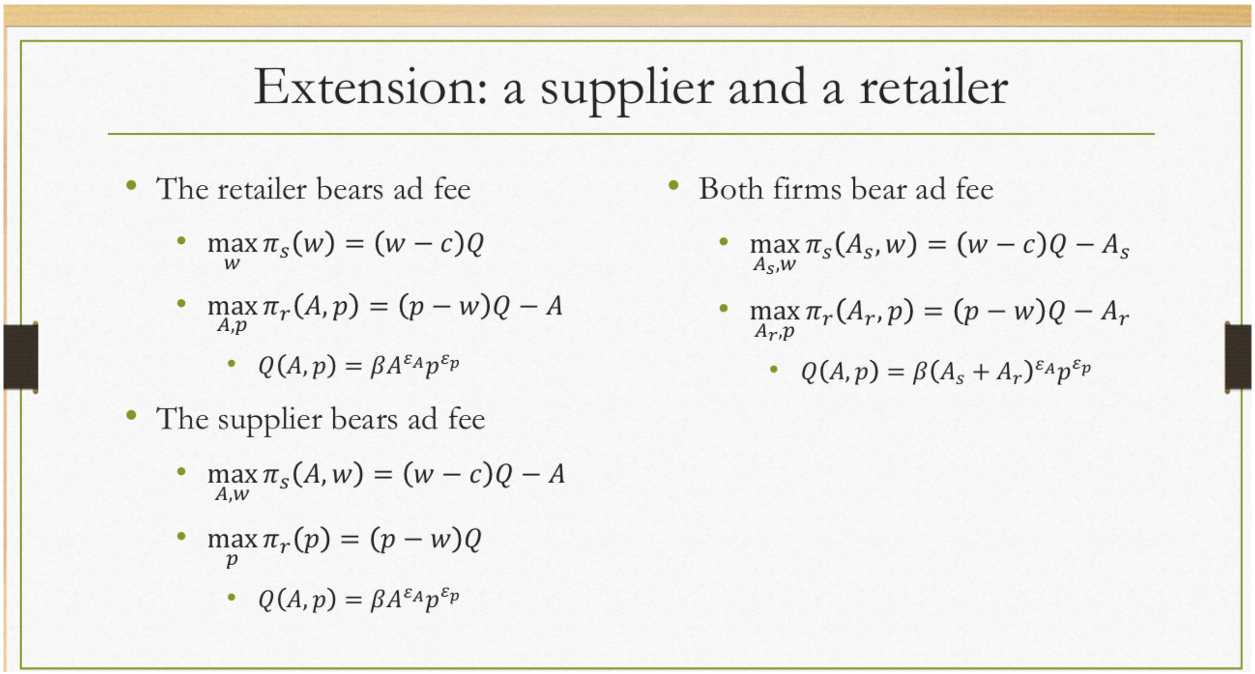
B_ Stackelberg model（MARL）



第一部分：你的理解正確嗎？

是的，你的理解完全正確，而且非常精準。

教授所描述的邏輯，就是標準的 **Stackelberg Equilibrium（史塔克伯格均衡）** 求解過程：

1. **固定 W（供應商先動）：** 假設供應商出了一個價格 w。
2. **檢驗零售商的最佳反應（Retailer Best Response）：** 在這個 w 之下，零售商會選擇什麼樣的 (p, A) 來讓 **他自己的 π_r 最大化**？（注意：零售商是自私的，他不在乎供應商賺多少）。
3. **供應商選擇（回頭看）：** 供應商預知了零售商的上述行為，因此供應商會遍歷所有可能的 w，找出那個「當零售商做出最佳反應後，供應商自己 π_s 最大」的 w。

