DOI: 10.54254/3049-5660/2025.26876

博弈论中的新发现

刘天琦

（树德实验中学（东区），四川省成都市，610002；13088083868@163.com）

摘　要：本文突破经典博弈中“双方完全理智”的假设，引入概率参数 c1，c2，...，cn 描述玩家对各策略的选择概率（简称“ c 计发”），从经典博弈论出发：首先系统性梳理纳什均衡的现有计算方法（含基于梯度的方法、分布式计算方法、量子退火算法等），明确各方法的核心逻辑与适用场景；随后以囚徒困境、石头剪刀布两个经典博弈为例展开演示，通过构建 n×m 博弈收益矩阵推导玩家总收益计算公式，借助拉格朗日函数处理“概率和为 1 ”的约束条件，设计投影梯度法实现策略概率的迭代优化；同时，通过树结构描述多次博弈的状态过程，为后续引入动态规划思想、探索模型的记忆化拓展（以适配动态博弈场景）奠定基础，其完整推导将在后续研究中展开；最后通过 Python 工具开展多组迭代实验（如 100 次初始随机化计算、调整迭代次数至 10000 次验证收敛性），以统计结果证明模型能有效收敛至经典博弈的纳什均衡解，验证模型正确性。

关键词：博弈论；数学；概率论；梯度下降

引言

在这里，作者取消了经典博弈中双方都是完全理智的假设，通过使用 {c_1, c _ 2, c _ 3 . . . c _ n }描述一位玩家对于每种选择做出的概率，并用梯度下降进行对这些参数的调整。

1. 文献综述

纳什均衡作为博弈论中的关键概念，在众多领域都有着广泛应用。而对纳什均衡的计算方法研究也一直是学界的重要课题。

前人分别从以下几个方面有过研究：

1.1. 基于梯度的方法

在 “Approximating N - Player Nash Equilibrium through Gradient Descent” [1] 一文中，给出了一种基于纯策略最佳响应的到纳什均衡的距离度量，将计算纳什均衡有效地转化为通过梯度下降寻找该距离函数的全局最小值。具体而言，当效用函数为凸函数时，该过程以 $O\left(\frac{1}{T}\right)$，（中T为迭代次数）的速率收敛到纳什均衡。

1.2. 分布式计算方法

在 “Distributed Nash Equilibrium Computation With Uncertain Dynamics and Disturbances” [2] 里，考虑了受未知干扰和动力学影响的一阶和二阶玩家。基于一阶低通滤波器建立未知动力学估计器，用于估计未知动力学和干扰，再结合梯度类优化组件和共识组件，构建分布式计算方法。通过适当选择控制参数，玩家的行动和速度（针对二阶玩家）能分别收敛到纳什均衡附近的小球和零。

1.3. 量子退火算法相关计算

“Calculating Nash Equilibrium on Quantum Annealers” [3] 指出，寻找两人非合作博弈中的纳什均衡问题是一个有约束的双重二次优化问题。通过向纳什均衡的二次函数公式中添加惩罚项，可将其转化为二次无约束二进制优化（QUBO）公式，从而能在量子退火器上执行。

1.4. 微分进化算法改进计算

在 “一种改进的微分进化算法求解纳什均衡问题与广义纳什均衡问题” [4]中，基于相对占优策略概念挖掘，改进了一种求解非线性连续博弈的纳什均衡问题（NEP）和广义纳什均衡问题（GNEP）的微分进化算法。该算法通过随机选取三个不同位置的父代个体进行变异和交叉操作产生新个体，依据相对占优策略实现优胜劣汰，既保持种群多样性，又增强全局寻优能力。

1.5. 变样本方法计算随机稳定纳什均衡

“Variable - sample method for the computation of stochastic Nash equilibrium” [5] 提出一种用于计算随机稳定纳什均衡的变样本方法。在每次迭代中，用不同样本大小的样本平均近似来逼近目标函数。首先研究变样本框架下的压缩映射性质，在一些适度条件下，证明从该算法获得的聚点以概率 1 满足一阶均衡条件。此外，利用渐近无偏性条件证明算法的聚点收敛到不动点集，并证明算法的有限终止性质。

2. 方法

在这里，作者取消了经典博弈中双方都是完全理智的假设也就是“理性人假设”，重新引入参数 $c_1, c_2, c_3 \ldots c_n$ 并规定有规定1

$$\sum_{i=1}^{n} c_i = 1 \qquad \text{(规定1)}$$

为了方便下文讨论，这里称之为“ c 计发”。描述一位玩家对于每种选择做出的概率，并用梯度下降进行对这些参数的调整。

2.1. 引入

让我们深入浅出，首先考虑最简单的囚徒困境，在一般的博弈中，我们有“理性人假设”即玩家被认为是“理性的”，即他们会通过权衡利弊，选择能最大化自身 “效用”。下面这张图表是对其的解释（表 1）

