京：经济科学出版社 2014
- [3] 郑天翔 基于单步协调控制的大型游乐场游客智能导览系统的建模与仿真暨南学报（哲学社会科学版）第 37 卷第 10 期：P138-P145 2015 年 6 月
- [4] 郑天翔 基于动态实时调度的主题公园游客时空分流导航管理研究 旅游科学 第 26 卷第 4 期 P8-P15 2012 年 6 月

[5] 天气后报

<http://www.tianqihoubao.com/lishi/shanghai/month/201512.html> 2016/5/3

[6] 上海市统计局

<http://www.stats-sh.gov.cn/data/toMData.xhtml?u=/monthdata/2015.html>2016/5/3

[7] 陈平雁 编 SPSS10.0 统计软件高级应用教程（第一版）人民军医出版社
2004

附件

【1】基于最小游览代价的决策模型算法

```
#include "iostream.h"
#include "stdio.h"
#include "stdlib.h"
#include "time.h"
#include "fstream"
#include "math.h"
using namespace std;

struct event
{
    double occurtime;           //事件发生时间
    int currentsite;            //游客在该事件发生时所处景点位置
    int property;               //事件类型，1 为到达，2 为待游玩，3 为完成游玩
    int visitedsite[10];        //该事件发生时每位游客已玩过的景点
    double availabletime[10];    //该事件发生时各景点下一次空闲时间
    int *queue;                 //该事件发生时各景点前排队人数
    int goalsite;                //该事件为待游玩时目标景点
    int status;                 //游客在该事件发生时所处状态，1 为行走中，2 为使用娱乐设施中，3 为排队等待中
};

struct person
{
    int visitedsite[10];        //该事件发生时该游客已玩过的景点
    int Time;                   //游客入园时间
    double Twait;                //游客总共等待时间
    struct event Event[100];     //每位游客产生的各类事件
}Leute[100];

double velocity=6.0;           //步行速度，单位为 m/min
int Ort[10]={1,2,3,4,5,6,7,8,9,10}; //代表 10 个景点
int
distance1[10][10]={ {5000,300,600,1500,350,1450,900,1050,600,250}, {300,5000,300,
750,350,1250,900,1250,800,550}, {600,300,5000,450,500,950,1050,1400,950,850}, {15
00,750,450,5000,950,500,1150,1550,1400,1300}, {350,350,500,950,5000,1200,550,900
,450,600},
    {1450,1250,950,500,1200,5000,
650,1050,1500,1800},
    {900,900,1050,1150,550,650,
5000,400,850,1200},
```



```

        {1050 , 1250 , 1400 , 1550 ,
900      ,1050      ,400      , 5000      , 450      ,800},
        { 600 , 800 , 950      , 1400      , 450      , 1500      ,
850      , 450      , 5000      , 350},
        { 250 , 550 , 850      , 1300, 600      , 1800 ,
1200      , 800 , 350      , 5000}};
    // 景点到景点间最短距离, 单位为 m
    double PlayT[10]={33, 1. 25, 2. 5, 2. 5, 5, 2. 5, 2, 1. 5, 1. 5, 2}; //各景点设施每场运行时间, 单位为 Min
    double belast[10]={4, 0. 3, 0. 5, 0. 3, 1, 0. 5, 0. 3, 0. 3, 0. 2, 0. 5};
    //景点承载力

int man=1;

int *queuenum(int N,int queue[],int m)
{
    double status[20000],occurtime[2000],currentsite[2000];

    ifstream outputFile1("C:\\status.txt");
    int r=0;
    while(outputFile1>>status[r])
    {
        ++r;
    }
    outputFile1.close();

    ifstream outputFile2("C:\\occurtime.txt");
    int u=0;
    while(outputFile2>>occurtime[u])
    {
        ++u;
    }
    outputFile2.close();

    ifstream outputFile3("C:\\currentsite.txt");
    int e=0;
    while(outputFile3>>currentsite[e])
    {
        ++e;
    }
    outputFile3.close();

```

```

for(int mq=0;mq<m;mq++)
{
    queue[mq]=0;
}

for(int j=0;j<N;j++)
{

    for(int k=0;k<m;k++)
    {

        if(status[N]==3&&occurtime[j]<occurtime[N]+2&&occurtime[j]>occurtime[N]-
2&&currentsite[N]==0rt[k])
            {
                queue[k]=queue[k]+1;
            }

    }

}

return queue;
}

int Optimalsite(int currentsite,int visitedsite[10],double Tf[10],person
Leute,int queue[10]) //基于最小代价策略的实时调度函数
{
    double timecost[100];int i=0,m=0,p=0,n=0,num=0,I,J,min=0;
    int unvisited[10],optimal=0;double T1,T2;
    for(int j=0;j<10;j++)
    {

        for(i=0;i<10;i++)
        {

            if(0rt[j]==visitedsite[i])
                break;
            if(i==9)
            {
                unvisited[m]=0rt[j];m=m+1;
            }
        }

    }
}

