本科实验报告

课程名称: 数据要素市场

学　　院: 竺可桢学院

系: 所在系

专　　业: 计算机科学与技术

姓名学号: 张晋恺 3230102400

姓名学号: 汪昕 323010xxxx

姓名学号: 施兴睿 3230102392

指导教师: 刘金飞

2025 年 7 月 18 日

目录

1 INTRODUCTION

1.1 项目概述

2 THEORY

2.1 理论部分

2.1.1 基于 UCB 的动态定价算法

2.1.2 基于 FTPL 的动态定价算法

3 EXPERIMENTS

3.1 实验部分 - UCB

3.1.1 基本结构

实验主要分为三个主要的 python 文件

- UCB.py: 主程序，负责调用其他模块
- utils.py: 工具函数，负责处理数据和模型
- visualization.py: 可视化函数，负责可视化结果

接下来依次介绍每个文件的实现思路

3.1.2 utils.py

估值函数

首先需要为玩家生成估值函数，其数学表达式为

$$v(n) = C * (1 - e^{-\beta * n})$$

其中，C 是估值函数的最大值，β 是估值函数的衰减系数。
这个函数当 n 趋近于无穷大时，$v(n)$ 趋近于 C，当 $n = 0$ 时，$v(n) = 0$。
而且满足 $v(n)$ 随 n 的增大而增大，并且增速越来越小，边际收益递减。
然后根据玩家的类型，生成不同参数的估值函数，用于模拟不同玩家的行为。
具体来说通过 base_rate 和 max_value 两个参数来控制估值函数的形状。

生成价格曲线集合

根据论文中的算法 2，生成一个离散化的价格曲线集合，而且是 $m - step$ 的，即跳跃次数不超过 m 次。

算法 2 的实现主要分为三个步骤：

步骤 1：构建数据空间网格 N_D 首先构建数据空间网格，用于离散化数据点数量 n：

- 设定阈值 $threshold\_n = \lfloor \frac{2Jm}{\epsilon^2} \rfloor$
- 对于 $n \leq threshold\_n$ 的数据点，不进行离散化，直接添加到网格中
- 对于 $n > threshold\_n$ 的数据点，采用对数离散化：
  - 计算 Y_i 点：$Y_0 = threshold\_n$，$Y_{i+1} = Y_i \cdot (1 + \epsilon^2)$
  - 在相邻的 Y_i 点之间插入约 $2Jm$ 个内点
  - 使用线性插值生成内点：$inner\_points = linspace(Y_i, Y_{i+1}, 2Jm)$
- 最终将 0 也加入网格中，确保包含不购买的情况

步骤 2：构建估值空间网格 W 然后构建估值空间网格，用于离散化价格水平：

- 计算 Z_i 点：$Z_0 = \epsilon$，$Z_{i+1} = Z_i \cdot (1 + \epsilon)$
- 在相邻的 Z_i 点之间进行细粒度插值：

- 计算间隔：$gap = Z_i \cdot \frac{\epsilon}{m}$
- 计算插值步数：$num\_steps = \lfloor \frac{Z_{i+1} - Z_i}{gap} \rfloor$
- 使用线性插值生成内点：$inner\_points = linspace(Z_i, Z_{i+1}, num\_steps)$

- 最终将 0 也加入网格中，确保包含免费价格

步骤 3：生成 m-阶梯价格曲线 基于上述两个网格，随机采样生成 m-阶梯价格曲线：

- 首先添加一条”免费”价格曲线（全零价格），这是算法 3 第一轮必需的
- 对于每条新的价格曲线：
  - 随机决定阶梯数 $k \in [1, m]$
  - 从 N_D 中随机选择 k 个跳变点 $jump\_points\_n$
  - 从 W 中随机选择 k 个价格水平 $price\_levels$
  - 构建阶梯价格曲线：

$$p(n) = \begin{cases} price\_levels[0] & \text{if } 0 \le n \le jump\_points\_n[0] \\ price\_levels[1] & \text{if } jump\_points\_n[0] < n \le jump\_points\_n[1] \\ \vdots & \\ price\_levels[k-1] & \text{if } jump\_points\_n[k-2] < n \le jump\_points\_n[k-1] \\ price\_levels[k-1] & \text{if } n > jump\_points\_n[k-1] \end{cases}$$

这种离散化方法确保了价格曲线集合的大小是多项式级别的，同时保持了足够的表达能力来近似最优解。

生成估值函数集合

在 UCB 算法中，需要预先计算每个 (价格曲线, 买家类型) 组合的购买决策和收入，这模拟了 Agent 拥有的先验知识。

购买决策计算 对于每个价格曲线 p 和买家类型 i，计算最优购买量：

- 计算所有可能购买量 n 的效用：$u_i(n) = v_i(n) - p(n)$
- 找到效用最大化的购买量：$n^* = \arg\max_n u_i(n)$
- 如果最大效用小于 0，则选择不购买（$n^* = 0$）
- 如果存在多个效用相等的最大值，根据论文规则选择最大的 n

