项目编号：T120PRP43002

上海交通大學

本科生研究计划（PRP）研究论文

(第 43 期)

论文题目：基于最优化方法的航空旅客购票行为分析

项目负责人：曹宇峰 学院（系）：安泰经济与管理学院

指导教师：曹宇峰 学院（系）：安泰经济与管理学院

参与学生：陈琪颖、李云艾、钱苒熙、吴家仪、陈敏慧

项目执行时间：2023 年 2 月至 2023 年 9 月

/

摘 要

航空旅客的购票行为是复杂而受多因素影响的决策。本研究通过搜集的真实数据，对其中影响购票的各因素（自变量）进行了特征工程，并运用离散优化模型中的多项 logit 模型，综合线搜索与置信域的优化方法实现效用最大化，并评估各因素对购票行为的影响大小。本研究的重点在于将最优化的各类方法投入应用-优化本场景的最大似然函数，并对不同方法的收敛性进行比较，并尝试用数值分析的方法以及其他数学的工具给出诠释。

关键词： 顾客决策分析，离散选择模型，线搜索方法，置信域方法

ABSTRACT

Air travelers' ticket purchasing behavior is a complex and multifactor influenced decision. In this study, we have feature engineered the factors (independent variables) affecting ticket purchasing among them through the real data set collected, and used the multinomial logit model from the discrete optimization model to maximize the utility by integrating the optimization methods of line search methods and trust region methods, and evaluated the magnitude of the influence of each factor on the ticket purchasing behavior. We focus this study on putting the various methods of optimization into application-optimizing the maximum likelihood function for this air traveling scenario and comparing the convergence of the different methods and attempting to give interpretations using numerical analysis and other mathematics tool.

Keywords : Consumer behavior; Discrete choice model ; Line search methods; Trust region methods

1. 绪论

航空旅客的购票行为是一个复杂的决策过程，受到多种因素的影响。 在当今竞争激烈的航空市场中，航空公司需要寻找方法来优化旅客购票行为，以提高收入和客户满意度，因此需要通过了解和理解旅客的购票行为，来制定更有效的策略。 购票行为的复杂性体现在旅客在购票过程中需要考虑诸多因素，如航班时间、价格、舱位选择、退改签政策等。 因此，为了更好地理解旅客的决策，我们需要运用合适的方法和模型来分析这些因素及其相互关系。 本研究旨在通过搜集真实数据，使用特征工程建立离散优化模型（多项 logit 模型），并运用最优化方法来分析航空旅客购票行为，评估各因素对购票行为的影响。

我们参考了相关文献以获得建模航空旅客购票行为的思路。 在 Choice Based Revenue Management for Parallel Flights（Dai, Ding, Kleywegt, Wang, & Zhang）中考虑了基于选择的随机组合优化问题，旨在最大化航空公司的预期收入。 研究中使用了离散选择模型，以纳入不同出发时间的客户偏好。 研究结果表明，客户的选择行为存在不连续性，即使价格差异很小，最便宜的选择也比次便宜的选择受到更多的需求。 而 Network Revenue Management under a Spiked Multinomial Logit Choice Model"（Cao, Kleywegt, & Wang）这篇文献研究了一个基于 spiked-MNL 选择模型的网络收入管理问题， 其通过线性规划来高效地求解随机动态规划的确定性近似问题，构建预订限制策略。 Nocedal 和 Wright 的 Numerical Optimization 是经典的关于数值优化的教材，其中介绍的线搜索和置信域方法，以及在约束条件下的拉格朗日乘子法为本研究中的优化方法提供了理论支持和实践指导

本研究主要分为对购票行为数据集的特征工程，对购票行为决策建模，对模型中函数的优化以及对建模结果的评估几部分。 相对于“航空旅客购票行为分析”研究小组中的其他研究，本文将重点偏向于尝试将多种最优化方法应用于航空旅客购票行为的分析和优化。 其中包含对不同方法在本问题中的适用性与数值结果（收敛性）的对比，以及通过数学视角用数值分析和其他数学工具对结果进行解释的尝试。

2. 关于航旅信息数据集的特征工程

2.1 数据来源

本研究数据集来源于指导教师 Yufeng Cao 在其关于 Network Revenue Management under a Spiked Multinomial Logit Choice Model 的研究工作中根据实际航空旅客购票历史数据整理收集的数据。其具体市场中周一航班一年的机票预订数据，数据提供了每个客户的选择集和最终选择，并包括相关信息如预订渠道、出发天数、预订时间等。

2.2 数据分析，数据处理以及特征选择

2.2.1 试点性定性分析

由于原数据集数据量庞大且信息非常详尽具体，为保证本研究的可行性，我们先通过定性的分析进行了一定的数据降维。通过观察以及调研国内主流航空购票软件的航旅信息呈现，以及关于航空常旅客的试点性采访,我们对影响旅客购票决策的因素进行了初步探讨由此在接下来覆盖更广泛样本量的数据集中,采用量化研究方法对这些影响因素进行精确测量,并利用本实验中的模型对其影响程度进行定量分析，为后续的大样本量研究提供了借鉴这些关键影响因素的参考,并便于据此设计适当的量化指标与变量。从而使后续研究能够更有针对性和效率地对问题给出定量结论。

最终，基于此方法，本研究初步保留了原数据集中以下自变量：

Tkt_Price：机票价格

is_nonchangeable；Change_Fee：是否不可改签；改签费用

is_refundable；是否可退款；

Cancel_Fee：取消费用；

is_cheapest；是否为个人选择集中最便宜的机票；

book_channel_idx：购票渠道；

book_DtD：订票距离行程时间；

dept_hour_idx：起飞时间段。

2.2.2 进一步数据处理及最终特征选取

首先，我们将 book_channel_idx 和 book_DtD 两个多分类变量进行了哑变量处理记为（book_channel_idx_1，book_channel_idx_2，book_channel_idx_3）；（dept_hour_idx_0，dept_hour_idx_1）。

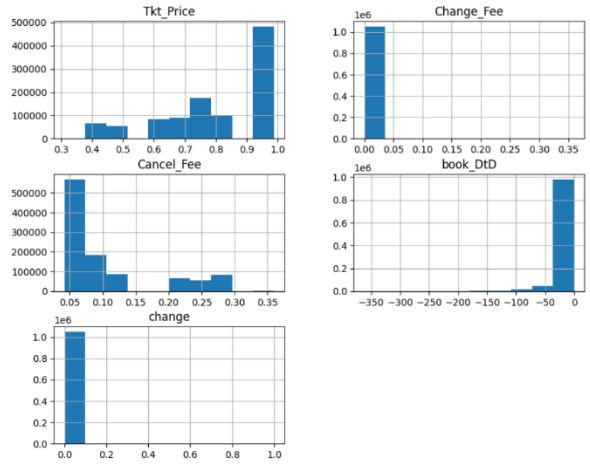
进一步考虑到其哑变量取值性质，以及购票渠道和购票价格在实际情况中的潜在关联我们将（book_channel_idx_1，book_channel_idx_2，book_channel_idx_3）与 Tkt_Price 相乘，将其合并为一个变量，仍然记为（book_channel_idx_1，book_channel_idx_2，book_channel_idx_3）。