```

```

    }
    for (p=0;p<m;p++)
    {
                                                                    //
    cout<<unvisited[p]<<"haha";
        if (Tf[unvisited[p]-1]==0)
        {
            timecost[p]=distance1[currentsite-1][unvisited[p]-1]/velocity;
        }
        else
        {
            T1=Tf[unvisited[p]-1]-Leute.Time+PlayT[unvisited[p]-
1]*queue[unvisited[p]-1]/belast[unvisited[p]-1];
            T2=distance1[currentsite-1][unvisited[p]-1]/velocity;
            if (T1>T2)
            {
                timecost[p]=T1;
            }
            else
                timecost[p]=T2;
        }
    }
    for (I=0;I<p;I++)
    {
        for (J=0;J<p;J++)
        {
            if (timecost[J]<timecost[I])
                break;
            if (J==p-1)
                optimal=unvisited[I];
        }
    }
    return optimal;
}

```

```

void function(struct person &Leute, struct event &Event, int N, int status)
{

```

```

int chosensite,h=0,a[10];double Tfinish;
if(Event.property==1)
{
    ofstream          inputFile1("C:\\status.txt",ios::out          |
ios::app);inputFile1<<status<<' ' ;inputFile1.close();
    Event.currentsite=N;                                          ofstream
inputFile2("C:\\currentsite.txt",ios::out          |
ios::app);inputFile2<<Event.currentsite<<' ' ;inputFile2.close();

//cout<<"didian"<<Event.currentsite<<endl;
    Leute.visitedsite[Event.currentsite-1]=Event.currentsite;
                                                // for(int i=0;i<10;i++)

//{cout<<Leute.visitedsite[i]<<"haha"<<endl; } //该事件发生时所处景点位置随机生成

    Event.queue=queuenum(man-1,a,10);

chosensite=Optimalsite(Event.currentsite,Leute.visitedsite,Event.availabletime,
Leute,Event.queue);    //计算当前最快可游玩的景点
    Leute.Event[man].property=2;
    Leute.Event[man].goalsite=chosensite;                                //
cout<<"jisuanchudemubiaodian"<<Leute.Event[man].goalsite<<endl;
    if(chosensite==0)
    {
        ofstream          inputFile3("C:\\currentsite.txt",ios::out          |
ios::app);inputFile3<<Event.currentsite<<' ' ;inputFile3.close();
    }
    else
    {
        ofstream          inputFile4("C:\\currentsite.txt",ios::out          |
ios::app);inputFile4<<Leute.Event[man].goalsite<<' ' ;inputFile4.close();
    }
    Leute.Event[man].occurtime=Event.occurtime+distance1[Event.currentsite-1][chosensite-1]/velocity;
    ofstream          inputFile5("C:\\occurtime.txt",ios::out          |

```

```

ios::app);inputFile5<<Leute.Event[man].occurtime<<' ';inputFile5.close();
    Leute.Event[man].status=1;
    ofstream          inputFile6("C:\\status.txt", ios::out
ios::app);inputFile6<<Leute.Event[man].status<<' ';inputFile6.close();
    function(Leute, Leute.Event[man], N, Leute.Event[man].status);

    return;

}
if(Event.property==2)
{
    if(abs(Event.availabletime[Event.goalsite-1]-Event.occurtime)-
PlayT[Event.goalsite]*int(abs(Event.availabletime[Event.goalsite-1]-
Event.occurtime)/PlayT[Event.goalsite])==0)
    {
        Leute.Twait=0;Event.status=2;
        ofstream          inputFile7("C:\\status.txt", ios::out
ios::app);inputFile7<<Event.status<<' ';inputFile7.close();
    }
    else
    {
        Event.status=3;
        ofstream          inputFile8("C:\\status.txt", ios::out
ios::app);inputFile8<<Event.status<<' ';inputFile8.close();
        Event.queue=queuenum(man-1, a, 10);

        cout<<"duilierenshu";
        for(int g=0;g<10;g++)
        {
            cout<<Event.queue[g]<<' ';
        }

        //cout<<Event.queue[Event.goalsite]<<endl;
        Leute.Twait=abs(Event.availabletime[Event.goalsite-1]-
Event.occurtime)-
PlayT[Event.goalsite]*int(abs(Event.availabletime[Event.goalsite-1]-
Event.occurtime)/PlayT[Event.goalsite])+PlayT[Event.goalsite-
1]*Event.queue[Event.goalsite-1]/belast[Event.goalsite-1];
        cout<<"Waiting Time="<<Leute.Twait<<endl;
    }
    Tfinish=PlayT[Event.goalsite-1]+Leute.Twait;
    man++;