收入计算 基于购买决策计算收入：

- 收入矩阵：$revenues[i,j] = p_j(n^*_{i,j})$，其中 p_j 是第 j 条价格曲线，$n^*_{i,j}$ 是类型 i 买家在价格曲线 j 下的最优购买量
- 购买量矩阵：$purchases[i,j] = n^*_{i,j}$，记录每种组合的购买量

这种预计算方式确保了算法在运行时能够快速查询任何价格曲线和买家类型组合的预期收入，为 UCB 算法的决策提供基础数据。

3.1.3 UCB.py

Config

配置类负责管理实验参数和生成真实的买家类型分布：

- **基本参数**：
  - M_TYPES：买家类型数量（默认 10）
  - N_ITEMS：数据点总数（默认 100）
  - $T\_HORIZON$：总回合数（默认 $M\_TYPES \times 10000$）
  - J：收益递减常数（默认 2.0）
  - ϵ：近似精度参数（默认 0.1）
  - $num\_samples$：生成的价格曲线数量（默认 $\max(7, M\_TYPES + 5)$）
- **真实分布生成**：根据买家类型数量自动生成合理的概率分布
  - 对于 1-4 种类型，使用预设的分布
  - 对于更多类型，使用高斯分布生成后归一化

UCBAgent

实现了论文中算法 3 的 UCB 算法，主要包含以下函数：

初始化

- 初始化内部状态：$t = 1$（当前回合数）
- T_i：类型 i 被”探索”的次数
- N_i：类型 i 被实际观察到的次数
- 历史记录字典：用于存储学习过程数据

choose_action　根据 UCB 规则选择价格曲线：

1. 第一轮强制选择免费价格曲线（索引 0）
2. 计算经验估计：$\bar{q}_i = \frac{N_i}{T_i}$
3. 计算置信度奖励：$\sqrt{\frac{\log T}{T_i}}$
4. 构造乐观估计：$\hat{q}_i = \bar{q}_i + \sqrt{\frac{\log T}{T_i}}$
5. 计算 UCB 收入：$\hat{rev}(p) = \sum_{i=1}^{m} \hat{q}_i \cdot revenue(p,i)$
6. 选择 UCB 收入最高的价格曲线

update　根据环境反馈更新内部状态：

- 更新探索次数：$T_i \leftarrow T_i + 1$（对于所有会购买类型 i 的买家）
- 更新观察次数：$N_i \leftarrow N_i + 1$（如果本轮观察到类型 i 买家）
- 记录历史数据：回合数、动作、收入、买家类型等