一、終止條件設計

1. 最大回合數限制： 如果一直沒收斂，最多跑 `n_episodes = 100000` 回合。
2. 提早終止檢查（Early Stopping）：
   - 啟動時間： 訓練超過 `min_episodes = 5000` 後開始檢查。
   - 滑動視窗比較： 比較「最近 1000 筆 (`recent_avg`)」與「前 1000 筆 (`prev_avg`)」的供應商平均利潤。
   - 判定標準： 同時滿足以下三個條件才算穩定：
     1. 利潤變化極小 (`abs(recent_avg - prev_avg) < stability_tolerance`)。
     2. 平均利潤必須是正的 (`recent_avg > 0`)。
     3. 探索率 Epsilon 已經夠低 (`epsilon < 0.1`)。

二、結果與程式碼

B1：

- 批發價 w: 12、售價 p ：30、廣告 A ：1、供應商利潤: 0.6086、零售商利潤: 0.0954
- 會在18000-20000 間收斂

```
import numpy as np
import random
import matplotlib.pyplot as plt
from tqdm import tqdm

# ============================================
# (B.1) Stackelberg MARL: Robust & Stable (With Early Stopping)
# ============================================

# ------------------- 參數設定 -------------------
n_episodes = 100000
gamma = 0.9
```

```
# 學習率
alpha_r = 1.0       # Retailer: 過目不忘
alpha_s = 0.05      # Supplier: 穩定學習

# Epsilon Decay 參數
epsilon_start = 0.5
epsilon_end = 0.01
decay_rate = 0.9999

# ---【新增】收斂判斷參數 ---
min_episodes = 5000        # 至少跑幾輪才開始檢查
convergence_window = 1000   # 取最近 1000 筆的平均
stability_tolerance = 0.005 # 容許誤差 (比舊版嚴格一點，因為這裡利潤數值較小)
stable_counter = 0          # 連續穩定次數計數器
stop_threshold = 20         # 連續 20 次檢查都穩定才停止
history_pi_s = []           # 紀錄歷史利潤
# ---------------------------

beta = 10.0
epsA = 0.5
epsp = -1.5
c = 2

max_w = 15; min_w = c + 1
max_p = 30; min_p = c + 1
max_A = 15; min_A = 1
penalty = -10.0

# ------------------- 動作空間定義 -------------------
possible_w = list(range(min_w, max_w + 1))

possible_retailer_actions = []
for p in range(min_w, max_p + 1):
    for A in range(min_A, max_A + 1):
        possible_retailer_actions.append((p, A))
```

```
n_actions_s = len(possible_w)
n_actions_r = len(possible_retailer_actions)

# Q-Tables
Q_supplier = np.zeros(n_actions_s)
Q_retailer = np.zeros((n_actions_s, n_actions_r))

# ------------------- 函數 -------------------
def get_profits(w, p, A):
    if p <= w: return penalty, penalty
    Q = beta * (A ** epsA) * (p ** epsp)
    pi_s = (w - c) * Q
    pi_r = (p - w) * Q - A
    return pi_s, pi_r

# ------------------- 主訓練 -------------------
best_results = {'w': 0, 'p': 0, 'A': 0, 'pi_s': -np.inf, 'pi_r': -np.inf}

# 初始化 epsilon
epsilon = epsilon_start

print("【B1: Robust MARL Training with Early Stopping】Start...")
pbar = tqdm(range(n_episodes))

for episode in pbar:
    # 1. 更新 Epsilon
    epsilon = max(epsilon_end, epsilon * decay_rate)

    # --- Supplier Move ---
    if random.random() < epsilon:
        idx_w = random.randint(0, n_actions_s - 1)
    else:
        idx_w = np.argmax(Q_supplier)
    w = possible_w[idx_w]
```

```
# --- Retailer Learning Phase (Thinking Loop) ---
n_thinking_steps = 1000

# 保持記憶
current_best_idx = np.argmax(Q_retailer[idx_w])
p_best, A_best = possible_retailer_actions[current_best_idx]
