Bargaining Game

61518218 沈书杨

报告的代码部分均由 Python 实现

1. 符号及限制：

i 玩家 i

j 玩家 j

n 最终博弈的轮数，即在 n 次博弈后本次博弈结束

t 当前博弈的轮数，满足$0 < t < n, t \in Z$

σ_i 玩家 i 的衰减率，满足$0 < \sigma_i \leq 1$

σ_j 玩家 j 的衰减率，满足$0 < \sigma_j \leq 1$

r_i^t 在第 t 轮博弈时 i 分到的比率

r_j^t 在第 t 轮博弈时 j 分到的比率

v_i^t 在第 t 轮博弈时 i 获得的收益

v_j^t 在第 t 轮博弈时 j 获得的收益

$s_t = (r_i^t, r_j^t)$ 在第 t 轮博弈时玩家做出的决策

限制：

$$r_i^t + r_j^t = 1$$

2. 问题描述：

题目：

由 i，j 两名玩家分一块价值为 1 的蛋糕。博弈由 i 开始，由一人提出方案，另一人选择是否通过。如果通过则按照提出的方案进行分割；如果不同意则交换，又第二人提出方案，另一人决定是否通过。博弈一直进行 n 轮结束。若最后所有方案均为通过，则分割作废，两名玩家的收益均为 0。蛋糕对于每个玩家的价值会在每轮衰减，对于 i, j 其衰减系数分别为 σ_i 和 σ_j。

(1). 当玩家 i 与玩家 j 的衰减率相同时，即 $\sigma_i = \sigma_j = \sigma$ 时，并且是常识。分别设计玩家 i 和玩家 j 的策略。

(2). 当玩家 i 与玩家 j 的衰减率不同时，即 $\sigma_i \neq \sigma_j$ 时，并且是常识。分别设计玩家 i 和玩家 j 的策略。

(3). 当玩家 i 与玩家 j 的衰减率不同，且每个玩家只知道自己的折扣因子，不知道对方的折扣因子。分别设计玩家 i 和玩家 j 的策略。

3. 问题分析：

本博弈在考虑**所有玩家都不会做出损人不利己的决策**的情况下，运用逆向归纳法求解。在博弈的过程中，由于当博弈到达 n 轮时，无论此时由哪个玩家进行分配，它们都

有一个绝对占优决策——都会给自己分配 1 的蛋糕比例而给对手 0 以获得最大收益(如果游戏能进行到最后一轮)。但此时玩家 1 为自己分配了所有的收益，而为玩家 2 获得的收益为 0, 此时需要玩家 2 在 n-1 轮做出能让玩家 1 接受且能让自己获得更多收益的决策。如果同意了该分配方案，则玩家 2 的收益明显会大于 0。于是采用逆向归纳法，则在第 n-1 轮，由玩家 2 进行分配时，考虑到第 n 轮的情况，则分配给玩家 1 的比例至少需要在满足玩家 1 在最后一轮获得的收益，并把剩余收益归为己有。以此类推，在每一轮博弈时, 每个玩家都可以确定自己在每一轮提出的决策能被对方接受且使自己的收益最大化。

4. 求解过程：

(1) $\sigma_i = \sigma_j = \sigma$，常识

a. 简单情况下的规律分析

先考虑简单的情况，即 n=1，此时易得 $s_1 = (1,0)$。

再考虑 n=2 时，此时易得 $s_2 = (0,1)$，而 $v_j^2 = 1 \times \sigma = \sigma$。为此，i 需要调整自己在第一轮做出的决策，即，使 j 在第一轮获得的利益不少于在第二轮获得的利益：

$$r_j^1 \times 1 \geq \sigma$$

所以此时$s_1 = (1-\sigma, \sigma)$。

紧接着考虑 n=3 时，此时易得 $s_3 = (1,0)$，而 $v_i^3 = 1 \times \sigma^2 = \sigma^2$。为此，j 需要调整自己在第二轮做出的决策, 即, 使 i 在第二轮获得的利益不少于在第三轮获得的利益：

$$r_i^2 \times \sigma \geq \sigma^2$$

所以此时$s_2 = (\sigma, 1-\sigma)$。同理，i 也需要相应的在第一轮改变自己的策略，即，使 j 在第一轮获得的利益不少于在第二轮获得的利益：

$$r_j^1 \times 1 \geq (1-\sigma)\sigma$$

所以此时$s_1 = (1-(1-\sigma)\sigma, (1-\sigma)\sigma)$。

b. 规律总结及推广(数学推导)