表1 方法引入的例子

囚徒 A / 囚徒 B	合作，记为 c_{a1}	背叛，记为 c_{a2}
合作，记为 c_{b1}	双方各判 1 年（-1，-1）	囚徒 A 判 10 年，囚徒 B 判 0 年（-10，0）
背叛，记为 c_{b2}	囚徒 A 判 0 年，囚徒 B 判 10 年（0，-10）	双方各判 5 年（-5，-5）

如果我们使用“ c 计发”对这次博弈进行描述，则我们记“囚徒 A”选择“合作”的概率为 c_{a1} ，则因为规定一，我们不必引入 c_{a2} 记录“囚徒 A”选择“背叛”的概率，而是使用 $1 - c_{a1}$ 表示“囚徒 A”选择“背叛”的概率（为了在下文方便讨论，这里称这种优化技巧为“概率和优化”）

则同理，我们使用同样的方法给出“囚徒 B”选择“合作”与“背叛”的概率为 c_{b1} 与 c_{b2} ，立马使用“概率和优化”，不难将 c_{b2} 改写成 $1 - c_{b1}$ 。

考虑到经典博弈中“理性人假设”，我们不难知道上文中 $c_{a1} = 1$ $c_{b1} = 1$ ，它的实际意义表示为“囚徒 A ”会以 1 的概率选择“合作”，会以 0 的概率选择“背叛”，“囚徒 B ”同理为以 1 的概率选择“合作”，会以 0 的概率选择“背叛”，为了达到这种效果，我们下文再说。

这无疑符合经典博弈的结果。

2.2. 让我们考虑更为复杂的例子

我们在这里考虑另一个经典但又与“囚徒困境”不一样的博弈——石头剪刀布，它的博弈表如表 2所示，方便于讨论，这里假设两位玩家（玩家 A 和玩家 B）的收益规则为：获胜得 1 分，平局得 0 分，失败得 - 1 分。表格中每一格的第一个数值为玩家 A 的收益，第二个数值为玩家 B 的收益。

表2 更为复杂的例子

玩家 A \ 玩家 B	石头	剪刀	布
石头	（0，0）【平局】	（1，-1）【A 胜，B 负】	（-1，1）【A 负，B 胜】
剪刀	（-1，1）【A 负，B 胜】	（0，0）【平局】	（1，-1）【A 胜，B 负】
布	（1，-1）【A 胜，B 负】	（-1，1）【A 负，B 胜】	（0，0）【平局】

不难发现这个博弈的最大特点即为不存在纯策略纳什均衡，但这不影响我们使用“ c 计发”进行描述。

首先，记“玩家 A”选择出“石头”的概率为 c_{a1}，出“剪刀”的概率为 c_{a2}，出“布”的概率为 c_{a3}，但考虑到如果使用“概率和优化”对下文计算会造成些许困难，所以不进行“概率和优化”。

同理，给出“玩家 B”分别选择“石头”，“剪刀”，“布”的概率为 c_{b1} c_{b2} c_{b3}，同样不进行“概率和优化”。

所以答案非常显然，博弈收敛至 $\mathrm{c_{a1}} = \mathrm{c_{a2}} = \mathrm{c_{a3}} = \frac{1}{3}$ 且 $\mathrm{c_{b1}} = \mathrm{c_{b2}} = \mathrm{c_{b3}} = \frac{1}{3}$，同样满足经典博弈的结果。

2.3. 如何得到上述的结论

接下来的讨论都将以双人博弈进行展开。

首先，我们将表格改写成 $\mathrm{n} \times \mathrm{m}$ 的矩阵形式，也就是表3 的形式。

表3 博弈收益矩阵

Player a / Player b	$Action_1$	$Action_2$	...	$Action_{\mathrm{n}}$
	c_{a1}	c_{a2}		c_{an}
$Action_1$ c_{b1}	$(c_{a1} \bullet \mathrm{f_{a,1,1}}, c_{b1} \bullet \mathrm{f_{b,1,1}})$	$(c_{a2} \bullet \mathrm{f_{a,1,2}}, c_{b1} \bullet \mathrm{f_{b,1,2}})$	...	$(c_{an} \bullet \mathrm{f_{a,1,n}}, c_{b1} \bullet \mathrm{f_{b,1,n}})$
$Action_2$ c_{b2}	$(c_{a1} \bullet \mathrm{f_{a,2,1}}, c_{b2} \bullet \mathrm{f_{b,2,1}})$	$(c_{a2} \bullet \mathrm{f_{a,1,2}}, c_{b2} \bullet \mathrm{f_{b,1,2}})$	...	$(c_{an} \bullet \mathrm{f_{a,2,n}}, c_{b2} \bullet \mathrm{f_{b,2,n}})$
...	...	...	...	...
$Action_{\mathrm{n}}$ c_{bm}	$(c_{a1} \bullet \mathrm{f_{a,m,1}}, c_{bm} \bullet \mathrm{f_{b,m,1}})$	$(c_{a2} \bullet \mathrm{f_{a,m,2}}, c_{b1} \bullet \mathrm{f_{b,m,2}})$	...	$(c_{an} \bullet \mathrm{f_{a,m,n}}, c_{bm} \bullet \mathrm{f_{b,m,n}})$