_, pi_best_check = get_profits(w, p_best, A_best)
if pi_best_check > 0:
    Q_retailer[idx_w, current_best_idx] = pi_best_check

for _ in range(n_thinking_steps):
    rand_idx = random.randint(0, n_actions_r - 1)
    p_try, A_try = possible_retailer_actions[rand_idx]
    _, pi_r_try = get_profits(w, p_try, A_try)

    if pi_r_try > 0:
        Q_retailer[idx_w, rand_idx] = pi_r_try
    else:
        Q_retailer[idx_w, rand_idx] = penalty

# --- Retailer Execution ---
idx_r = np.argmax(Q_retailer[idx_w])
p, A = possible_retailer_actions[idx_r]

# --- Interaction & Update ---
pi_s, pi_r = get_profits(w, p, A)

if pi_r <= 0.05:
    r_s = penalty
else:
    r_s = pi_s
    if pi_s > best_results['pi_s']:
        best_results = {'w': w, 'p': p, 'A': A, 'pi_s': pi_s, 'pi_r': pi_r}

# Supplier Update
Q_supplier[idx_w] += alpha_s * (r_s - Q_supplier[idx_w])
```

```
    # ---【新增】紀錄歷史與收斂檢查 ---
    history_pi_s.append(r_s)

    if episode > min_episodes:
        # 計算最近 N 筆 與 前 N 筆 的平均差異
        recent_avg = np.mean(history_pi_s[-convergence_window:])
        prev_avg = np.mean(history_pi_s[-convergence_window*2 : -convergenc
e_window])

        # 條件 1: 變動極小
        # 條件 2: 平均利潤必須是正的 (避免卡在 penalty -10 收斂)
        # 條件 3: Epsilon 必須夠小 (避免還在瞎猜時運氣好連續幾次一樣就停了)
        if abs(recent_avg - prev_avg) < stability_tolerance and recent_avg > 0 an
d epsilon < 0.1:
            stable_counter += 1
        else:
            stable_counter = 0

        if stable_counter > stop_threshold:
            print(f"\n✅ Converged at episode {episode}!")
            print(f"   Stable Average Profit: {recent_avg:.4f}")
            break

    if episode % 1000 == 0:
        pbar.set_description(f"Eps: {epsilon:.2f} | Best w: {best_results['w']}")

# -------------------- 結果 --------------------
print("-" * 30)
print("【B1 MARL 最終結果】")
print(f"總訓練回數: {episode + 1}")
print(f"批發價 w: {best_results['w']}")
print(f"售價 p  : {best_results['p']}")
print(f"廣告 A  : {best_results['A']}")
print(f"供應商利潤: {best_results['pi_s']:.4f}")
print(f"零售商利潤: {best_results['pi_r']:.4f}")
```

```
print("-" * 30)

# ------------------- 驗證 (動態 A) -------------------
cand_w = best_results['w']
cand_p = best_results['p']
cand_A = best_results['A']

real_best_pi_r = -np.inf
real_best_p = -1
real_best_A = -1

for check_p in range(cand_w + 1, max_p + 1):
    for check_A in range(min_A, max_A + 1):
        Q_val = beta * (check_A ** epsA) * (check_p ** epsp)
        val_r = (check_p - cand_w) * Q_val - check_A

        if val_r > real_best_pi_r:
            real_best_pi_r = val_r
            real_best_p = check_p
            real_best_A = check_A

print(f"驗證: 在 w={cand_w} 下，數學最佳解 p={real_best_p}, A={real_best_A}, 利潤={real_best_pi_r:.4f}")

if cand_p == real_best_p and cand_A == real_best_A:
    print("✅ 完美收斂 (Perfect Convergence)。")
else:
    print("⚠️ 仍有差距。")
```