首先，不论 n 的奇偶性，当 T=n 时，$s_n = \left(\frac{1-(-1)^n}{2}, \frac{1+(-1)^n}{2}\right)$

当 T=t 时，$r_p^t = \sigma r_p^{t+1}$,

$$p = \begin{cases} i & if\ 1+(-1)^{t+1} = 0 \\ j & otherwise \end{cases}$$

此时，$r_q^t = 1 - r_p^t$

$$q = \begin{cases} i & if\ p = j \\ j & otherwise \end{cases}$$

这样就可以得出 T=t 时的s_t：

$$s_t = \begin{cases} (r_p^t, r_q^t) & if\ p = i \\ (r_q^t, r_p^t) & otherwise \end{cases}$$

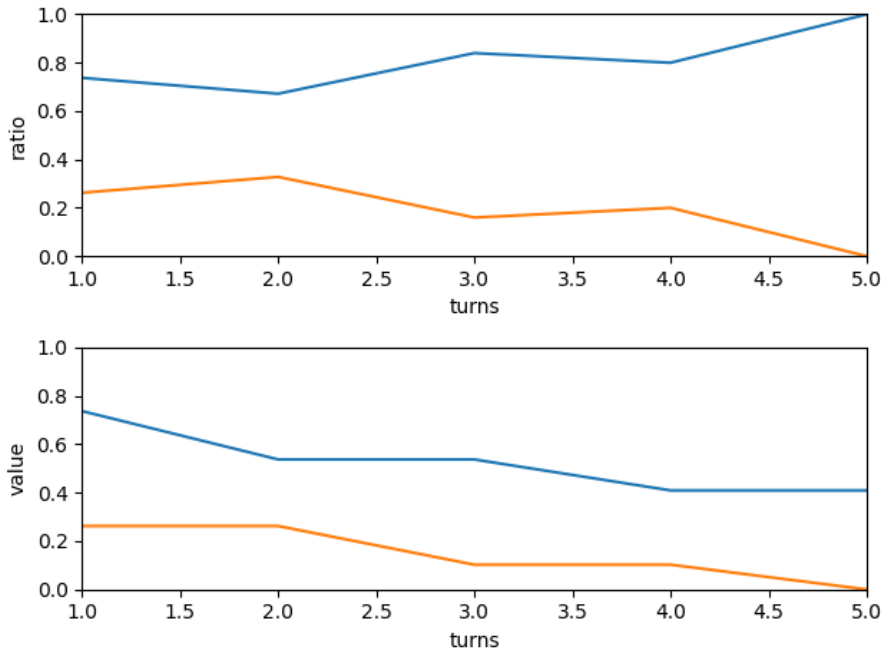
至此，我们完成了玩家 i 和玩家 j 在各轮游戏最优策略的制定。

c. 代码实现及图表展示

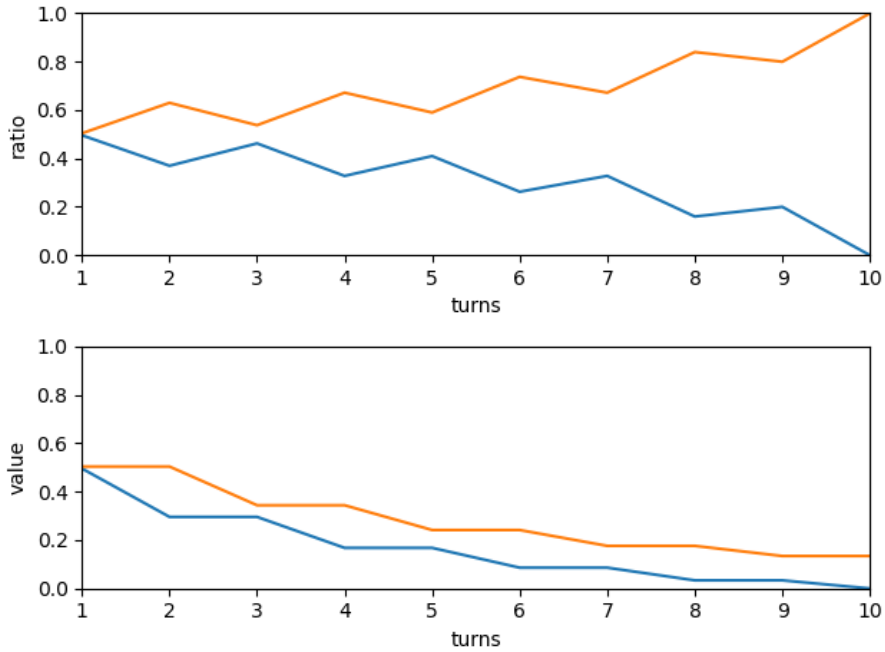
示意代码：

```
import math
import numpy as np
import matplotlib.pyplot as plt

n=5
sigma=0.8
s=[]
v=[]

if (1+math.pow(-1,n))/2==0:
    s.append([1,0])
    v.append([1*math.pow(sigma,n-1),0])
else:
    s.append([0,1])
    v.append([0,1 * math.pow(sigma, n - 1)])
for i in range(1,n):
    if (1+math.pow(-1,n-i+1))/2==0:
        temp_s=[sigma*s[-1-i+1][0],1-sigma*s[-1-i+1][0]]
        s.insert(0,temp_s)
        temp_v=(np.array(temp_s)*math.pow(sigma, n - 1-i)).tolist()
        v.insert(0,temp_v)
    else:
        temp_s=[1-sigma*s[-1-i+1][1],sigma*s[-1-i+1][1]]
        s.insert(0,temp_s)
        temp_v=(np.array(temp_s)*math.pow(sigma, n - 1-i)).tolist()
        v.insert(0,temp_v)
```