同时考虑到 is_nonchangeable；Change_Fee 均是关于改签政策的变量，我们作如下处理，将其合并为一个变量 change：若 is_nonchangeable=0，及可以改签，则令变量 change=Change_Fee（数据集中已经归一化处理）；若 is_nonchangeable=0，及可以改签，则令变量 change=Change_Fee，若 is_nonchangeable=1，即不能改签，则令 change=1。易得 change 为一个分布在（0，1）之间的变量，且其值越小，越经济上利好潜在的航空旅客。

故我们最终选取的除顾客本人外的自变量为【change，（book_channel_idx_1，book_channel_idx_2，book_channel_idx_3），（dept_hour_idx_0，dept_hour_idx_1，is_refundable，Cancel_Fee，is_cheapest，book_DtD】

2.2.3 对特征的描述性分析以及定量评价

我们首先绘制了非离散型(这里指非 index，非 0-1 型变量)变量（包含未经处理的变量）的大致的频率分布直方图如下，观察其大致范围：



其后，对各变量基本统计量进行了计算：

Variable	mean	std	min	25%	50%	75%	max
Tkt_Price	0.808264	0.190121	0.31	0.687	0.796	0.989	0.989
Change_Fee	0.0055	0.017307	0	0	0	0	0.36
Cancel_Fee	0.103501	0.080051	0.042	0.049	0.072	0.137	0.36
is_cheapest	0.378295	0.484962	0	0	0	1	1
is_refundable	0.4579	0.498225	0	0	0	1	1
is_nonchangeable	0.001564	0.039517	0	0	0	0	1
book_DtD	-11.4493	20.89767	-3.62	-12	-5	-2	0
book_channel_idx_1	0.378295	0.484962	0	0	0	1	1
book_channel_idx_2	0.424157	0.428706	0	0	0.419	0.796	0.989
book_channel_idx_3	0.2874	0.400275	0	0	0	0.721	0.989
dept_hour_idx_0	0.146973	0.354079	0	0	0	0	1
dept_hour_idx_1	0.853027	0.354079	0	1	1	1	1
change	0.006502	0.040608	0	0	0	0	1

同时，为了评价最终特征选取的合理性，我们计算了其与因变量及顾客最终决策变量（Booked_Itin）的相关性的显著性：

Column	Spearman p-value	Pearson p-value
Tkt_Price	3.02E-128	5.22E-55
Change_Fee	3.10E-32	5.06E-111
Cancel_Fee	1.65E-303	8.34E-09
is_nonchangeable	1.37E-104	1.37E-104
change	3.09E-32	2.20E-118

可见两种相关性计算方法对应的 p-value 都趋近于零，说明其相关性是非常显著的.

3.基于多项 Logit 模型（MNL）的购票行为决策建模

3.1MNL（多项 Logit 模型）模型与效用最大化准则

MNL 是 DCM（Discrete Choice Model（离散选择模型））中的一类，其因变量不是连续的变量，在实际应用中往往包含决策者（Decision Makers），即做出选择行为的主体及其自身的属性，备选方案集（Alternatives）及各个方案的属性（Attributes of Alternatives）等相关的因素作为自变量。

不同的方案属性描述了各方案提供给人们的不同维度上的效用（Utility，其常用于度量与实际中直觉相符的决策者偏好，如更低的费用等）

效用最大化准则是实际应用中最为常见的决策准则，而为了量化这样的效用，可以使用 MNL 模型以及使用到似然性（likelihood）这个概念。似然性与概率不同其是从观测结果（最终决策）出发，计算分布函数的参数的可能大小（权重）。以下，我们具体设置函数，即最大似然函数，来描述似然性。

3.2 最大似然函数的设置

对于相同的个体其面对一个选择集我们的目的是计算在给定参数向量下观测数据出现的概率，此即似然函数将每个个体决策的对数概率累加，得到对数似然函数值。在本研究中，我们使用以下最大似然函数来分析航空旅客的决策行为。

$$\log L(\theta) = \sum_{i=1}^{N} y_i \log\big(P(y_i = 1|x_i, \theta)\big) + (1 - y_i) \log\big(P(y_i = 0|x_i, \theta)\big)$$

其中：(N) 为决策者总数；(y_i) 是二进制变量，表示个体 (i) 的行程是否预订；(x_i) 是个体 (i) 的输入变量向量（个人数据集）；(\theta) 是参数向量[change，book_channel_idx_1，book_channel_idx_2，book_channel_idx_3，dept_hour_idx_0，dept_hour_idx_1，is_refundable，Cancel_Fee，is_cheapest，book_DtD]（其在 pytorch 中的具体计算和实现见附录）

最优化最大似然函数的目的是找到使观测数据概率最大化的参数值（权重）。在分析航空旅客的决策行为时，我们希望通过调整参数向量权重的值来最大化观测数据中个体决策的概率。通过这种方式，我们可以推断出航空旅客的决策行为规律，了解他们在不同情境下的决策偏好。

为了评估各因素对航空旅客行为的影响并进一步用于分析和预测，我们应该找到该最大似然函数的极值及此时对应的各指标权重。在最优化中，对于无约束的优化策略主要分为两类：线搜索方法及置信域方法。这是两类完全不同的算法，本研究分别进行了尝试并观察和评估了其优化结果。

3.3 使用拉格朗日乘子法转化最大似然函数优化中的约束条件

注意到在我们最终选取的特征（自变量）中，book_channel_index 以及 dept_hour_index 是基于哑变量生成的向量，但是为了方便优化我们将其分割为不同的单变量，故自然的，book_channel_index[i] (i=1,2,3)三个变量应该有相近的权重（及对应的 theta 相近），同理 dept_hour_index[i] (i=1,2)也应该有相近权重。为了达成这一目的，我们可以使用拉格朗日乘子法（将有约束条件的优化问题转换成无约束条件的优化问题），及优化原始目标函数（最大似然函数）与约束条件函数（这里我们采用（theta_1-theta_j）^2））的线性组合，这样我们可以得到在约束条件下最大似然函数的最优解。

4. 基于线搜索方法（Line Search methods）的最大似然函数优化

4.1 方向与步长的选择： Wolfe-Goldstein 条件与 backtracking（回溯法求步长）方法

Wolfe-Goldstein 条件（Wolfe-Goldstein conditions）是一种用于线搜索的条件，其目的是确保线搜索过程中的步长选择合适，既能保证足够的下降，又能避免选择过大的步长导致搜索方向不合理。选择合适的步长能够提高优化算法的收敛性和效率。Wolfe-Goldstein 条件通常包含两个部分：强 Wolfe 条件（strong Wolfe condition）和 Goldstein 条件（Goldstein condition）。

强 Wolfe 条件：强 Wolfe 条件是保证线搜索步长合适的一个重要条件。它包含两个不等式条件：

下降条件（Sufficient descent condition）： $f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k$

曲率条件（Curvature condition）： $|\nabla f(x_k + \alpha_k p_k)^T p_k| \geq c_2 |\nabla f_k^T p_k|$

其中，x_k 是当前迭代点，alpha_k 是步长，p_k 是搜索方向，f_k 是目标函数在 x_k 处的梯度 c_1 和 c_2 是取值范围在(0, 1)的常数。强 Wolfe 条件要求目标函数在给定步长下满足曲率条件，并且在足够的下降程度上保持。

Goldstein 条件：Goldstein 条件是强 Wolfe 条件的一种松弛形式。它包含一个单一的不等式条件：

$$(1 - c)\alpha_k \nabla f_k^T p_k f(x_k) \geq f(x_k + \alpha_k p_k) \geq f(x_k) + c\alpha_k \nabla f_k^T p_k$$