B2

- 批發價 w: 15、售價 p ：30、廣告 A ：1 (Supplier付)、供應商利潤: 0.5823、零售商利潤: 1.8257
- 會在20000左右收斂

```
import numpy as np
import random
import matplotlib.pyplot as plt
from tqdm import tqdm

# ==========================================
# (B.2 New) Stackelberg: Supplier Pays Ad
# ==========================================

# ------------------ 參數設定 ------------------
n_episodes = 100000
gamma = 0.9

alpha_r = 1.0       # Retailer: 快速適應
alpha_s = 0.05      # Supplier: 穩定學習

epsilon_start = 0.5
epsilon_end = 0.01
decay_rate = 0.9999

# --- 收斂判斷參數 ---
min_episodes = 5000
convergence_window = 1000
stability_tolerance = 0.005
stable_counter = 0
stop_threshold = 20
history_pi_s = []
# --------------------

beta = 20.0  # 這題市場大小參數為20。若為10，供給商profit 為負，因為市場不夠大
epsA = 0.5
epsp = -1.5
c = 2
```

```
max_w = 15; min_w = c + 1
max_p = 30; min_p = c + 1
max_A = 15; min_A = 1
penalty = -10.0

# ------------------- 動作空間定義 -------------------
# Supplier 決定 (w, A)
possible_supplier_actions = []
for w in range(min_w, max_w + 1):
    for A in range(min_A, max_A + 1):
        possible_supplier_actions.append((w, A))

# Retailer 決定 (p)
possible_retailer_actions = list(range(min_w, max_p + 1)) # p 的範圍

n_actions_s = len(possible_supplier_actions)
n_actions_r = len(possible_retailer_actions)

# Q-Tables
Q_supplier = np.zeros(n_actions_s)
# Retailer 的 Q 表依照 Supplier 的動作 (w, A) 來索引
Q_retailer = np.zeros((n_actions_s, n_actions_r))

# ------------------- 函數 -------------------
def get_profits(w, A, p):
    if p <= w: return penalty, penalty
    Q = beta * (A ** epsA) * (p ** epsp)

    # B2: Supplier 付 A
    pi_s = (w - c) * Q - A
    pi_r = (p - w) * Q

    return pi_s, pi_r

# ------------------- 主訓練 -------------------
best_results = {'w': 0, 'p': 0, 'A': 0, 'pi_s': -np.inf, 'pi_r': -np.inf}
```

```
epsilon = epsilon_start

print("【B2 New: Stackelberg (Supplier Pays Ad)】Training Start...")
pbar = tqdm(range(n_episodes))

for episode in pbar:
    epsilon = max(epsilon_end, epsilon * decay_rate)

    # 1. Supplier Move (選 w, A)
    if random.random() < epsilon:
        idx_s = random.randint(0, n_actions_s - 1)
    else:
        idx_s = np.argmax(Q_supplier)
    w, A = possible_supplier_actions[idx_s]

    # 2. Retailer Thinking Phase (針對目前的 w, A 找出最佳 p)
    n_thinking_steps = 1000

    # 記憶保護：先檢查已知的最佳解
    current_best_r_idx = np.argmax(Q_retailer[idx_s])
    p_best_check = possible_retailer_actions[current_best_r_idx]
    _, pi_best_check = get_profits(w, A, p_best_check)
    if pi_best_check > 0:
        Q_retailer[idx_s, current_best_r_idx] = pi_best_check

    for _ in range(n_thinking_steps):
        rand_r_idx = random.randint(0, n_actions_r - 1)
        p_try = possible_retailer_actions[rand_r_idx]

        _, pi_r_try = get_profits(w, A, p_try)

        if pi_r_try > 0:
            Q_retailer[idx_s, rand_r_idx] = pi_r_try
        else:
            Q_retailer[idx_s, rand_r_idx] = penalty
```

```
    # 3. Retailer Execution
    idx_r = np.argmax(Q_retailer[idx_s])
    p = possible_retailer_actions[idx_r]

    # 4. Interaction
    pi_s, pi_r = get_profits(w, A, p)

    # 生存檢查
    if pi_r <= 0.05:
        r_s = penalty
    else:
        r_s = pi_s
        if pi_s > best_results['pi_s']:
            best_results = {'w': w, 'p': p, 'A': A, 'pi_s': pi_s, 'pi_r': pi_r}

    # Supplier Update
    Q_supplier[idx_s] += alpha_s * (r_s - Q_supplier[idx_s])

    # 5. 收斂檢查
    history_pi_s.append(r_s)
    if episode > min_episodes:
        recent_avg = np.mean(history_pi_s[-convergence_window:])
        prev_avg = np.mean(history_pi_s[-convergence_window*2 : -convergenc
e_window])

        if abs(recent_avg - prev_avg) < stability_tolerance and recent_avg > 0 an
d epsilon < 0.1:
            stable_counter += 1
        else:
            stable_counter = 0

        if stable_counter > stop_threshold:
            print(f"\n✅ Converged at episode {episode}!")
            print(f"   Stable Average Profit: {recent_avg:.4f}")
            break
```

```
    if episode % 1000 == 0:
        pbar.set_description(f"Eps: {epsilon:.2f} | Best S_Profit: {best_results['pi_
s']:.2f}")

# ------------------- 結果 -------------------
print("-" * 30)
print("【B2 New 最終結果】")
print(f"總訓練回數: {episode + 1}")
print(f"批發價 w: {best_results['w']}")
print(f"售價 p  : {best_results['p']}")
print(f"廣告 A  : {best_results['A']} (Supplier付)")
print(f"供應商利潤: {best_results['pi_s']:.4f}")
print(f"零售商利潤: {best_results['pi_r']:.4f}")
print("-" * 30)
```