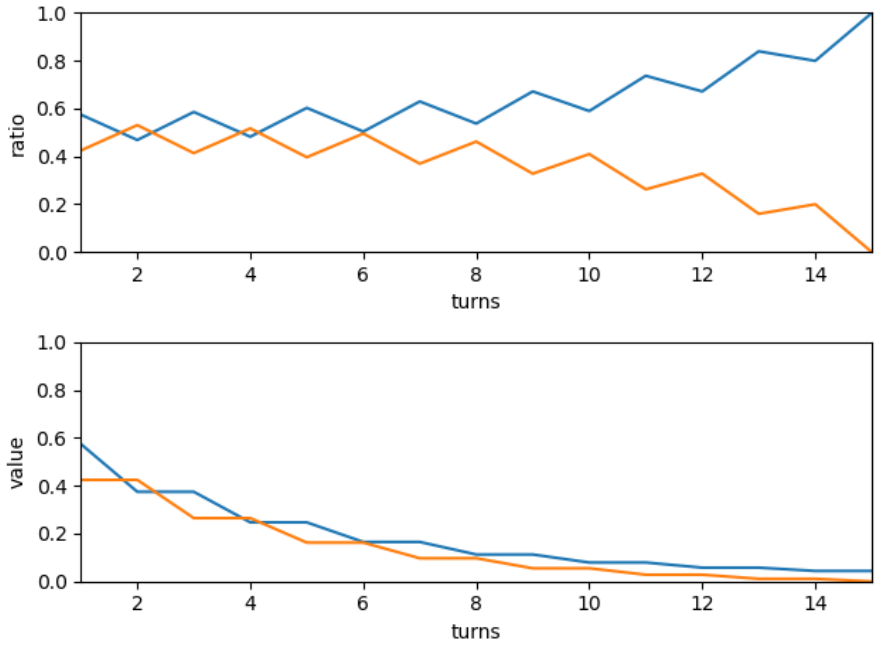
下面展示 n=5，$\sigma = 0.8$时的决策图及收益图（蓝线为 i，橙线为 j）：



下面展示 n=10，$\sigma = 0.8$时的决策图及收益图（蓝线为 i，橙线为 j）：



下面展示 n=15，$\sigma = 0.8$时的决策图及收益图（蓝线为 i，橙线为 j）：



需要注意的是，由于**计算机计算精度的丢失**，**有些数据会不符常识**，但可以**正确地描述变化趋势**（如在 python 中，1-0.8=0.1999···996）。上述展示的图像能较好反应数学证明过程中 i，j 的收益比例及收益价值的变化特点，也证明了数学推导的正确性。

d. 总结

在本问题中，我同时考虑了 n 为奇数和偶数的情况，代码具有可适应性，计算机表示精度带来的影响可以忽略不计。

(2) $\sigma_i \neq \sigma_j$，常识

a. 简单情况下的规律分析

先考虑简单的情况，即 n=1，此时易得 $s_1 = (1,0)$。

再考虑 n=2 时，此时易得 $s_2 = (0,1)$，而 $v_j^2 = 1 \times \sigma_j = \sigma_j$。为此，i 需要调整自己在第一轮做出的决策，即，使 j 在第一轮获得的利益不少于在第二轮获得的利益：

$$r_j^1 \times 1 \geq \sigma_j$$

所以此时$s_1 = (1 - \sigma_j, \sigma_j)$。

紧接着考虑 n=3 时，此时易得 $s_3 = (1,0)$，而 $v_i^3 = 1 \times \sigma_i^2 = \sigma_i^2$。为此，j 需要调整自己在第二轮做出的决策，即，使 i 在第二轮获得的利益不少于在第三轮获得的利益：

$$r_i^2 \times \sigma_i \geq \sigma_i^2$$

所以此时$s_2 = (\sigma_i, 1 - \sigma_i)$。同理，i 也需要相应的在第一轮改变自己的策略，即，使 j 在第一轮获得的利益不少于在第二轮获得的利益：

$$r_j^1 \times 1 \geq (1 - \sigma_i)\sigma_j$$

所以此时$s_1 = (1 - (1 - \sigma_i)\sigma_j, (1 - \sigma_i)\sigma_j)$。

b. 规律总结及推广(数学推导)