其中，c 是一个取值范围在(0, 1/2)的常数。Goldstein 条件要求目标函数在给定步长下满足在目标函数值上的一定降低。

Backtracking 在线搜索的过程中主要起到控制步长的作用，可以根据以上 Wolfe-Goldstein 或者 Armijo 条件来确定步长，其具体算法如下：

Algorithm 1 Backtracking Line Search

Require: $f, g, \alpha, \beta, \sigma, tol$

$x \leftarrow x_0$

while not converged **do**

  $d \leftarrow -g(x)$ ▷ Search direction

  $\alpha_{max} \leftarrow +\infty$

  $\alpha \leftarrow 1$

  while $\alpha > 0$ **do** $x' \leftarrow x + \alpha d$ **If** $f(x') \leq f(x) + \alpha\sigma g^T(x)d$ ▷ Sufficient decrease condition $x \leftarrow x'$ $\alpha_{max} \leftarrow \alpha$ $\alpha \leftarrow \beta\alpha$ **else** $\alpha \leftarrow \alpha/\beta$

  $\alpha \leftarrow \min(\alpha_{max}, 1)$

4.2 线搜索方法的亚类：搜索方向的选择

搜索方向：

梯度下降法： $-\nabla f(x_k)$ 当前位置负梯度方向（当前位置的最快下降方向）

牛顿法： $-\nabla^2 f(x)^{-1}\nabla f$ (考虑当前下降速度，以及迭代一次后下降速度)

拟牛顿法：（BFGS）： $-B_k\nabla f(x_k)$ （B_k为 Hessian 矩阵求逆的近似）

共轭梯度法： $-\nabla f(x_k) + \beta_k s_{k-1}$ (s_{k-1} 为上一步搜索方向，beta 为参数)

当然，每一类算法还有很多变种，可以增快收敛速率，减少计算量和占用内存等，我们会在之后简要介绍和讨论。

4.3 梯度下降法在最大似然函数优化上的具体实践

梯度下降为最常见与简单的方法—由于它的搜索方向即为其所在位置梯度的复制，所以这样的搜索是“最速”的，其具体算法如下（详细代码请见附录）：

Algorithm 2 Steepest Descent Method

Require: f, x_0, tol

$x \leftarrow x_0$

while not converged **do**

  $g \leftarrow$ Compute gradient of f at x

  $\alpha \leftarrow$ Choose step size using line search

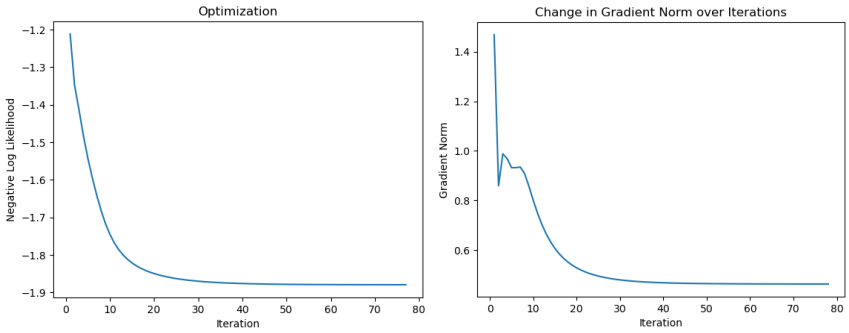
  $x \leftarrow x - \alpha g$ ▷ Update x

4.3.1 方法收敛性与收敛结果（convergence rate and result ）

我们将最大似然函数的优化放在 GPU（CUDA）上进行，考虑到设备显卡的显存，我们先选择了较小的数据集进行似然函数。我们选择了前两百个不同的 PNR（顾客），即近两万条数据，大概是指导教师 Yufeng Cao 在其论文中使用的完整数据集的 1/50.因此其泛化性有待加强。同时也是基于有限的显存每种方法的迭代次数有一定限制，不过其收敛还是有所保证。

其具体使用了较弱的 Goldstein 条件，其中 c 设置为 0.2，以避免在线搜索上的时间与内存过大成本。其余参数设置为：初始步长：1e-1；优化起点；theta = torch.ones(10)；线搜索中收缩因子：0.9；最大迭代次数：100。

其数值优化结果如下：



可以看到虽然优化函数不是理想的凸函数，最速下降法还是能保证其较快的收敛性，最大似然函数及其梯度在第二十次迭代后其速度明显减缓，梯度最终趋向于零，且输出如下：

十个自变量的权重：Optimized theta: [1.0230004 1.2535872 1.1529996 0.3462587 1.0521692 1.1500072 0.98690915 1.0261024 1.421194 0.8518269]

最终梯度 norm of gradient 0.46328047; Minimum all_likelihood: -1.8798

4.4 牛顿法和拟牛顿法（Quasi-Newton method）中 BFGS 在最大似然函数优化上的具体实践

以下分别为牛顿法和 BFGS 用于优化的具体算法：

Algorithm 3 Newton's Method

Require: $f, \nabla f, \nabla^2 f, x_0, tol$

$x \leftarrow x_0$
while not converged **do**
  $\nabla f(x) \leftarrow$ Compute gradient of f at x
  $\nabla^2 f(x) \leftarrow$ Compute Hessian matrix of f at x
  $\Delta x \leftarrow -\nabla^2 f(x)^{-1} \nabla f(x)$ ▷ Compute Newton step
  $x \leftarrow x + \Delta x$ ▷ Update x

Algorithm 4 Quasi-Newton-BFGS

Require: f, x_0, tol

Initialize H_0 as an approximation of the inverse Hessian matrix
$x \leftarrow x_0$
while not converged **do**
  $g \leftarrow$ Compute gradient of f at x
  Solve $H_k \Delta x_k = -g$ for Δx_k
  Line search: Choose step size α_k that satisfies the sufficient decrease condition
  $x \leftarrow x + \alpha_k \Delta x_k$ ▷ Update x
  Update H_{k+1} using BFGS update formula

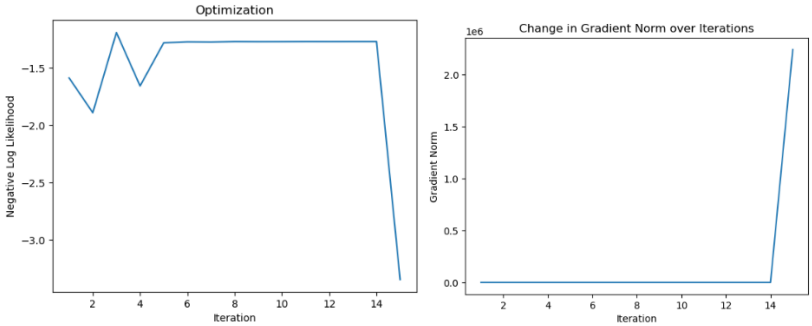
4.4.1 方法收敛性与收敛结果（convergence rate and result）