B3：

- 批發價 w: 15、售價 p ：30、廣告 As：1 (Supplier)、廣告 Ar：1 (Retailer)、供應商利潤: 0.1189、零售商利潤: 0.2910
- 會在28000-30000間收斂

```
import numpy as np
import random
import matplotlib.pyplot as plt
from tqdm import tqdm

# ==========================================
# (B.3 New) Stackelberg: Shared Ad Fee
# ==========================================

# ------------------- 參數設定 -------------------
n_episodes = 100000
gamma = 0.9
```

```
alpha_r = 1.0
alpha_s = 0.05

epsilon_start = 0.5
epsilon_end = 0.01
decay_rate = 0.9999

# --- 收斂判斷參數 ---
min_episodes = 5000
convergence_window = 1000
stability_tolerance = 0.005
stable_counter = 0
stop_threshold = 20
history_pi_s = []
# --------------------

beta = 10.0
epsA = 0.5
epsp = -1.5
c = 2

max_w = 15; min_w = c + 1
max_p = 30; min_p = c + 1
# 設定各別廣告上限
max_A_individual = 10
min_A_individual = 1

penalty = -10.0

# ------------------- 動作空間定義 ------------------
# Supplier 決定 (w, As)
possible_supplier_actions = []
for w in range(min_w, max_w + 1):
    for As in range(min_A_individual, max_A_individual + 1):
        possible_supplier_actions.append((w, As))
```

```
# Retailer 決定 (p, Ar)
possible_retailer_actions = []
for p in range(min_w, max_p + 1):
    for Ar in range(min_A_individual, max_A_individual + 1):
        possible_retailer_actions.append((p, Ar))

n_actions_s = len(possible_supplier_actions)
n_actions_r = len(possible_retailer_actions)

# Q-Tables
Q_supplier = np.zeros(n_actions_s)
# Retailer 的 Q 表依賴於 Supplier 的 (w, As)
Q_retailer = np.zeros((n_actions_s, n_actions_r))

# -------------------- 函數 --------------------
def get_profits(w, As, p, Ar):
    if p <= w: return penalty, penalty

    A_total = As + Ar
    if A_total <= 0: return penalty, penalty # 總廣告量不能為 0

    Q = beta * (A_total ** epsA) * (p ** epsp)

    # B3: 共同分擔
    pi_s = (w - c) * Q - As
    pi_r = (p - w) * Q - Ar

    return pi_s, pi_r

# -------------------- 主訓練 --------------------
best_results = {'w': 0, 'p': 0, 'As': 0, 'Ar': 0, 'pi_s': -np.inf, 'pi_r': -np.inf}
epsilon = epsilon_start

print("【B3 New: Stackelberg (Shared Ad)】Training Start...")
pbar = tqdm(range(n_episodes))
```

```
for episode in pbar:
    epsilon = max(epsilon_end, epsilon * decay_rate)

    # 1. Supplier Move (選 w, As)
    if random.random() < epsilon:
        idx_s = random.randint(0, n_actions_s - 1)
    else:
        idx_s = np.argmax(Q_supplier)
    w, As = possible_supplier_actions[idx_s]

    # 2. Retailer Thinking Phase (針對目前的 w, As 找出最佳 p, Ar)