首先，不论 n 的奇偶性，和问题**(1)**一样，当 T=n 时，$s_n = \left(\frac{1-(-1)^n}{2}, \frac{1+(-1)^n}{2}\right)$

当 T=t 时，$r_p^t = \sigma_p r_p^{t+1}$，

$$p = \begin{cases} i & if\ 1 + (-1)^{t+1} = 0 \\ j & otherwise \end{cases}$$

此时，$r_q^t = 1 - r_p^t$

$$q = \begin{cases} i & if\ p = j \\ j & otherwise \end{cases}$$

这样就可以得出 T=t 时的s_t：

$$s_t = \begin{cases} (r_p^t, r_q^t) & if\ p = i \\ (r_q^t, r_p^t) & otherwise \end{cases}$$

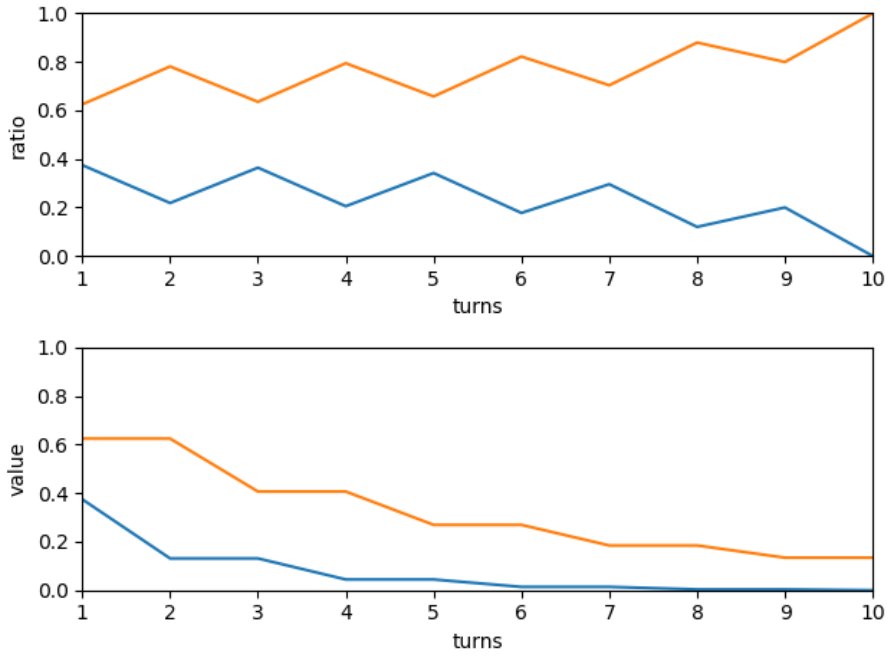
至此，我们完成了玩家 i 和玩家 j 在各轮游戏最优策略的制定。

c. 代码实现及图表展示

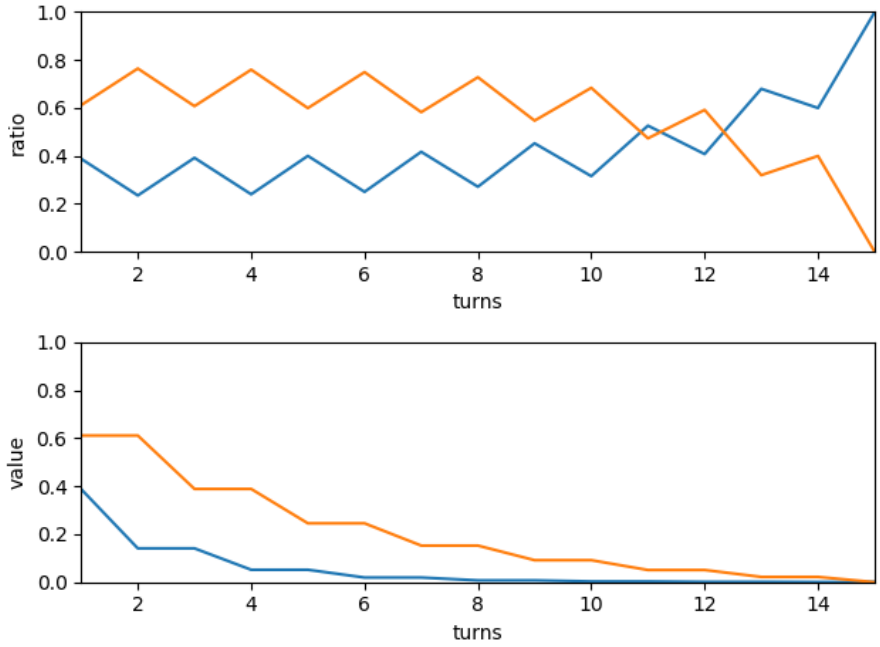
示意代码：

```python
import math
import numpy as np
import matplotlib.pyplot as plt

n=15
sigma_i=0.8
sigma_j=0.8
s=[]
v=[]

if (1+math.pow(-1,n))/2==0:
    s.append([1,0])
    v.append([math.pow(sigma_i,n-1),0])
else:
    s.append([0,1])
    v.append([0,math.pow(sigma_j, n - 1)])
for i in range(1,n):
    if (1+math.pow(-1,n-i+1))/2==0:
        temp_s=[sigma_i*s[-1-i+1][0],1-sigma_i*s[-1-i+1][0]]
        s.insert(0,temp_s)
        temp_v=(np.array(temp_s)*np.array([sigma_i,sigma_j])**(n - 1-i)).tolist()
        v.insert(0,temp_v)
    else:
        temp_s=[1-sigma_j*s[-1-i+1][1],sigma_j*s[-1-i+1][1]]
        s.insert(0,temp_s)
        temp_v=(np.array(temp_s)*np.array([sigma_i,sigma_j])**(n - 1-i)).tolist()
        v.insert(0,temp_v)
```