我们把 c 计发的 c 表在第一行，第一列。

其余部分规定有如下计算方法：

对于第一个玩家有式1的计算：

$$m_{i,j}[a] = c_{aj} \bullet f_{a,i,j} \tag{1}$$

中 $m_{i,j}[a]$ 是矩阵中 Player a 的位置，c_{aj} 以在上文定义，$f_{a,i,j}$ 为 Player a 选择此位置的权重。

同理，对于第二个玩家有式2：

$$m_{i,j}[b] = c_{bj} \bullet f_{b,i,j} \tag{2}$$

中 $m_{i,j}[b]$ 是矩阵中 Player b 的位置，c_{bj} 以在上文定义，$f_{b,i,j}$ 为 Player b 选择此位置的权重。

则第一位玩家在这次博弈中的额总收益可以表示为式3：

$$I_a = \sum_{i=1}^{m} \sum_{j=1}^{n} m_{i,j}[a] \tag{3}$$

第二位玩家在这次博弈中的额总收益可以表示为式4：

$$I_{\mathrm{b}} = \sum_{i=1}^{m} \sum_{j=1}^{n} m_{i,j}[b] \tag{4}$$

一般的，我们认为两位玩家会尝试使得自己的利益最大化，使用我们上文所提的数学形式改写这个要求即为：

1. Player a 总会尝试找到一组 $c_{a1}, c_{a2}, c_{a3} \ldots c_{an}$ ，使得 $\sum_{i=1}^{n} c_{ai} = 1$ 且使得 $I_a = \sum_{i=1}^{m} \sum_{j=1}^{n} m_{i,j}[a]$ 最小。

2. Player a 总会尝试找到一组 $c_{b1}, c_{b2}, c_{b3} \ldots c_{bn}$ ，使得 $\sum_{i=1}^{n} c_{bi} = 1$ 且使得 $I_a = \sum_{i=1}^{m} \sum_{j=1}^{n} m_{i,j}[b]$ 最小。

接下来我们以 Player a 的视角尝试使用梯度下降进行计算。

首先，考察两次博弈的 $\mathrm{c}_1, \mathrm{c}_2$ ，记它们分别为向量 $\overrightarrow{A}, \overrightarrow{B}$ ，则定义价值函数如式5：

$$V(x) = \sum_{i=1}^{m} \sum_{j=1}^{n} x_j \bullet f_{a,i,j} \tag{5}$$

则两次博弈的价值分别为 $V\left(\overrightarrow{A}\right), V\left(\overrightarrow{B}\right)$ 。

在明确的场景后，函数与约束条件可整理为：

1． 目标函数： $f(x) = \sum_{i=1}^{m} \sum_{j=1}^{n} x_j \cdot c_{i,j} \left(x = (x_1, x_1, \ldots, x_n)\right)$ 是 n 维向量， $c_{i,j}$ 是 $m \times n$ 的常数矩阵

2． 约束条件： $\sum_{j=1}^{n} x_j = 1$ （当然我们考虑的是概率，所以 $x_j \geq 0$ ）。

首先简化目标函数，明确其与向量 x 的关系：

由于 x_j 与 i 无关，所以我们可以可交换求和顺序方便后续计算见式6：

$$f(x) = \sum_{j=1}^{n} x_i \cdot \left(\sum_{i=1}^{m} c_{i,j}\right) \tag{6}$$

令有式7

$$d_j = \sum_{i=1}^{m} c_{i,j} \tag{7}$$

则函数立马改写为式8的形式：

$$f(x) = \sum_{j=1}^{n} x_j \cdot d_j \tag{8}$$

其本质为： $\mathrm{f(x)}$ 是关于向量 x 的线性函数，可表示为向量内积形式： $f(x) = x \cdot d$ ，其中 $d = (, d_2, \ldots, d_n)^T$ 是常数向量。

接下来，带约束地求解方程的最优值，即在约束条件为 $\sum_{j=1}^{n} x_j = 1$ 的情况下，求解 $\mathrm{f(x)}$ 的极值。最大值还是最小值我们下文讨论。