由于牛顿法和拟牛顿法的思想上的类似性，我们选择了直接使用计算量较小的拟牛顿法中的 BFGS 法，且参数与 3.3.1 中设置基本一致。

经过数值实验我们发现拟牛顿法对于优化最大似然函数有以下三个问题。

1. 线搜索方法对 BFGS 方法并不是很适用，往往需要将步长收缩到很小才可以跳出 backtracking 的循环，这会使得优化效率极其低下，故我们需要用其他方式调节步长，在这里我们采用最一般的步长衰减策略：每 20 个 iterations 乘上衰减系数 0.98.
2. 牛顿法和拟牛顿法对于高维非线性问题如本问题不稳定，需要增加最大循环次数以及进行梯度裁剪。我们自定义了 normalize_gradient 函数，对二范数模长大于 1 的梯度向量进行了归一化处理（即用原向量除以范数）以防止梯度爆炸。在实验中，我们常常注意到在某一个迭代步之后，因变量（最大似然函数），自变量（theta）与其梯度均变成了

nan 格式，经检查此时往往是发生了梯度爆炸，如下面两图（步长为 14 时梯度达到两万以上）。



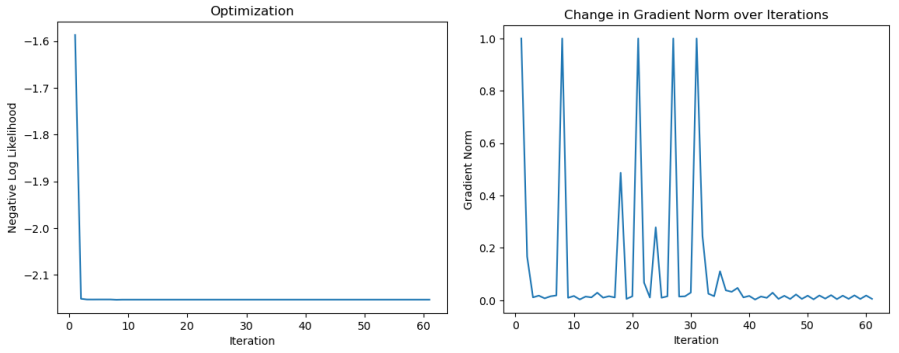
3. 与最速下降法相比，其初始下降速度很慢，所以增大初始步长到 5*1e-1

解决以上问题后，此方法有较好收敛性：到迭代六十次左右，梯度在 0.01 上下波动，函数的每次变化值已经到了 e-07 左右

last print：Iteration: 60, norm of gradient: 0.013105966970324516；

all_likelihood: tensor([-2.1529], device='cuda:0', grad_fn=<AddBackward0>), decrease: tensor([-9.5367e-07], device='cuda:0', grad_fn=<SubBackward0>)）

最终权重：[1.0701, 1.7839, 1.5350, -1.3129, 1.1366, 1.4852, 1.0251, 1.0586， 1.9668, 0.7266]，整个迭代结果如下：



4.5 共轭梯度法在最大似然函数优化上的具体实践

以下为共轭梯度法用于优化的具体算法：

Algorithm 5 Conjugate Gradient Descent

Require: A, b, x_0, tol

$r_0 \leftarrow b - Ax_0$ ▷ Compute initial residual

$p_0 \leftarrow r_0$ ▷ Initialize search direction

$k \leftarrow 0$

while not converged **do**

  $\alpha_k \leftarrow \frac{r_k^T r_k}{p_k^T A p_k}$ ▷ Compute step size

  $x_{k+1} \leftarrow x_k + \alpha_k p_k$ ▷ Update x

  $r_{k+1} \leftarrow r_k - \alpha_k A p_k$ ▷ Update residual

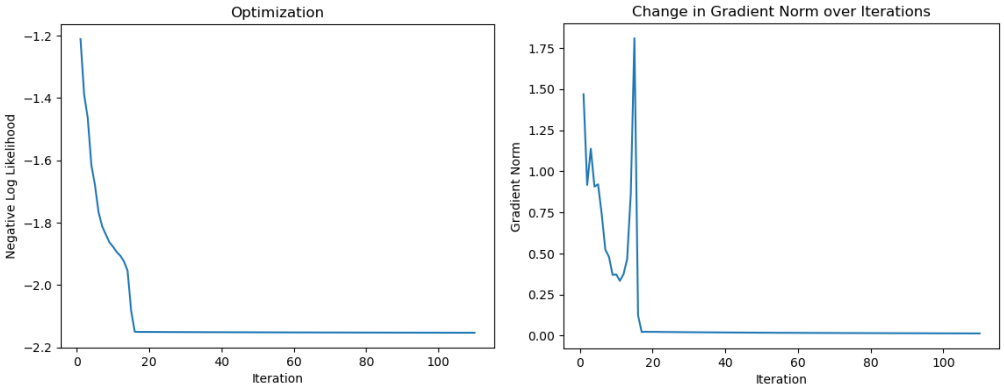
  $\beta_k \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$ ▷ Compute conjugate direction coefficient

  $p_{k+1} \leftarrow r_{k+1} + \beta_k p_k$ ▷ Update search direction

  $k \leftarrow k + 1$

4.5.1 方法收敛性与收敛结果（convergence rate and result）

本方法中参数设置与 4.3.1 中相同，最终输出为（theta 在 1e-4 的量级上还有波动）
iteration: 109, norm of gradient: 0.013049105182290077, 权重：[1.0289, 1.5242, 1.3209, -0.4316, 1.0829, 1.3424, 0.9630, 1.0311， 2.0418, 0.7209], device='cuda:0', grad_fn=<AddBackward0>),all_likelihood： -2.1532], device='cuda:0', grad_fn=<AddBackward0>), decrease: tensor([-1.5259e-05], device='cuda:0', grad_fn=<SubBackward0>)
iteration: 110, norm of gradient: 0.012988166883587837, 权重：[1.0289, 1.5246, 1.3209, -0.4321, 1.0829, 1.3424, 0.9630, 1.0311, 2.0428, 0.7207], device='cuda:0', grad_fn=<AddBackward0>),all_likelihood: tensor([-2.1532], device='cuda:0', grad_fn=<AddBackward0>), decrease: tensor([-1.5497e-05], device='cuda:0', grad_fn=<SubBackward0>)，整体下降趋势如下：



4.6 不同线搜索方法的比较与数学原理解释

算法	优势	劣势
梯度下降法	简单易实现；对大规模数据集和高维问题有效；收敛到局部最小值（或全局最小值，具体取决于问题和初始点）	收敛速度慢；可能陷入局部最小值或鞍点；对于病态问题（条件数高）效果较差
牛顿法	收敛速度快；可以直接求解二阶导数（Hessian矩阵）；通常在接近最优解时表现良好	需要计算和存储Hessian矩阵（二阶导数），在高维问题上代价高；Hessian矩阵可能是非正定的，导致算法不稳定
拟牛顿法（LBFGS）	无需计算和存储完整的Hessian矩阵；近似Hessian矩阵的更新可以通过梯度计算得到；在大规模问题上比牛顿法更有效	对于高度非线性的问题，可能收敛速度较慢；需要额外的存储空间来存储历史梯度信息
共轭梯度法	对于二次凸优化问题，收敛速度快；内存消耗较低；适用于大规模问题	只适用于无约束优化问题；对于非二次凸问题的收敛性和稳定性较差；需要确保问题是对称正定的

从数值分析的角度来看，牛顿法是二阶收敛，梯度下降是一阶收敛，所以牛顿法就更快，如 5.2 中所述，假设目标是找一条最短的路径走到一个盆地的最底部，梯度下降法每次只当前所处位置选一个坡度最大的方向走一步，牛顿法在选择方向时，不仅会考虑坡度是否够大，还会考虑你走了一步之后，坡度是否会变得更大。所以，在某些情况下能更快地走到最底部。