下面展示 n=10，$\sigma_i = 0.6$，$\sigma_j = 0.8$时的决策图及收益图（蓝线为 i，橙线为 j）：



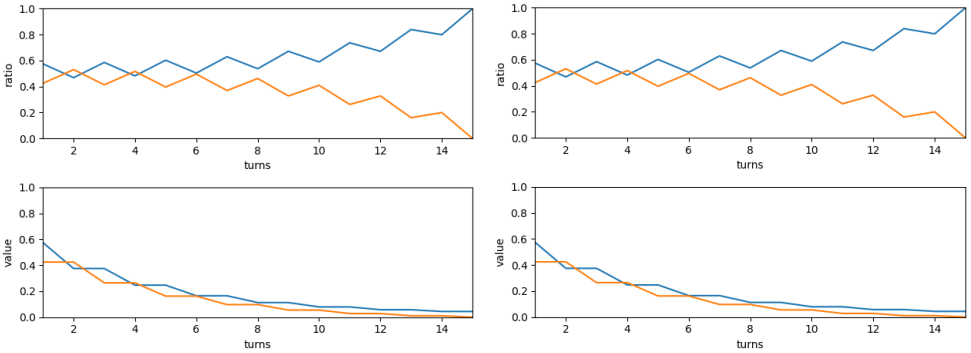
下面展示 n=15，$\sigma_i = 0.6$，$\sigma_j = 0.8$时的决策图及收益图（蓝线为 i，橙线为 j）：



需要注意的是，由于**计算机计算精度的丢失**，**有些数据会不符常识**，但可以**正确地描述变化趋势**（如在 python 中，1-0.8=0.1999···996）。上述展示的图像能较好反应数学证明过程中 i，j 的收益比例及收益价值的变化特点，也证明了数学推导的正确性。

d. 对比

下列左图为使用问题**(2)**的代码在问题**(1)**的条件下画的图，右图为问题**(1)**中代码画的图：



可以看出，两者的结果具有一致性，是对问题**(1)(2)**的数学推导的确认。

e. 总结

和问题**(1)**一样，我同时考虑了 n 为奇数和偶数的情况，代码具有可适应性，计算机表示精度带来的影响可以忽略不计。

同时，通过观察总结可以发现，在发现了迭代规律以后，问题(1)和问题(2)之间并没有太大的差别，可以说只是有了一些参数数值的变化

(3) $\sigma_i \neq \sigma_j$，非常识

a. 符号添加

σ_{ij} 玩家 i 对玩家 j 衰减率的猜测，满足$0 < \sigma_{ij} \leq 1$

σ_{ji} 玩家 j 对玩家 i 衰减率的猜测，满足$0 < \sigma_{ji} \leq 1$

b. 基本推理分析

当 n=1 时，易得此时第一轮的决策同问题**(1)(2)**一样，是 $s_1 = (1,0)$。

但是当 n=2 时情况变化

考虑玩家 i 的视角。对于 i 来说，若 i 在第一轮不能吃亏，i 就要考虑所有可能的σ_{ij}，当$\sigma_{ij} \to 0$时，$s_1 = (1,0)$是 i 的最优方案。但是，只要$\sigma_j > \sigma_{ij}$，则 j 不接受 i 的方案，i 的最终受益为 0，这显然是不可接受的。

