(1) 演算法公式（LaTeX）

```
\textbf{Softmax Policy Algorithm}

\begin{align*}
& \text{Initialize: } Q(a) = 0, \, N(a) = 0 \quad \forall a \in \{1, \dots, k\} \\
& \text{For each time step } t = 1 \text{ to } T: \\
& \quad P(a) = \frac{e^{Q(a)/\tau}}{\sum_{b=1}^{k} e^{Q(b)/\tau}} \\
& \quad \text{Sample } a_t \sim P(a) \\
& \quad \text{Receive reward } r_t \sim \mathcal{N}(\mu_{a_t}, 1) \\
& \quad N(a_t) \leftarrow N(a_t) + 1 \\
& \quad Q(a_t) \leftarrow Q(a_t) + \frac{1}{N(a_t)} (r_t - Q(a_t))
\end{align*}
```

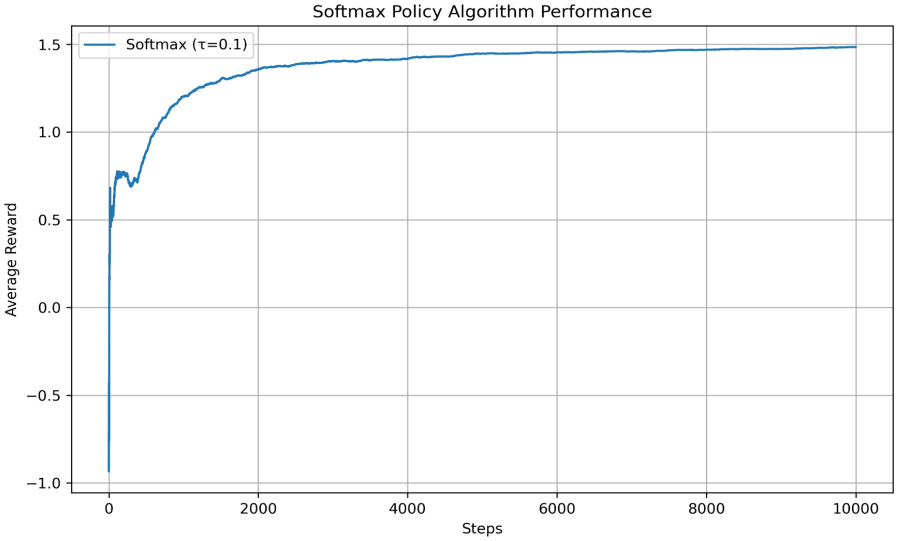
(2) ChatGPT 提示語

請實作 Softmax（Boltzmann 探索）策略，利用溫度參數控制選擇機率，觀察其如何隨時間動態調整探索與利用的平衡。

(3) Python 程式碼

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

np.random.seed(42)
k = 10
steps = 10000
tau = 0.1
true_rewards = np.random.normal(0, 1, k)

Q = np.zeros(k)
N = np.zeros(k)
rewards = np.zeros(steps)

def softmax(x, tau):
    x = x / tau
    x -= np.max(x)
    exp_x = np.exp(x)
    return exp_x / np.sum(exp_x)

for t in range(steps):
    prob = softmax(Q, tau)
    a = np.random.choice(np.arange(k), p=prob)
    reward = np.random.normal(true_rewards[a], 1)

    N[a] += 1
    Q[a] += (reward - Q[a]) / N[a]
    rewards[t] = reward

cumulative_average = np.cumsum(rewards) / (np.arange(steps) + 1)

plt.figure(figsize=(10, 6))
plt.plot(cumulative_average, label=f"Softmax (tau={tau})")
plt.xlabel("Steps")
plt.ylabel("Average Reward")
plt.title("Softmax Policy Algorithm Performance")
plt.legend()
plt.grid(True)
plt.savefig("softmax_result.png", dpi=300)
plt.show()
```

(3-1) 圖表



(4) 結果分析
時間複雜度： O(T * k)
空間複雜度： O(k + T)

說明：
- 利用 softmax 機率動態選擇 action
- 溫度參數 tau 可調節探索程度，低 tau 趨近 greedy
- 適合 reward 非常接近的情境，可避免早期誤判