而从几何的角度来看，对于这个“盆地”，牛顿法就是用一个二次曲面去拟合所处位置的局部曲面，而梯度下降法是用一个平面去拟合当前的局部曲面，通常情况下，二次曲面的拟合会比平面更好，此时牛顿法选择的下降路径会更符合真实的最优下降路径。拟牛顿法和牛顿法类似，只是改善牛顿法每次需要求解复杂的 Hessian 矩阵的逆矩阵的缺陷（数值计算中矩阵求逆是一种成本很高的运算），它使用正定矩阵来近似 Hessian 矩阵的逆，从而简化了运算的复杂度。

在共轭梯度法中，可以将最优解和初始值的差表示为共轭向量基的线性组合，然后沿着解空间（令优化函数为零的向量空间）的维度逐步构建最优解，共轭梯度法在寻找每一个共轭向量时只需要利用上一个共轭向量，而不需要记住先前所有共轭向量。同样，如 5.2 中所述，每一次迭代用到的新方向是负残差和上一个搜索方向的线性组合。同样考虑的是一阶收敛，其避免

了最速下降法中的一些“之字形”的下降，相比这种情况下曲折的路径，显然使用共轭梯度法会更优更快。

而对于本函数，对比图象及输出的数据我们可以发现：在使用最速下降法时，下降曲线更为平滑，收敛速度较慢，最终似然函数甚至其梯度都被卡在了局部的一个最小值（后两者收敛到-2.1 左右，最速下降法收敛到-1.8 左右，梯度量级也明显大很多，一直停留在 0.4 左右）；拟牛顿法在控制梯度的情况下收敛速度非常理想，最终 theta 和似然函数基本都收敛到了一个定值，但可能由于函数本身的性质，梯度在后期一直有较小的波动；共轭梯度法的收敛性也比较理想，特别的，其梯度范数的收敛性是该三种方法中最好的，虽然本函数非凸函数本问题不属于凸优化范围，但是其表现还是良好，但是注意到不同于前两个，如 4.5.1 中的输出显示其后期一直存在 theta 的小范围波动，也许是迭代次数受限的原因。

在这三种线搜索方法的表现中，其最显著的差异其实是梯度的收敛性牛顿法，共轭梯度法这种“考虑了上一步和下一步”的做法其实可以某种程度上看作对一阶导（梯度）及其范数范数的一种优化.因此，接下来我们简要证明一下给定一定条件牛顿法的梯度范数的二次收敛性：

Claim：如果被优化函数的 Hessian 矩阵在最小值（记为 x*）附近符合局部 Lipchitz 连续，则其梯度范数具有二阶收敛性.

Proof:对于第 k 步迭代，其方向记为$p_k^N = -\nabla^2 f_k(x)^{-1}\nabla f_k$，则有

$$\begin{aligned} x_k + p_k^{\mathrm{N}} - x^* &= x_k - x^* - \nabla^2 f_k^{-1}\nabla f_k \\ &= \nabla^2 f_k^{-1}\left[\nabla^2 f_k(x_k - x^*) - (\nabla f_k - \nabla f_*)\right]. \end{aligned}$$

而

$$\nabla f_k - \nabla f_* = \int_0^1 \nabla^2 f(x_k + t(x^* - x_k))(x_k - x^*)\,dt,$$

由 lipchitz 连续性可知

$$\begin{aligned} &\left\|\nabla^2 f(x_k)(x_k - x^*) - (\nabla f_k - \nabla f(x^*))\right\| \\ &\quad= \left\|\int_0^1 \left[\nabla^2 f(x_k) - \nabla^2 f(x_k + t(x^* - x_k))\right](x_k - x^*)\,dt\right\| \\ &\quad\le \int_0^1 \left\|\nabla^2 f(x_k) - \nabla^2 f(x_k + t(x^* - x_k))\right\| \|x_k - x^*\|\,dt \\ &\quad\le \|x_k - x^*\|^2 \int_0^1 Lt\,dt = \tfrac{1}{2}L\|x_k - x^*\|^2, \end{aligned}$$

$$\|x_k + p_k^{\mathrm{N}} - x^*\| \le L\|\nabla^2 f(x^*)^{-1}\|\|x_k - x^*\|^2 = \tilde{L}\|x_k - x^*\|^2,$$

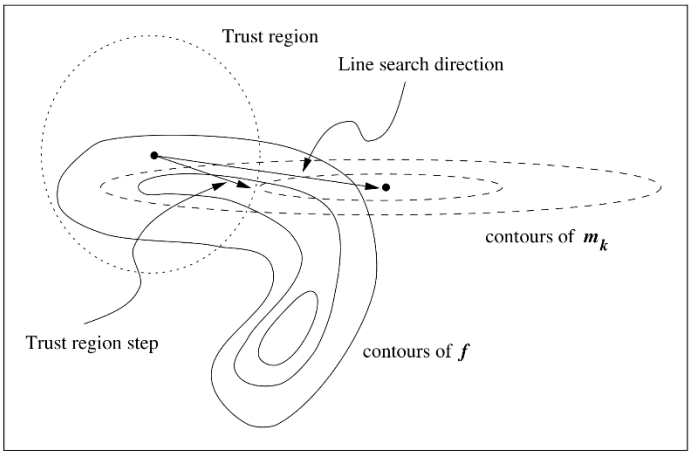
故

$$\begin{aligned} \|\nabla f(x_{k+1})\| &= \|\nabla f(x_{k+1}) - \nabla f_k - \nabla^2 f(x_k)p_k^{\mathrm{N}}\| \\ &= \left\|\int_0^1 \nabla^2 f(x_k + t p_k^{\mathrm{N}})(x_{k+1} - x_k)\,dt - \nabla^2 f(x_k)p_k^{\mathrm{N}}\right\| \\ &\le \int_0^1 \left\|\nabla^2 f(x_k + t p_k^{\mathrm{N}}) - \nabla^2 f(x_k)\right\| \|p_k^{\mathrm{N}}\|\,dt \\ &\le \tfrac{1}{2}L\|p_k^{\mathrm{N}}\|^2 \\ &\le \tfrac{1}{2}L\|\nabla^2 f(x_k)^{-1}\|^2\|\nabla f_k\|^2 \\ &\le 2L\|\nabla^2 f(x^*)^{-1}\|^2\|\nabla f_k\|^2, \end{aligned}$$

证毕

5. 基于置信域方法（Trust Region Methods）的最大似然函数优化

信赖域方法定义了当前迭代点周围的一个区域，在该区域选择合适的迭代步（包含方向和步长）作为模型在该区域中的最小值的近似。实质上，置信域方法同时选择方向和步长。如果迭代结果不理想，将会减小域的大小，然后搜索新的最小值。一般来说，每当置信域的大小改变时，迭代步的方向就会改变。如果该区域太小，则算法错过了采取使其更接近目标函数的最小值的迭代的机会；如果太大，模型的最小值可能难以收敛到区域中目标函数的最小值。



上图取自于 **Nocedal,Wright.Numerical Optimization**，其中 contour 是 f 的等高，m_k 是来自于线搜索方法中上一的相关信息，是 f 的等高线的一个近似（如同 5.7 中，若线搜索使用的是牛顿法 m_k 就是用一个二次曲面去拟合所处位置 f 的局部曲面,若使用梯度下降法 m_k 就是用一个平面去拟合 f 的局部曲面），可以看到通过线搜索方法，函数下降到 m_k 区域中的最小值；而给定一个如图置信域，函数可能有更大的下降。