在这样的情况下当 n=3 时也是一样，并且可以推广。若每个玩家都把极端情况考虑在内且不肯做出让步，每回合做出的决策必然是$0-1$决策，而这样的决策是没有意义的。

c. 基本推理分析 II

当 n=1 时，易得此时第一轮的决策同问题**(1)(2)**一样，是 $s_1 = (1,0)$。

但是当 n > =2 时情况变化，考虑双方玩家会虚张声势：

考虑玩家 i 的视角。对于 i 来说，若σ_i很小，可以通过假装自己的衰减系数很大来为自己争取更多利益。由于两方玩家都是同样理性的，这种做法只有在σj同样很小而且虚张声势的期望收益要大于理性决策的收益时才适用。

d. 提出假设

综上本题 **b**，**c** 部分分析，为了简化问题，我们假设双方玩家都会做出让步，

而且不会进行虚张声势，单纯理性进行决策。

同时，玩家可以认为衰减系数都是服从同一分布的，本题假设分布为正态分布。

e. 推理分析

由问题**(2)**的结论和本题假设的变量σ_{ij}和σ_{ji}可以迭代的得到在双盲情况下，玩家 i 和玩家 j 每回合的最优决策。并且玩家 i（j）可以根据每回合对手的接受/拒绝进行动态的σ_{ij}（σ_{ji}）参数调整。只要进行到某一回合中，有一个玩家拒绝对手的决策在接下来回合所带来的收益不可能超过接受决策带来的收益时，游戏结束，可以认为每个玩家都获得了相对最优的收益。

本题主要问题转化为参数σ_{ij}和σ_{ji}初始值的设置和玩家 i（j）在每回合对σ_{ij}（σ_{ji}）的调整幅度的设置。（考虑到两个玩家应该具有相同的思路，初始值设置思路应该相等，衰减系数的调整幅度计算公式应当类似）博弈双方不知道对方的衰减系数，这就导致每一轮只有提出决策并被拒绝的玩家会调整自己对对手衰减系数的估计。不具有最后一轮决策权的玩家对另一玩家的衰减系数的估计需要在倒数第二轮尽可能的大，否则最后一轮收益为 0。

f. 推理总结

本题是无法求出定式解的，只能定性的给出解决思路。

假设衰减系数服从正态分布，简单使σ_{ij}和σ_{ji}初始值为 0.3，这个初始值也可以从一种角度表现出每个玩家所能接受的损失。

接下来设计合适的衰减系数的调整幅度计算公式。

g. 公式设计

玩家对于对手的衰减系数的预测始终为$\sigma_{pq}x^{\alpha}+1-\sigma_{pq}$，其中$\sigma_{pq}$为玩家对对手衰减速率的初始预测值，$\alpha \in (0,+\infty)$，这里不对其余的变量进行优化，只考虑$\alpha$对结果的影响。

以上的自变量$x \in [0,1]$，由总轮数 n 和当前博弈轮数 t 决定。

决策的制定依然主要沿用第**(2)**题的方法。

h. 参数设计

同第(2)题中的测试，选取 n=15，$\sigma_i = 0.6$，$\sigma_j = 0.8$作为输入。

对α分别取 0.5 和 2 进行测试

i. 结果展示

代码展示（注释中包含了设计公式的具体解释）：

```
stop_turn=None

truth = strategy(sigma_i, sigma_j, n)[1][0]
for turn in range(n):
    s_i, v_i = strategy(sigma_i, sigma_ij, n)
    s_j, v_j = strategy(sigma_ji, sigma_j, n)

    if (turn+1) % 2 == 1:
        if v_i[turn][1] >= v_j[turn][1]:
            s = [s_i[turn][0], s_i[turn][1]]
            print('停止轮数：',turn+1)
            stop_turn=turn
            break
        else:
            # 当总轮数为奇数时，i为最终决策者，在奇数轮时，若i的方案被拒绝，i提高对j的衰减系数预期，j会提高自己的容忍度以在i下次决策时不至于让自己走进拖延的窘境
            if flag==1:
                #tol_j=tol_j*math.pow()+1-tol_j
                sigma_ij = init_sigma_ij*math.pow(turn/(n+1),alpha)+1-init_sigma_ij

            # 当总轮数为偶数时，j为最终决策者，在奇数轮时，若i的方案被拒绝，i提高对j的衰减系数预期，j会降低自己的容忍度以在i下次决策时让自己尽可能多的获得利益
            else:
                # tol_j=tol_j*(1-math.pow())
                sigma_ij = init_sigma_ij*math.pow(turn/n,alpha)+1-init_sigma_ij


    else:
        if v_j[turn][0] >= v_i[turn][0]:
            s = [s_j[turn][0], s_j[turn][1]]
            print('停止轮数：',turn+1)
            stop_turn=turn
            break
        else:
            # 当总轮数为奇数时，i为最终决策者，在偶数轮时，若j的方案被拒绝，j提高对i的衰减系数预期，i会降低自己的容忍度以在i下次决策时让自己尽可能多的获得利益
            if flag==1:
                #tol_i=tol_i*(1-math.pow())
                sigma_ji = init_sigma_ji*math.pow(turn/n,alpha)+1-init_sigma_ji

            # 当总轮数为偶数时，j为最终决策者，在偶数轮时，若j的方案被拒绝，j提高对i的衰减系数预期，i会提高自己的容忍度以在j下次决策时不至于让自己走进拖延的窘境
            else:
                #tol_i=tol_i*math.pow(turn/n,2)+1-tol_i
                sigma_ji = init_sigma_ji*math.pow(turn/(n+1),alpha)+1-init_sigma_ji

print('理想收益：',truth)
print('实际收益：',np.array(s)*(np.array([sigma_i,sigma_j])**stop_turn))
```