```

```

        Leute.Event[man].property=3;
        Leute.Event[man].occurtime=Event.occurtime+Tfinish;
        ofstream          inputFile9("C:\\occurtime.txt", ios::out
ios::app);inputFile9<<Leute.Event[man].occurtime<<' ';inputFile9.close();
        Leute.Event[man].currentsite=Event.goalsite;
        ofstream          inputFile10("C:\\currentsite.txt", ios::out
ios::app);inputFile10<<Event.currentsite<<' '; inputFile10.close();
        Leute.Event[man].status=2;
        ofstream          inputFile11("C:\\status.txt", ios::out
ios::app);inputFile11<<Leute.Event[man].status<<' ';inputFile11.close();
        function(Leute, Leute.Event[man], N, Leute.Event[man].status);return;

    }
    if (Event.property==3)
    {

        Event.queue=queuenum(man-1, a, 10);
        status=2;
        ofstream          inputFile12("C:\\status.txt", ios::out
ios::app);inputFile12<<status<<' ';inputFile12.close();
        Leute.visitedsite[Event.currentsite-1]=Event.currentsite;
        ofstream          inputFile13("C:\\currentsite.txt", ios::out
ios::app);inputFile13<<Event.currentsite<<' '; inputFile13.close();

                                cout<<"wanchengyouwanhou";
                                for(int i=0;i<10;i++)
                                {cout<<Leute.visitedsite[i];
                                    }                                cout<<"

"<<Event.occurtime<<endl;

        if (Event.queue[Event.currentsite-1]==0)
        {
            Event.avabletime[Event.currentsite-1]=Event.occurtime;

        }
        else
        {
            Event.avabletime[Event.currentsite-
1]=Event.occurtime+PlayT[Event.currentsite-1];
        }
        man++;
    }
}

```

```

        Leute.Event[man].property=2;
        Event.queue=queuenum(man-1, a, 10);

        Leute.Event[man].goalsite=Optimalsite(Event.currentsite, Leute.visitedsite, Event
        .availabletime, Leute, Event.queue); //cout<<" 下一个理想景点
        "<<Leute.Event[man].goalsite<<' ';
            if(Leute.Event[man].goalsite==0)
            {
                ofstream      inputFile14("C:\\currentsite.txt", ios::out
        ios::app); inputFile14<<Event.currentsite<<' '; inputFile14.close();
            }
        else
        {
                ofstream      inputFile15("C:\\currentsite.txt", ios::out
        ios::app); inputFile15<<Leute.Event[man].goalsite<<' '; inputFile15.close();
        }
        // cout<<"*****"<<Leute.Event[man].goalsite<<"*****";
            if(Leute.Event[man].goalsite==0)
            {

                return;

            }
            else
            {

                Leute.Event[man].occurtime=Event.occurtime+distance1[Event.currentsite-
        1][Leute.Event[man].goalsite-1]/velocity;
                ofstream      inputFile16("C:\\occurtime.txt", ios::out
        ios::app); inputFile16<<Leute.Event[man].occurtime<<' '; inputFile16.close();

                Leute.Event[man].status=2; //cout<<"occurtime="<<Event.occurtime<<endl;
                ofstream      inputFile17("C:\\status.txt", ios::out
        ios::app); inputFile17<<Leute.Event[man].status<<' '; inputFile17.close();
                function(Leute, Leute.Event[man], N, Leute.Event[man].status);
            }
        }
        // for(int q=0;q<100;q++)
        // {
        //     cout<<Leute.Event[q].occurtime<<' ';
        // }
        // cout<<endl;
    }
}

```

```

void main()
{

    int a[250], tag(0), m, j, z, N=100, num(0);


    ofstream fileout1("C:\\occurtime.txt", ios::trunc);
    fileout1.close();
    ofstream fileout2("C:\\status.txt", ios::trunc);
    fileout2.close();
    ofstream fileout3("C:\\currentsite.txt", ios::trunc);
    fileout3.close();


    srand(time(NULL));
    z=int(rand()%101);
    a[0]=z;
    for(m=0;m<N;m++)
    {

        z=int(rand()%101);
        for(j=0;j<=num;j++)
        {
            if(z==a[j])
            {tag=1;}
        }
        if(tag==0)
        {a[m+1]=z;num++;}
        if(tag==1)
        {m=m-1;}
        tag=0;
    }
}