5.1 置信域选择的基本算法

Algorithm 6 Trust Region Method for Optimization

Require: $f, \nabla f, \nabla^2 f, x_0, \Delta_0, \eta, tol$

$x \leftarrow x_0$
$\Delta \leftarrow \Delta_0$
while not converged **do**
  $\nabla f(x) \leftarrow$ Compute gradient of f at x
  $\nabla^2 f(x) \leftarrow$ Compute Hessian matrix of f at x
  Solve the trust region subproblem: Find Δx by solving
    $\min_{\Delta x} m(\Delta x) = f(x) + \nabla f(x)^T \Delta x + \frac{1}{2}\Delta x^T \nabla^2 f(x) \Delta x$
    subject to $\|\Delta x\| \leq \Delta$
  Compute the ratio: $\rho = \frac{f(x) - f(x+\Delta x)}{m(0) - m(\Delta x)}$
  if $\rho < \frac{1}{4}$ **then**
    $\Delta \leftarrow \frac{1}{4}\Delta$ ▷ Shrink the trust region
  else
    if $\rho > \frac{3}{4}$ and $\|\Delta x\| = \Delta$ **then**
      $\Delta \leftarrow \min(2\Delta, \Delta_{\max})$ ▷ Expand the trust region
  if $\rho > \eta$ **then**
    $x \leftarrow x + \Delta x$ ▷ Update x

以上选择置信域的算法的思路基本如下：显然在合理的置信域范围内ρ在任意点处为一正值，如果ρ为正值且在 1 附近，则可以扩大置信域进行下一步迭代，如果ρ负值或接近 0，则需要缩小置信域进行下一步迭代，可以从图上看出，这样能保证函数有理想的下降趋势。

5.1.2 迭代步的选择：Cauchy point

事实上，置信域方法在选定置信域后每一步可以归纳成解决以下问题-在置信域中寻找 p(迭代步)使得下式取最小值；

$$m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p$$

但与线搜索方法类似，我们往往每一步只需要取到一个近似的最小值，故可以使用 Cauchy 点方法，即在信赖域中取得如下步长：

$$p_k^C = -\tau_k \frac{\Delta_k}{\|\nabla f_k\|} \nabla f_k$$

$$m_k(p_k^C) = f_k + \nabla f_k^T p_k^C + \frac{1}{2} p_k^{C\,T} B_k p_k^C$$

$$= m_k'(\tau_k) = f_k + \nabla f_k^T \left(-\tau_k \frac{\Delta_k}{\|\nabla f_k\|} \nabla f_k\right) + \frac{1}{2}\left(-\tau_k \frac{\Delta_k}{\|\nabla f_k\|} \nabla f_k\right)^T B_k \left(-\tau_k \frac{\Delta_k}{\|\nabla f_k\|} \nabla f_k\right)$$

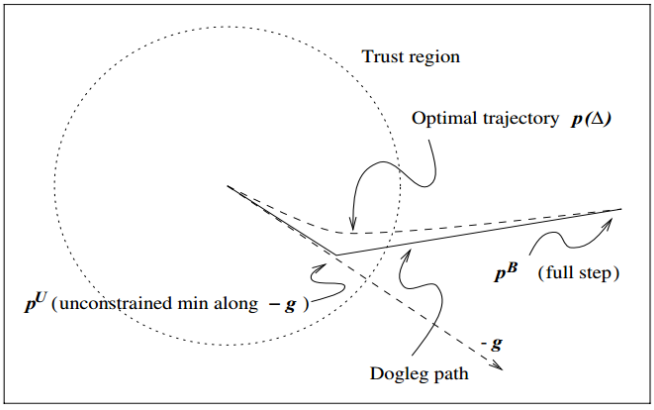
$$= f_k - \tau_k \frac{\Delta_k}{\|\nabla f_k\|} \nabla f_k^T \nabla f_k + \frac{1}{2} \tau_k^2 \frac{\Delta_k^2}{\|\nabla f_k\|^2} \nabla f_k^T \nabla B_k f_k$$

由于

故直接令步长达到信赖域的边界，则有

$$\tau_k = \begin{cases} 1 & \text{if } \nabla f_k^T B_k \nabla f_k \le 0 \\ \min\left(\|\nabla f_k\|^3 / \left(\Delta_k \nabla f_k^T B_k \nabla f_k\right), 1\right) & \text{otherwise.} \end{cases}$$

事实上，使用 Cauchy point 类似于置信域中的最速下降（所以置信域的选取对于收敛速度很重要），并且经过迭代它是全局收敛的。由于最佳步长往往取决于置信域的大小，所以我们采用了收敛更快的改进方法-Dogleg。它的性质是当信赖域半径很小时，比较接近于 Cauchy Point 方向，当信赖域半径比较大时，比较接近于无约束的 Cauchy Point 与类似于牛顿法的迭代步的 p_k 连线方向,如下图所示（图源：**Nocedal,Wright.Numerical Optimization**）：

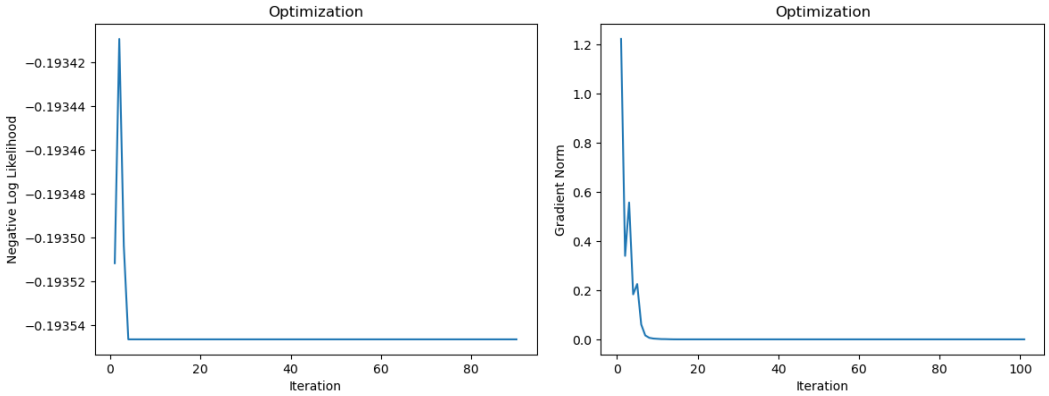


5.2 置信域方法基本算法在最大似然函数优化上的具体实践

5.2.1 方法收敛性与收敛结果（convergence rate and result）

在本问题中，置信域方法的参数设置为 initial_trust_radius=1；max_trust_radius=100（圆形置信域半径），其他参数如 rho 的取值边界和 5.1 代码中相同。

置信域方法在优化本似然函数时收敛过程略不稳定，整体收敛性良好，最终输出为 iteration:100,norm 0.0002140647266060114,theta: 权重：([-1.0325, 0.4109, 1.3469, 3.8814, 2.4630, 1.5611, 1.8308, 1.8020, 3.3401, 2.6272], device='cuda:0', grad_fn=<AddBackward0>),all_likelihood tensor([-0.1935], device='cuda:0', grad_fn=<AddBackward0>),其具体优化过程如下；