首先，构造拉格朗日函数。

引入拉格朗日乘数 λ ，它是一个标量，将约束融入目标函数，参见式9：

$$\mathscr{L}(x, \lambda) = f(x) + \lambda \cdot \left(\sum_{j=1}^{n} x_j - 1\right) = \sum_{j=1}^{n} x_j d_j + \lambda \left(\sum_{j=1}^{n} x_j - 1\right) \tag{9}$$

万事俱备，开求最值。

首先求得 KKT 条件。

对 x_j 和 λ 求偏导并令其为 0 ，得到极值必要条件：

1. 对 x_j： $\frac{\partial \mathscr{L}}{\partial x_j} = d_j + \lambda = 0 \quad \Rightarrow \quad d_j = -\lambda$（对所有 j）；

2. 对 λ： $\frac{\partial \mathscr{L}}{\partial \lambda} = \sum_{j=1}^{n} x_j - 1 = 0 \quad \Rightarrow \quad \sum_{j=1}^{n} x_j = 1$（它正是约束条件）。

让我们继续深入分析一下：

从极值条件不难看出：

1. 若所有 d_j 相等，即 $(d_1 = d_2 = \ldots = d_n = k)$： 此时 $d_j = -\lambda$ 对所有 j 成立，拉格朗日函数的偏导条件满足。代入目标函数为式10：

$$f(x) = \sum_{j=1}^{n} x_j \cdot k = k \cdot \sum_{j=1}^{n} x_j = k \cdot 1 = k \tag{10}$$

即 $f(x)$ 为常数，所有满足约束 $\sum x_j = 1$ 的 x 都是最优解。

2. 若 d_j 不相等，也是我们要讨论的情况： $d_j = -\lambda$ 无法对所有 j 成立，说明极值不在可行域内部，而在边界上。 可行域为 $\sum x_j = 1$，此外还有 $x_j \geq 0$，则可行域是 n 维单纯形，其顶点为 “仅一个 $x_j = 1$，其余 $x_k = 0\,(k \neq j)$” 的向量。

线性函数在单纯形上的极值必在顶点处取得：

1) 最大化 $f(x)$：选择 d_j 最大的索引 j^*，令 $x_{j^*} = 1$，其余 $x_k = 0$，此时 $f(x) = d_{j^*}$，这是最大值；

2) 最小化 $f(x)$：选择 d_j 最小的索引 j'，令 $x_{j'} = 1$，其余 $x_k = 0$，此时 $f(x) = d_{j'}$，这无疑是最小值。

有了理论我们立即使用，我们把它实现在约束上：

通过梯度下降优化 $f(x)$，需在迭代中保持约束 $\sum x_j = 1$，我们选择使用投影梯度法：

首当其冲的是无约束梯度更新：

根据梯度 $\nabla f(x) = d$，更新公式显然为：

1. 最小化 $f(x)$： $x_j^{(t+1)} = x_j^{(t)} - \alpha \cdot d_j$，这是沿梯度反方向；

2. 最大化 $f(x)$： $x_j^{(t+1)} = x_j^{(t)} + \alpha \cdot d_j$，这是沿梯度方向；

其中 α 为学习率， t 为迭代次数。虽然我们认为没有人会最小化自己的收益。

加下来投影到可行域。

由于更新后可能破坏约束 $\sum x_j = 1$，需投影回可行域： $\widehat{x}_j^{(t+1)} = \frac{x_j^{(t+1)}}{\sum_{k=1}^{n} x_k^{(t+1)}}$,当然若有非负约束，还需截断负值： $\widehat{x}_j^{(t+1)} = max\left(0, \widehat{x}_j^{t+1}\right)$，再重新归一化。

如果你需要解决诸如：“连锁店博弈” [6]、“ $Stackelberg$ 模型” [7]，则你可以在梯度下降的途中动态更改价值矩阵以达到接近动态博弈中 “收益矩阵随阶段更新” 的情况。从其梯度计算逻辑来看，更改价值矩阵后，梯度会基于新的收益矩阵重新计算，因为梯度依赖于收益矩阵中的元素 $f_{a,i,j}$ 或 $f_{b,i,j}$，策略概率向量也会随之调整以适应新的收益结构。当然，这种调整能够让模型适配收益变化的场景，从而突破静态博弈中 “收益固定” 的限制。当然，这种方法并未从根本上 “解决” 静态博弈本身，而是将其转化为另一种博弈类型进行分析。如何从根本解决这个问题也是后人的研究方向之一。如果你需要更为“不理性”的选择，尝试再计算中途停下。