    n_thinking_steps = 1000

    # 記憶保護
    current_best_r_idx = np.argmax(Q_retailer[idx_s])
    p_best, Ar_best = possible_retailer_actions[current_best_r_idx]
    _, pi_best_check = get_profits(w, As, p_best, Ar_best)
    if pi_best_check > 0:
        Q_retailer[idx_s, current_best_r_idx] = pi_best_check

    for _ in range(n_thinking_steps):
        rand_r_idx = random.randint(0, n_actions_r - 1)
        p_try, Ar_try = possible_retailer_actions[rand_r_idx]

        _, pi_r_try = get_profits(w, As, p_try, Ar_try)

        if pi_r_try > 0:
            Q_retailer[idx_s, rand_r_idx] = pi_r_try
        else:
            Q_retailer[idx_s, rand_r_idx] = penalty

    # 3. Retailer Execution
    idx_r = np.argmax(Q_retailer[idx_s])
    p, Ar = possible_retailer_actions[idx_r]

    # 4. Interaction
```

```
    pi_s, pi_r = get_profits(w, As, p, Ar)

    # 生存檢查
    if pi_r <= 0.05:
        r_s = penalty
    else:
        r_s = pi_s
        if pi_s > best_results['pi_s']:
            best_results = {'w': w, 'p': p, 'As': As, 'Ar': Ar, 'pi_s': pi_s, 'pi_r': pi_r}

    # Supplier Update
    Q_supplier[idx_s] += alpha_s * (r_s - Q_supplier[idx_s])

    # 5. 收斂檢查
    history_pi_s.append(r_s)
    if episode > min_episodes:
        recent_avg = np.mean(history_pi_s[-convergence_window:])
        prev_avg = np.mean(history_pi_s[-convergence_window*2 : -convergence_window])

        if abs(recent_avg - prev_avg) < stability_tolerance and recent_avg > 0 and epsilon < 0.1:
            stable_counter += 1
        else:
            stable_counter = 0

        if stable_counter > stop_threshold:
            print(f"\n✅ Converged at episode {episode}!")
            print(f"   Stable Average Profit: {recent_avg:.4f}")
            break

    if episode % 1000 == 0:
        pbar.set_description(f"Eps: {epsilon:.2f} | Best S_Profit: {best_results['pi_s']:.2f}")

# ------------------- 結果 -------------------
```

```
print("-" * 30)
print(" 【B3 New 最終結果】 ")
print(f"總訓練回數: {episode + 1}")
print(f"批發價 w: {best_results['w']}")
print(f"售價 p  : {best_results['p']}")
print(f"廣告 As : {best_results['As']} (Supplier)")
print(f"廣告 Ar : {best_results['Ar']} (Retailer)")
print(f"供應商利潤: {best_results['pi_s']:.4f}")
print(f"零售商利潤: {best_results['pi_r']:.4f}")
print("-" * 30)
```

三、關鍵作法

作法： 使用了 Nested Loop（巢狀結構)。

外層：供應商選擇 w。

內層：強制零售商進行 1000 次思考 (`n_thinking_steps`)，直到找出該 w 下的最好 (p, A)，才回傳結果給供應商。

優勢： 這完全符合教授說的邏輯：「先把 w 固定下來，單獨測試零售商最佳結果」。

do not call 每個機率