6 总体优化结果及其评估

6.1 不同优化方法预测权重的差异

我们通过计算向量夹角来评估四种优化方法优化结果的差异度（这是自然的，虽然作为 all_likelihood 的函数自变量不同的 theta 向量取值意味不同，但由于其实际意义是权重，所以同一个方向的向量都是等价的），结果如下（单位为度）：

夹角	最速下降法	拟牛顿法	共轭梯度法	置信域法
最速下降法	0	25.13	15.02	43.68
拟牛顿法	25.13	0	12.11	62.98
共轭梯度法	15.02	12.11	0	52.76
置信域法	43.68	62.98	52.76	0

我们可以看到三种线搜索方法差异较小，特别是 LBFGS 和共轭梯度下降，他们的实际优化得到的函数最小值差异也最小（事实上这有一定的数学解释，将共轭梯度法中的搜索方向写成矩阵左乘梯度，则该矩阵经过最自然的对称和正定化处理后正好和 LBFGS 中矩阵相同（考虑一次迭代））。置信域法和另外三种方法差异均较大，又由于其最终值大于线搜索方法收敛到的值，且此时梯度很小，所以应该是收敛到了局部最小值。事实上线搜索方法和置信域方法在对于优化函数的适应性上有以下区别：

	优势	**劣势**
线搜索方法	适用于具有全局性质的函数，能处理非光滑函数，能处理不连续点和函数不光滑的情况	可能陷入局部极小值
置信域方法	适用于具有局部性质的函数，在局部极小值之间跳跃，可能更稳定	不适用于非光滑函数，不适用于存在不连续点的函数，可能无法找到全局最优解

所以对于例如含有绝对值或 max/min 运算的函数，或函数存在不连续点，例如跳跃或间断点，置信域方法可能无法正确处理这些情况，导致优化过程发生错误。由于似然函数是对数函数的累加，所以也可能存在这些潜在的问题。

6.2 优化结果在测试集上的准确度

基于上述讨论，我们选择优化到最小值的拟牛顿法的权重结果进行测试。

由于 booked_itin 是一个非常稀疏的参数（不适用于一般的 acc 计算或者绘制 ROC 曲线的评估方法），我们考虑对于一个 pnr 的选择集合中的所有向量。每一个向量与权重相乘的值被命名为 preference（因为其值越大，越被 preferred），而我们考虑的是 booked_itin=1 时的 preference 和选择集中所有 preferences 中的最大值的相近程度。根据下图（不完整数据），我们可以看到 Booked_Itin=1 preference 和 Max Preferences 是相近（相较于和 Min Preferences 的差值）甚至相

等（完美预测了旅客选择）的。所以基于对此的观察，优化得到的权重结果在测试集上是较准确的。

PNR	Max Preferences	Min Preferences	Booked_ltin=1 preference
6fbae232589745c9be9f23188f4f72c7	8.119897842	3.065918207	8.119897842
db75a5d8b2d43ac556678ba379b97de6	10.3237896	5.480977535	9.082519531
e1ff559c2e69f00fb58378ab13b5e189	20.79385757	15.96762753	17.62943649
0a731a446bc007ccb0311ee6ea0b90ed	12.05859756	7.004618645	12.05752754
f6284de3df5aa32034442e54a6bbaab8	11.60355854	6.777329445	11.60355854
02b2bf8441bf4040ffee8eed75242abe	15.99729729	10.94331741	15.99622726
7b0eb8469da0d5acb2929d688767823c	15.54225922	10.59393597	13.59167194
ccdc3a8008ff50ea9dbe761a9c9e1aaa	10.3237896	5.480977535	9.082519531
486f4314fa617d7b11aeb57d170da01a	10.29065895	5.464428902	9.049388885
f7e417195d0cd155d6454e49104675e8	6.385090351	1.417672873	5.143820286
09dcb627474425e4ed1135b9055e00e4	6.351958752	1.403636456	5.110688686
97adcfdaee54eb54b954cc04baaef3c5	9.432797432	4.222218037	8.191527367
b555a88be2c4f63ea3518abe73bc114a	7.664858818	2.716536522	5.714272022
05da07bfd4f4d01eb03e3f006e1616fa	8.119897842	3.065918207	6.878627777
8c43bc6a10444b2ecd96b44b8e116a50	6.806997776	1.596418381	5.565727711
e8894a31257b1f479383700607bdf232	6.806997776	1.596418381	5.56679821
804fcb716217e8d546f52fd095011c58	6.351958752	1.403636456	5.110688686
5a5951288071ad40251ff311c8309b5f	13.37149715	8.317518234	10.34282684
6042156d59e1a6b72e9243580a5cf108	13.37149715	8.317518234	10.34632683
6a6ddb04892372ce8bd3f5c686b793dd	12.05859756	7.004618645	12.05859756
0f8042a0a9ddbd8cbfd8eecaaad45908	11.63669014	6.793878078	11.63669014
e539ce7ef770b01a4fd71d478dd79110	11.63669014	6.793878078	8.72894001
883fdcc1298faec2763d5871f9ee7f44	11.60355854	6.777329445	10.36335945
bfd350eefe8fb14775508ff696408b86	13.37149715	8.317518234	13.37042713
02956ab7d11d5580c363439f4c2b3c25	9.432797432	4.222218037	9.431727409
ae8ed5bfd8743f1a603e11ebaf77b620	7.664858818	2.838629007	7.664858818
d6bd28017ab1b5829f6f69d383d35b59	9.432797432	4.222218037	9.431727409
56f464728762694f6d3cded726fe4473	26.50049591	21.44651794	23.07381821
a25e8014ba2a66afc3b7cb6b1dbff63f	15.99729729	10.94331741	12.57061863
c293ecc7b8478f0ed42f13cab83107d0	15.54225922	10.71602821	12.37783623
fa9b3e3c55ed91d818f5c0eb6b5452db	13.37149715	8.317518234	13.37042713
d3f21e1f8f6f215ec521c3135ac460c0	6.806997776	1.596418381	6.805927753
413762d7be446d1f5c0584d781020a76	6.806997776	1.596418381	5.565727711
7050c0569bc19c02405485a38b4ccfba	13.37149715	8.317518234	13.37042713
ca2a56841b35ee910de3486d4633d3cf	10.74569702	5.691718102	10.744627
0ce16d1cfa695c23cfafac8d15f23f18	10.74569702	5.691718102	10.744627

6.2 基于优化结果对航空旅客决策因素的讨论与结论

基于我们的验证，对于特征[Cancel_Fee,is_cheapest,is_refundable,book_DtD ,change, book_channel_idx_1,book_channel_idx_2,book_channel_idx_3,dept_hour_idx_0 ,dept_hour_idx_1]，[1.0701, 1.7839, 1.5350, -1.3129, 1.1366, 1.4852, 1.0251, 1.0586， 1.9668, 0.7266]是一个合理的权重其中我们可以看到乘客有明显的购买最便宜机票的倾向，相比之下退改政策没有那么重要，在所有与退改相关的政策中，可退款是更被看中的一个因素。购票时间对决策的影响一般，其值为负数是因为我们没有为负的原始数据取绝对值。在三个购票渠道中，乘客对购票渠道 2 有偏好，这和渠道本身与其提供的机票价格均有关。dept_hour_idx_0 ,dept_hour_idx_1 的较大差异，是本模型的较明显的一个不足，可能在于我们使用的拉格朗日乘子法条件较弱（系数较小），没有能很好地约束同一个量各个 index 的相近程度，也有在我们的采样集中两个 index 可能分布不均匀的原因。