此外，不难注意到如果我们用计算出的表格选择下一个博弈时，这种状态可以被描述成一棵树。举例来说假设我们在进行了一次“囚徒困境”后有进行了一次“囚徒困境”，使用一张图描述就是图1：

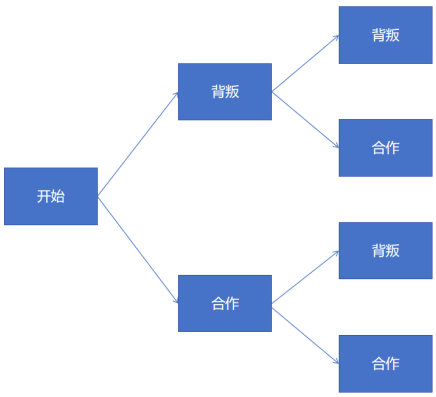

图1 两次囚徒困境博弈的状态树示意图

此外，仔细观察，还可以发现所有的边都是单向边，有一个确定的根，这点很好解释，我们不可能在事件发生后改变它的历史，至少目前还不行；这棵树每一个节点是否可以到达，只取决于其父节点的选择，这无疑不是在提示我们使用动态规划。

形式地说，假设记其父结点被选中的概率为 $\mathrm{P(father)}$，父节点选中它的概率为 $\mathrm{P(father->it)}$，中符号“ $->$ ”为父节点到它的过程。则显然，是现有父结点被选中，后有父节点选中它，它们之间有明显的先后顺序，则可描述有：

$$P(it) = P(father) * P(father -> it)$$

并规定P(开始)=1。

为了下文方便书写，我们定义 $child(node, i)$ 表示 $node$ 号节点的第 i 号子节点的编号。定义 $near_child(node, i)$ 表示 node 号节点到第 i 号最短路中 $node$ 号节点所过 $node$ 号节点的子节点的编号。定义 $shortestPath\,(start, end)$ 表示由 $start$ 号节点到 end 号节点的编号所组成的有序集合。

继续，假定给定两点 $start$, end ，则从 $strart$ 到 end 的路径被选中的概率为式11：

$$\begin{aligned} &P\,(start, end) = P\,(start) \times \\ &\prod_{i=0}^{k-1} P\,(shortestPath\,(start, end)_i \rightarrow \mathrm{near_child}\,(shortestPath\,(start, end)_i, end)) \end{aligned} \tag{11}$$

显然，这个模型与马尔科夫链的无记忆，很接近，但是现实生活中，有记忆的情况更为常见，所以接下来继续讨论如何给这个模型加上记忆化。

我们将记忆抽象成一根向量 n 维向量，并记为如式12的形式：

$$\mathrm{memory} = \begin{pmatrix} m_1 \\ \vdots \\ m_n \end{pmatrix} \tag{12}$$

此时，对于每一次博弈，我们使用 memory 作用于 c 上，当然，由于 memory 的维数可能不等于 c 的维数，所以改为式13的样子：

$$c_i = c_i + \sum_{j=1}^{n} \mathrm{q_j} * \mathrm{memory_j} \tag{13}$$

中 n 为 memory 的维数。

注意到 memory 也是梯度下降的一部分，碍于篇幅显示，我们将在下一个文章中给出它的更为具体的方法。不过，显然注意到在 memory 的计算中我们需要更为注意对手的选择，具体说，我们需要找到一种递归计算的方法使得它可以模拟两人的选择，在这个基础上进行优化。