$\alpha = 0.1$，结果：

```
停止轮数：  3
理想收益：  [0.38822811287551995, 0.61177188712448]
实际收益：  [0.18716085 0.3072696 ]
```

从左到右的收益依次是 i，j 的收益。

$\alpha = 0.5$，结果：

```
停止轮数:  4
理想收益:  [0.38822811287551995, 0.61177188712448]
实际收益:  [0.09620602 0.28395611]
```

从左到右的收益依次是 i，j 的收益。

$\alpha = 2$，结果：

```
停止轮数:  4
理想收益:  [0.38822811287551995, 0.61177188712448]
实际收益:  [0.0736104  0.33751608]
```

从左到右的收益依次是 i，j 的收益。

$\alpha = 0.1$的情况下应该是最优的，此时玩家对损失的容忍更强，停止的轮数由 4 变成了 3，收益的总和随之变大，原因应该是在谈判过程中各方对自己可以承受的损失的加快变大了，通俗的来说，就是玩家对敌方的衰减系数的预测更加大胆了。这也等同于，虽然自己的分配比例可能相对与对手来说可能会吃亏，但是从最后获得的总收益来看，自己的收益变高了，所以还是有这么决策的动机的。但是这样的决策的缺点是，可能会在双方的衰减系数近似时获得远低于理想收益的收益，而且如果考虑之前提到的虚张声势的情况下，会使决策出现不小的问题。在这时，也许更大的α，也就是更保守的预测，会对自己更有利。

同时，我们也可以分析得到，更大的容忍度能够显著减少谈判的轮数，从而在**大部分情况下**使双方玩家的收益之和最大，这里就或多或少牵扯到到了一种"less is more"的心理，我会在 **j. 延伸思考**中根据自己查询的资料简要概括。

j. 延伸思考

对于 bargain game，我也查询了相关资料，得到的结论是，在访谈报告中，大部分人只要得到 30%的利益便可以满足。所以在实际生活中，不论我们的"衰减系数"是多少，在双方不知道对方的衰减系数的情况下，在谈判的开始先提出 7-3 开的分配方法也不妨为一种简单有效的试探方法，同时也可能让自己的收益更大。

k. 总结及未来工作

本题作为涉及心理即计算思考和建模的博弈论经典题，具有独特的魅力。我在这里只能粗浅地表达我自己的看法，用一些自己创造的的一些条件来使这个过程能够以代码的方式呈现出一些量化的结果。在这个问题上，我对衰减系数变化公式的设计思路比较单一，也不知道最终的结论能不能作为一个有价值的结果。

在这之后，我可能会尝试使用神经网络对这个模型进行学习，通过黑盒的深度学习框架获得更优的谈判决策，然后对 layer 分析进行理论解释，来更好的理解这个问题并将其应用于我的生活。

我是对本题抱着十足的好奇心进行思考的，也是因此选择了本题作为这门课程的大作业报告内容。如果阅读者对对本问有独到的见解，或者觉得我对这个问题有理解不到位的地方，希望能拨冗对我指教或指正，不胜感激。

5. 结语：

以上为我本次报告的内容，如有疑问或建议，可发至邮箱 213182907@seu.edu.cn 。感谢王万元老师在本学期课程中生动有趣的教学!

6. 代码附录（为了简介，仅附第二问和第三问的完整代码）：

问题(2):

```
import math
import numpy as np
import matplotlib.pyplot as plt

n=15
sigma_i=0.8
sigma_j=0.8
s=[]
v=[]

if (1+math.pow(-1,n))/2==0:
    s.append([1,0])
    v.append([math.pow(sigma_i,n-1),0])
else:
    s.append([0,1])
    v.append([0,math.pow(sigma_j, n - 1)])
for i in range(1,n):
    if (1+math.pow(-1,n-i+1))/2==0:
        temp_s=[sigma_i*s[-1-i+1][0],1-sigma_i*s[-1-i+1][0]]
        s.insert(0,temp_s)
        temp_v=(np.array(temp_s)*np.array([sigma_i,sigma_j])**(n - 1-
i)).tolist()
        v.insert(0,temp_v)
    else:
        temp_s=[1-sigma_j*s[-1-i+1][1],sigma_j*s[-1-i+1][1]]
        s.insert(0,temp_s)
        temp_v=(np.array(temp_s)*np.array([sigma_i,sigma_j])**(n - 1-
i)).tolist()
        v.insert(0,temp_v)

x=range(1,n+1)

plt.subplot(2, 1, 1)
s_i=[s[i][0] for i in range(len(s))]
s_j=[s[i][1] for i in range(len(s))]
plt.plot(x,s_i)
```

```
plt.plot(x,s_j)
plt.xlabel('turns')
plt.ylabel('ratio')
plt.xlim(1,n)
plt.ylim(0,1)

plt.subplot(2, 1, 2)
v_i=[v[i][0] for i in range(len(v))]
v_j=[v[i][1] for i in range(len(v))]
plt.plot(x,v_i)
plt.plot(x,v_j)
plt.xlabel('turns')
plt.ylabel('value')
plt.xlim(1,n)
plt.ylim(0,1)
plt.show()
```