3.1.4　visualization.py

最后根据记录到的历史数据，绘制出 UCB 算法的可视化表现，代码不再赘述，接下来展示结果

3.1.5 **实验结果**

图 3.1: UCB 算法结果

可以看到，学习到的估计和真实分布非常接近，说明 UCB 算法能够很好地学习到真实分布。

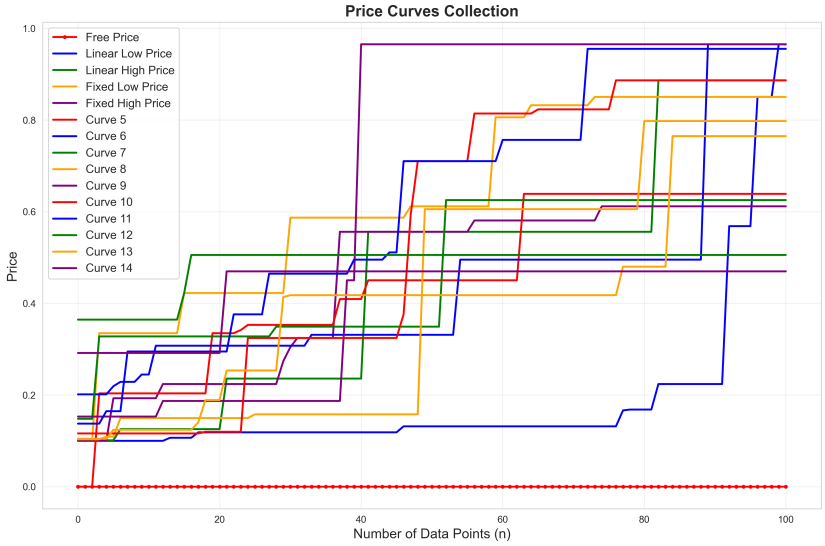

图 3.2: 价格曲线

可以看到价格曲线确实是阶梯状的，符合论文中的描述。

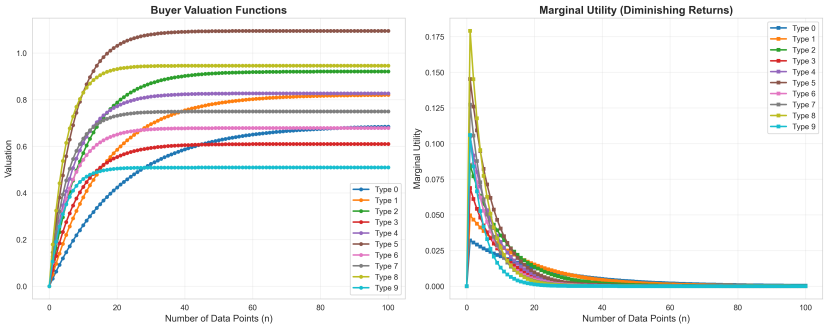

图 3.3: 估值函数

估值函数也与满足我们的要求

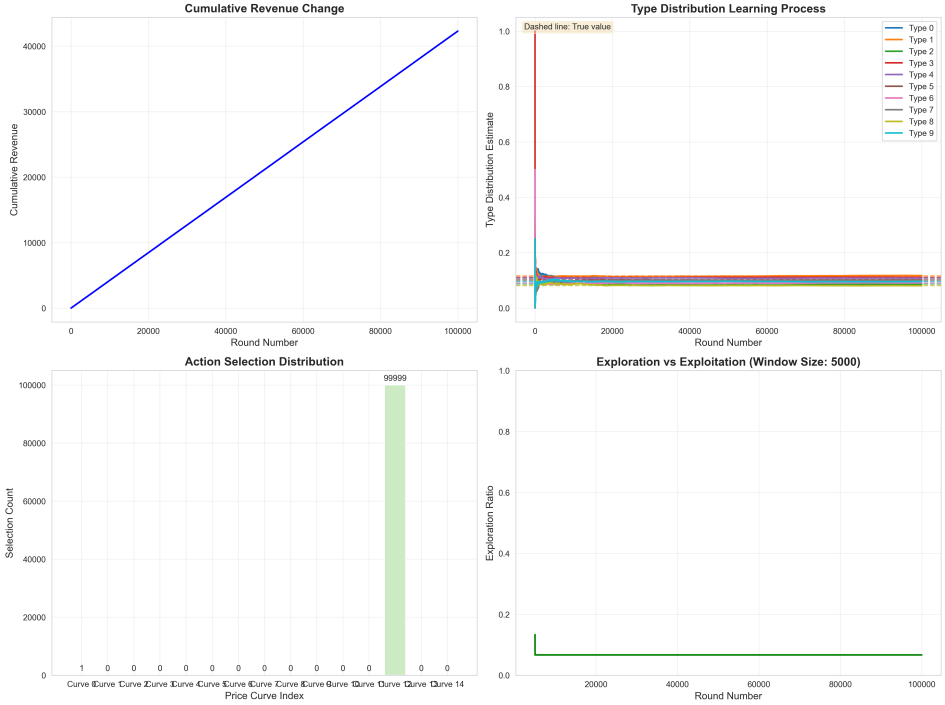

图 3.4: 学习过程

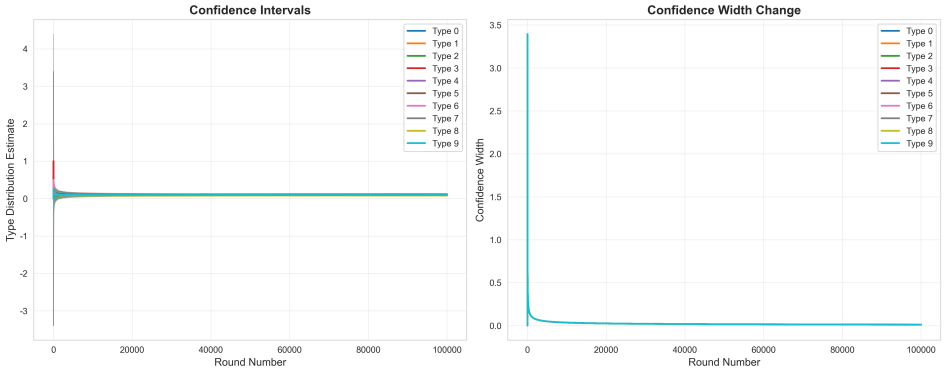

图 3.5: UCB 置信界

随着回合数的增加，置信界逐渐缩小，也满足我们的预期；

3.2 **实验部分** - FTPL