3. 结果

让我们举出一个以囚徒困境为例子，进行计算。
我们首先定义 $player_a$ 价值矩阵 c 为式14：

$$c = \begin{bmatrix} -1 & -10 \\ 0 & -5 \end{bmatrix} \tag{14}$$

同理，因为 $player_b$ 面对的情况与 $player_a$ 一样，所以有与式14一样的式15：

$$\begin{bmatrix} -1 & -10 \\ 0 & -5 \end{bmatrix} \tag{15}$$

然后，为了避免手动计算，我们采用一个 python 脚本进行计算，过程如图（图2），为了方便，这里一次计算 100 次：

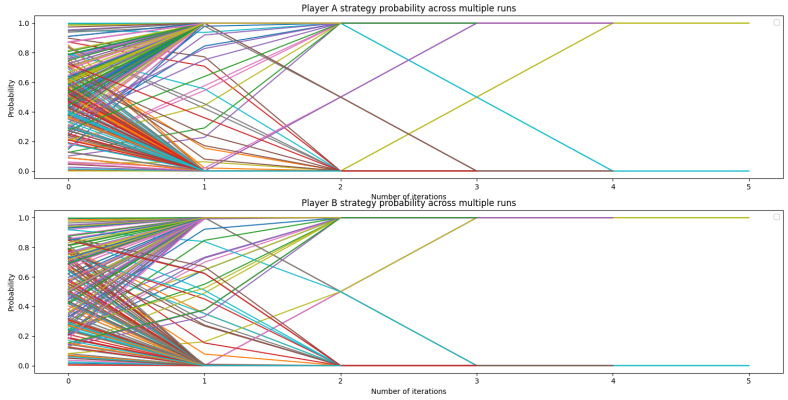

图2 计算100次的结果

不难看出：
虽然我们初始化时为随机初始化，但是最终都达到了式16

$$\begin{bmatrix} 1. & 0. \\ 1. & 0. \end{bmatrix} \tag{16}$$

的选择权重。当然，必须指出，由于梯度下降可能会陷入局部最优值而无法自拔，所以上述的图是否完全一样，作者不做保证。不过我们任然建议你多运行几次。
让我们总结一下，我们计算了价值矩阵，紧接着计算每位玩家的收益，使用梯度下降进行迭代。
对于进一步研究，其他研究者可以尝试解决梯度下降可能会陷入局部最优值而无法自拔的问题。常用的优化算法我们一列了一个表方便使用（表 4）：

表4 优化梯度下降速度和正确性的算法举例

优化方法	优点	缺点
批量梯度下降（BGD）[8]	1. 每次迭代使用全量数据计算梯度，得到的梯度方向更准确，收敛稳定便于并行计算2. 理论上能收敛到全局最优解	1. 计算成本高，尤其在数据量大时，每次迭代耗时久2. 内存占用大，需存储全部数据3. 无法在线更新模型，不适应实时数据流
随机梯度下降（SGD）[9]	1. 每次迭代仅用单个样本计算梯度，计算速度快，耗时短2. 内存占用小，适合大数据集3. 具有随机性，可能跳出局部最优，找到更好的解4. 支持在线学习，能实时更新模型	1. 梯度方向不稳定，收敛过程波动大，容易震荡2. 收敛到最优解的过程较慢，可能在最优解附近徘徊3. 学习率选择敏感，不合适的学习率会影响收敛效果
小批量梯度下降（MBGD）[10]	1. 结合了 BGD 和 SGD 的优点，既保证了梯度的稳定性，又提高了计算效率2. 可利用矩阵运算进行并行计算，兼顾速度和准确性3. 收敛过程比 SGD 更稳定，比 BGD 更高效	1. 批量大小的选择需要调参，不同任务可能需要不同的最佳批量大小2. 仍存在一定的收敛波动，虽小于 SGD，但大于 BGD3. 当批量较小时，仍可能出现局部最优问题
动量法（Momentum）[11]	1. 引入动量项，模拟物理中的惯性，加速收敛，减少震荡2. 对于高曲率、小但一致的梯度方向，能更快地收敛3. 有助于跳过局部最优的浅谷	1. 动量参数需要调优，选择不当可能导致过冲或收敛缓慢2. 在梯度方向变化剧烈的场景，可能保留过多旧梯度的影响
自适应梯度算法（AdaGrad）[12]	1. 自适应调整学习率，对稀疏特征或出现频率低的特征分配较大学习率，对频繁出现的特征分配较小学习率，适合处理稀疏数据2. 无需手动调整学习率	1. 学习率随迭代次数增加而单调递减，最终可能趋近于 0，导致模型后期无法更新2. 对全局学习率初始值较敏感
自适应动量估计（Adam）[13]	1. 结合了动量法和 RMSprop 的优点，同时考虑梯度的一阶矩（动量）和二阶矩（自适应学习率）2. 收敛速度快，稳定性好，对不同任务的适应性强3. 学习率具有自适应性，无需大量调参4. 内存占用合理，计算效率高	1. 在某些场景下，可能收敛到次优解而非全局最优解2. 对于极不平衡的数据，效果可能不如专门针对稀疏数据的算法（如 AdaGrad）3. 动量相关参数（β1、β2）的选择可能影响收敛效果，虽有默认值，但在特定任务中仍需微调
RMSprop [14]	1. 解决了 AdaGrad 学习率单调递减的问题，通过指数移动平均调整学习率，使学习率在迭代后期仍能保持一定的更新能力2. 对非平稳目标函数（如周期性数据）效果较好3. 收敛过程较稳定	1. 仍需要手动设置全局学习率2. 对于某些任务，可能需要与动量法结合才能达到更好的效果

你也可以试试解决这个算法在处理结果为混合纳什均衡时，它会出现难以收敛，这也是梯度下降的常见问题之一，见图 3:

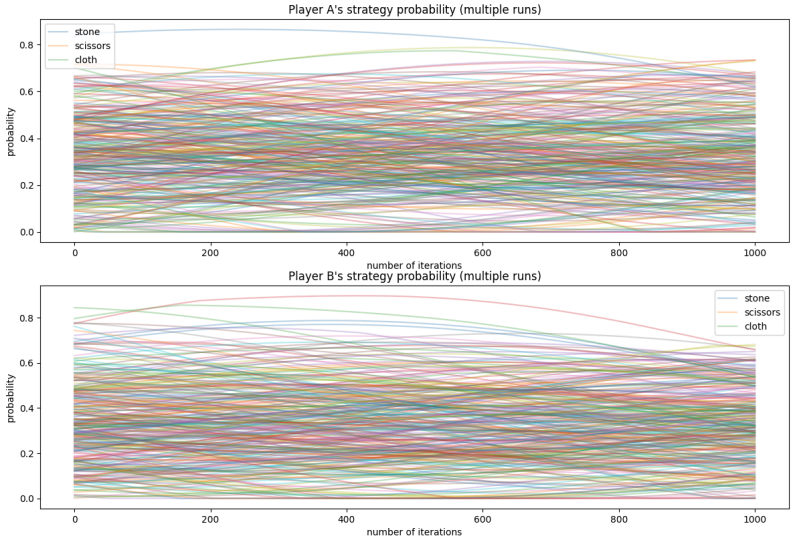

图3 在处理混合纳什均衡中出现的问题

同样，计算 100 次，发现最终处于纳什均衡的个数为 5 个，这说明我们的梯度下降有点下降过头了。附带给出我们的参数（表 5）：

表5 我们用于计算的关键参数

参数名称	值	解释
Learning rate	0.001	学习率
Max iter	1000	学习论数
threshold	0.1	允许的误差范围
Num runs	100	反复运行的次数

不过作者在这里尽力使用统计学的方法，分析了了计算混合纳什均衡的结果。首先是结果的分布。作者经 1000次运行，统计处理图 4的结果。不难看出，

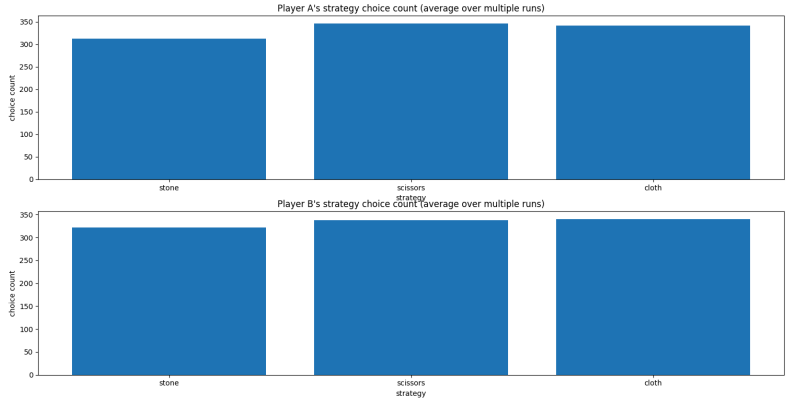

图4 计算结果的选择分布

总体结果分布较为均匀。不过让我们换个视角，看看每个这 1000 次计算结果相比于标准结果的距离。这里为了更好的演示这个差值而不是被正负数所抵消，选用了较为常用的取平方的做法。结果如图 4.4所示。这里的横轴取 (0,1).