```



```

int a1[250], tag1(0), m1, j1, z1, N1=100, num1(0);

srand(time(NULL));
z1=int(rand()%2+1);
a1[0]=z1;
for(m1=0;m1<N1;m1++)
{

    z1=int(rand()%2+1);
    for(j1=0;j1<=num1;j1++)
    {
        if(z1==a1[j])
            {tag1=1;}
    }
    if(tag1==0)
        {a1[m1+1]=z1;num1++;}
    if(tag1==1)
        {m1=m1-1;}
    tag1=0;
}

int a2[250], tag2(0), m2, j2, z2, N2=10, num2(0);

srand(time(NULL));
z2=int(rand()%3-1);
a2[0]=z2;
for(m2=0;m2<N2;m2++)
{

    z2=int(rand()%3-1);
    for(j2=0;j2<=num2;j2++)
    {
        if(z2==a2[j])
            {tag2=1;}
    }
    if(tag2==0)
        {a2[m2+1]=z2;num2++;}
    if(tag2==1)
        {m2=m2-1;}
    tag2=0;
}

```

```

for(int i=0;i<100;i++)
{
    cout<<"hahah";
    Leute[i].Time=a[i];    //随机生成游客入园时间
    cout<<Leute[i].Time<<endl;
    Leute[i].Event[0].property=1;    //第一个事件生成为到达事件
    Leute[i].Event[0].occurtime=Leute[i].Time;    //该事件发生时间为随机生成

    for(int j=0;j<10;j++)
    {
        cout<<a2[j]<<' ';

Leute[i].Event[0].availabletime[j]=Leute[i].Event[0].occurtime+PlayT[j]*a2[i];
        Leute[i].visitedsite[j]=0;
    }
    Leute[i].Event[0].status=1;
    function(Leute[i],Leute[i].Event[0],a1[i],Leute[i].Event[0].status);
//对生成的第一个到达事件进行循环函数操作
    cout<<"heiheiheih";
}

}

```

【2】灰色关联模型算法

```

#include"iostream.h"
#include"math.h"
#include"fstream"
#include"iomanip.h"
#include "stdio.h"
#include "stdlib.h"
using namespace std;
double min(double a[],int N)
{
    int i=0;double m=0;
    for(i=0;i<N;i++)
    {

```

```

for(int j=0;j<N;j++)
{
if(a[i]>a[j])
{
break;
}
if(j==9)
m=a[i];
}
}
return m;
}

```

```

double max(double a[],int N)
{
int i=0;double m=0;
for(i=0;i<N;i++)
{
for(int j=0;j<N;j++)
{
if(a[i]<a[j])
{
break;
}
if(j==9)
m=a[i];
}
}
return m;
}

```

```

void shuchu(double a[],int N)
{
for(int i=0;i<N;i++)
{
cout<<setprecision(9)<<a[i]<<' ';
}
cout<<endl;
}

```

```

double *initiate(double a[],int N,double b[])
{

```

```

int i=0;
for(i=0;i<N;i++)
{
b[i]=a[i]/a[0];
}
return b;
}
void main()
{
double x1[12]={38, 326, 2391, 18003, 17641, 19962, 17067, 18064, 192
45, 17201, 17825, 15887},
y1[12]={100, 101.2, 99.6, 100.2, 100, 100, 100.2, 100.2, 100.1, 100.1, 100, 1
00.4};
double *Ix1,*Iy1, guodu[12], m, n, e=0, yijicha[12], erjicha, yijida[12]
, erjida, jueduicha[12], guanlianxishu[12], guanliandu;
Ix1=new double[12];Iy1=new double[12];
Ix1=initiate(x1, 12, Ix1);Iy1=initiate(y1, 12, Iy1);
cout<<"数据 1:";shuchu(Ix1, 12);
cout<<"数据 2:";shuchu(Iy1, 12);

for(int i=0;i<12;i++)
{
for(int j=0;j<12;j++)
{
m=Iy1[j]-Ix1[i];
if(m>=0)
{guodu[j]=m;}
else{guodu[j]=-m;}
}
yijicha[i]=min(guodu, 12);yijida[i]=max(guodu, 12);
}
cout<<"输出一级最小差:";shuchu(yijicha, 12);
cout<<"输出一级最大差:";shuchu(yijida, 12);
erjicha=min(yijicha, 12);
erjida=max(yijida, 12);
cout<<"二级最小差="<<erjicha<<endl;
cout<<"二级最大差="<<erjida<<endl;

for(int h=0;h<12;h++)
{
n=Iy1[h]-Ix1[h];
if(n>=0)

```

```

    {jueduicha[h]=n;}
    else {jueduicha[h]=-n;}
}
cout<<"绝对差:";shuchu(jueduicha, 12);

for(int k=0;k<12;k++)
{
    guanlianxishu[k]=(erjicha+0.5*erjida)/(jueduicha[k]+0.5*erjida);
    e=e+guanlianxishu[k];
}
cout<<"关联系数:";shuchu(guanlianxishu, 12);
guanliandu=e/12;
cout<<"关联度为: "<<guanliandu<<endl;

}

```