问题(3):

```
import math
import numpy as np

def strategy(sigma_i,sigma_j,n):
    s = []
    v = []

    if (1 + math.pow(-1, n)) / 2 == 0:
        s.append([1, 0])
        v.append([math.pow(sigma_i, n - 1), 0])
    else:
        s.append([0, 1])
        v.append([0, math.pow(sigma_j, n - 1)])
    for i in range(1, n):
        if (1 + math.pow(-1, n - i + 1)) / 2 == 0:
            temp_s = [sigma_i * s[-1 - i + 1][0], 1 - sigma_i * s[-1 - i + 1][0]]
            s.insert(0, temp_s)
            temp_v = (np.array(temp_s) * np.array([sigma_i, sigma_j]) ** (n - 1 -
i)).tolist()
            v.insert(0, temp_v)
        else:
            temp_s = [1 - sigma_j * s[-1 - i + 1][1], sigma_j * s[-1 - i + 1][1]]
            s.insert(0, temp_s)
            temp_v = (np.array(temp_s) * np.array([sigma_i, sigma_j]) ** (n - 1 -
```

```
i)).tolist()
            v.insert(0, temp_v)
    return s,v

n=15

sigma_i=0.6
sigma_j=0.8

init_sigma_ij=0.3
init_sigma_ji=0.3


sigma_ij = 0.3
sigma_ji = 0.3

s = None

flag=n%2

alpha=0.1

stop_turn=None

truth = strategy(sigma_i, sigma_j, n)[1][0]
for turn in range(n):
    s_i, v_i = strategy(sigma_i, sigma_ij, n)
    s_j, v_j = strategy(sigma_ji, sigma_j, n)

    if (turn+1) % 2 == 1:
        if v_i[turn][1] >= v_j[turn][1]:
            s = [s_i[turn][0], s_i[turn][1]]
            print('停止轮数：',turn+1)
            stop_turn=turn
            break
        else:
            # 当总轮数为奇数时，i 为最终决策者，在奇数轮时，若 i 的方案被拒绝，i 提高对 j 的衰减系数预期，j 会提高自己的容忍度以在 i 下次决策时不至于让自己走进拖延的窘境
            if flag==1:
                #tol_j=tol_j*math.pow()+1-tol_j
                sigma_ij = init_sigma_ij*math.pow(turn/(n+1),alpha)+1-init_sigma_ij

            # 当总轮数为偶数时，j 为最终决策者，在奇数轮时，若 i 的方案被拒绝，i 提高对 j 的衰减系数预期，j 会降低自己的容忍度以在 i 下次决策时让自己尽可能多的获得利益
```

```
            else:
                # tol_j=tol_j*(1-math.pow())
                sigma_ij = init_sigma_ij*math.pow(turn/n,alpha)+1-init_sigma_ij


    else:
        if v_j[turn][0] >= v_i[turn][0]:
            s = [s_j[turn][0], s_j[turn][1]]
            print('停止轮数：',turn+1)
            stop_turn=turn
            break
        else:
            # 当总轮数为奇数时，i 为最终决策者，在偶数轮时，若 j 的方案被拒绝，j 提高对 i 的衰减系数预期，i 会降低自己的容忍度以在 i 下次决策时让自己尽可能多的获得利益
            if flag==1:
                #tol_i=tol_i*(1-math.pow())
                sigma_ji = init_sigma_ji*math.pow(turn/n,alpha)+1-init_sigma_ji

            # 当总轮数为偶数时，j 为最终决策者，在偶数轮时，若 j 的方案被拒绝，j 提高对 i 的衰减系数预期，i 会提高自己的容忍度以在 j 下次决策时不至于让自己走进拖延的窘境
            else:
                #tol_i=tol_i*math.pow(turn/n,2)+1-tol_i
                sigma_ji = init_sigma_ji*math.pow(turn/(n+1),alpha)+1-init_sigma_ji

print('理想收益：',truth)
print('实际收益：',np.array(s)*(np.array([sigma_i,sigma_j])**stop_turn))
```