为了验证算法收敛，再次调整Max iter为 10000，效果见图 5的上下变化。

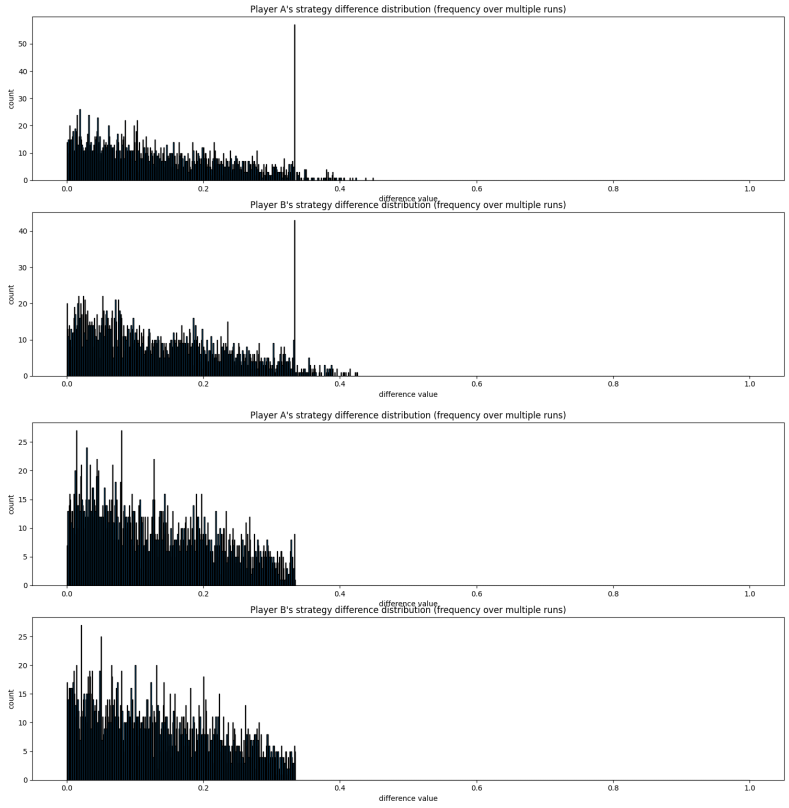

图5 迭代次数（Max iter）调成10000后的结果分布

显然，所有的结果更为靠左，或者说更为靠近正确的值了。
对于这一算法的理论描述就此结束。

参考文献

[1] D. WANG, X. YAN, Z. DOU, W. HUANG, Y. YANG, and X. DENG, “Approximating N-Player Nash Equilibrium through Gradient Descent, ” arXiv (Cornell University), Jan. 2025, doi: https: //doi.org/10.48550/arxiv.2501.03001.

[2] D. LI, M. YE, L. DING, and S. XU, “Distributed Nash Equilibrium Computation With Uncertain Dynamics and Disturbances, ” IEEE Transactions on Network Science and Engineering, vol. 9, no. 3, pp. 1376–1385, Jan. 2022, doi: https: //doi.org/10.1109/tnse.2022.3142523.

[3] F. S. KHAN, O. OKRUT, K. CANNON, K. H. EL-SAFTY, and N. ELSOKKARY, “Calculating Nash equilibrium on quantum annealers, ” Annals of Operations Research, Jan. 2024, doi: https: //doi.org/10.1007/s10479-023-05700-z.

[4] ZHANG GUOQIANG, ZHAO GUODANG. An Improved Differential Evolution Algorithm for Solving Nash Equilibrium Problem and Generalized Nash Equilibrium Problem [J]. Operations Research and Management Science, 2023, 32(3): 36-42.

[5] D. ZHANG, L. JI, S. ZHAO, and L. WANG, “Variable-sample method for the computation of stochastic Nash equilibrium, ” IISE Transactions, vol. 55, no. 12, pp. 1217–1229, Feb. 2023, doi: https: //doi.org/10.1080/24725854.2022.2163436.

[6] R. SELTEN, “The chain store paradox, ” Theory and Decision, vol. 9, no. 2, pp. 127–159, Apr. 1978, doi: https: //doi.org/10.1007/bf00131770.

[7] T. LI and S. P. SETHI, “A review of dynamic Stackelberg game models, ” Discrete & Continuous Dynamical Systems - B, vol. 22, no. 1, pp. 125–159, 2017, doi: https: //doi.org/10.3934/dcdsb.2017007.

[8] S. RUDER, “An overview of gradient descent optimization algorithms, ” arXiv.org, Jun. 15, 2017. https: //arxiv.org/abs/1609.04747?ref=ruder.io

[9] N. KETKAR, “Stochastic Gradient Descent, ” Deep Learning with Python, pp. 113–132, 2017, doi: https: //doi.org/10.1007/978-1-4842-2766-4_8.

[10] S. KHIRIRAT, H. R. FEYZMAHDAVIAN, and M. JOHANSSON, “Mini-batch gradient descent: Faster convergence under data sparsity, ” IEEE Xplore, Dec. 01, 2017. https: //ieeexplore.ieee.org/document/8264077

[11] Y. LIU, Y. GAO, and W. YIN, “An Improved Analysis of Stochastic Gradient Descent with Momentum, ” Jul. 2020, doi: https: //doi.org/10.48550/arxiv.2007.07989.

[12] R. WARD, X. WU, and L. BOTTOU, “AdaGrad stepsizes: Sharp convergence over nonconvex landscapes, ” arXiv (Cornell University), Jan. 2018, doi: https: //doi.org/10.48550/arxiv.1806.01811.

[13] D. KINGMA and J. BA, “Adam: A Method for Stochastic Optimization, ” Computer Science, 2014, doi: https: //doi.org/10.48550/arXiv.1412.6980.

[14] N. SHI and D. LI, “RMSPROP CONVERGES WITH PROPER HYPERPARAMETER, ” international conference on learning representation, Apr. 2021, Accessed: Jul. 02, 2024. [Online]. Available: https: //par.nsf.gov/biblio/10273075