7.致谢

感谢曹宇峰老师耐心地为我们答疑解惑以及提供各方面的帮助，也非常感谢小组成员们给我提供跨专业交流和相互学习的机会，以上

8.参考文献

① Cao, Kleywegt, & Wang, Network Revenue Management under a Spiked Multinomial Logit Choice Model

② Dai, Ding, Kleywegt, Wang, & Zhang, Choice Based Revenue Management for Parallel Flight

③ Nocedal,Wright , *Numerical Optimization*, springer

A. 附录：

a. 最大似然函数代码

```
def all_likelihood(theta):
    all_likelihood = 0.0
    theta = theta.to(device)
    theta_exp = torch.exp(theta)
    def theta_dot_x(x):
        return torch.matmul(x, theta_exp)  # x是数据,x=torch.stack([Cancel_Fee,is_cheapest,is_refundable,book_DtD...])

    for pnr_id in selected_PNR:#将个人选择集打包，这里selected_PNR = data['PNR'].unique()[:1000]就是前一千个旅客
        indices = data[data['PNR'] == pnr_id].index
        y=torch.index_select(x,0,torch.tensor(indices).to(device))
        booked_1_indices=torch.nonzero(Booked_Itin[indices]==1)[:,0].to(device)#找出顾客所作决策的索引
        z=theta_dot_x(y)
        # Calculate choice probability
        choice_prob = torch.exp(torch.index_select(z.to(device), 0, booked_1_indices))/torch.sum(((torch.exp(z.to(device)))+1))
        choice_prob = torch.clamp(choice_prob, 1e-10, 1.0)  # 避免概率为0
        log_likelihood=torch.log1p(choice_prob)#避免torch.log的不稳定性
        # Compute log likelihood
        all_likelihood += log_likelihood

    # Compute gradients
    all_likelihood = -all_likelihood + 0.0001*(torch.sum(torch.pow((theta[5]-theta[6]), 2))+torch.pow((theta[6]-theta[7]), 2)
                                               +torch.pow((theta[9]-theta[8]), 2))#使用拉格朗日乘子法，取相反数以优化得到最大值
    all_likelihood = all_likelihood.to(device)
    return all_likelihood
```

b. 线搜索代码

```
# Backtracking line search
while True:
    direction = -gradient#direction由优化方法决定，这里最速下降法是是梯度的相反数，每一次循环都会被更新
    theta_new = theta + alpha * direction #迭代
    theta_new = theta_new.to(device)

    gradient_new = torch.autograd.grad(all_likelihood(theta_new), theta, create_graph=True)[0]#更新梯度

    while True:# backtracking过程
        sufficient_decrease = all_likelihood(theta_new) <= all_likelihood(theta) - torch.abs(c1 * directional_derivative)
        #directional_derivative = torch.dot(gradient, direction)
        goldstein_condition = all_likelihood(theta_new) >= all_likelihood(theta) - (1 - c1) * torch.abs(directional_derivative)

        if sufficient_decrease and goldstein_condition:#goldstein condition是这两者
            break
        else:
            alpha *= beta #beta为收缩因子

    # Update theta and gradient
    theta = theta_new
    gradient = gradient_new
    gradient = gradient.to(device)
    gradient_norms.append(torch.norm(gradient).item())
    print(f"iteration: {iteration}, norm of gradient: {torch.norm(gradient)}, theta: {theta},all_likelihood: {all_likelihood(theta)}, decrease
    iteration += 1
    itr.append(iteration)
    likelihoods = all_likelihood(theta).detach().cpu()
    loglikelihood.append(likelihoods)
    torch.cuda.empty_cache()  # q清理gpu
    if torch.norm(gradient) < threshold or iteration >= 100:
        break
```

c. 置信域代码

```
def dogleg_method(Hk, gk, Bk, trust_radius):
    Hk=Hk.to(device)
    gk=gk.to(device)
    Bk=Bk.to(device)
    trust_radius=trust_radius

    # Compute the Newton point.
    # This is the optimum for the quadratic model function.
    # If it is inside the trust radius then return this point.
    pB = -torch.matmul(Hk, gk)
    norm_pB = torch.sqrt(torch.matmul(pB, pB))

    # Test if the full step is within the trust region.
    if norm_pB <= trust_radius:
        return pB

    # Compute the Cauchy point.
    # This is the predicted optimum along the direction of steepest descent.
    pU = - (torch.matmul(gk, gk) / torch.matmul(gk, torch.matmul(Bk, gk))) * gk
    dot_pU = torch.matmul(pU, pU)
    norm_pU = torch.tensor(sqrt(dot_pU))

    # If the Cauchy point is outside the trust region,
    # then return the point where the path intersects the boundary.
    if norm_pU >= trust_radius:
        return trust_radius * pU / norm_pU

    # Find the solution to the scalar quadratic equation.
    # Compute the intersection of the trust region boundary
    # and the line segment connecting the Cauchy and Newton points.
    # This requires solving a quadratic equation.
    # ||p_u + tau*(p_b - p_u)||**2 == trust_radius**2
    # Solve this for positive time t using the quadratic formula.
    pB_pU = pB - pU
    dot_pB_pU = torch.dot(pB_pU, pB_pU)
    dot_pU_pB_pU = torch.dot(pU, pB_pU)
    fact = dot_pU_pB_pU ** 2 - dot_pB_pU * (dot_pU - trust_radius ** 2)
    tau = (-dot_pU_pB_pU + sqrt(fact)) / dot_pB_pU

    # Decide on which part of the trajectory to take.
    return pU + tau * pB_pU
```

```
while True:
    trust_radius = initial_trust_radius
    gk = jac(theta).to(device)
    Bk = hess(theta).to(device)
    Bk_inverse=torch.linalg.inv(Bk)
    Hk =  Bk_inverse.to(device)

    pk = dogleg_method(Hk, gk, Bk, trust_radius).to(device)

    # Actual reduction.
    act_red = all_likelihood(theta) -  all_likelihood(theta + pk)

    # Predicted reduction.
    pred_red = -(torch.matmul(gk, pk) + 0.5 * torch.matmul(pk, torch.matmul(Bk, pk)))

    # Rho.
    rhok = act_red / pred_red
    if pred_red == 0.0:
        rhok = 1e99
    else:
        rhok = act_red / pred_red
    rhok=rhok.to(device)
    # Calculate the Euclidean norm of pk.
    norm_pk = sqrt(torch.dot(pk, pk))

    # Rho is close to zero or negative, therefore the trust region is shrunk.
    if rhok < 0.25:
        trust_radius = 0.25 * norm_pk
    else:
        # Rho is close to one and pk has reached the boundary of the trust region, therefore the trust region is expanded.
        if rhok > 0.75 and norm_pk == trust_radius:
            trust_radius = min(2.0 * trust_radius, max_trust_radius)
        else:
            trust_radius = trust_radius
    if rhok > eta: # Choose the position for the next iteration.
        theta =theta + pk
    else:
        theta = theta
    theta = theta.to(device)
    print(f"iteration:{k},norm {torch.norm(gk)},theta: {theta},all_likelihood{all_likelihood(theta)}")
    gradient_norm.append(torch.norm(gk))
    f_values.append(all_likelihood(theta))
    itr.append(k)

    # Check if the gradient is small enough to stop
    if torch.norm(gk) < gtol:
        break

    # Check if we have looked at enough iterations
    if k >= maxiter:
        break
```