(1) 演算法公式（LaTeX）

```
\textbf{Epsilon-Greedy 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 \text{With probability } \varepsilon, \text{ choose a random action } a_t \in \{1, \dots, k
& \quad \text{With probability } 1 - \varepsilon, \text{ choose } a_t = \arg\max_a Q(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)} \left(r_t - Q(a_t)\right)
\end{align*}
```

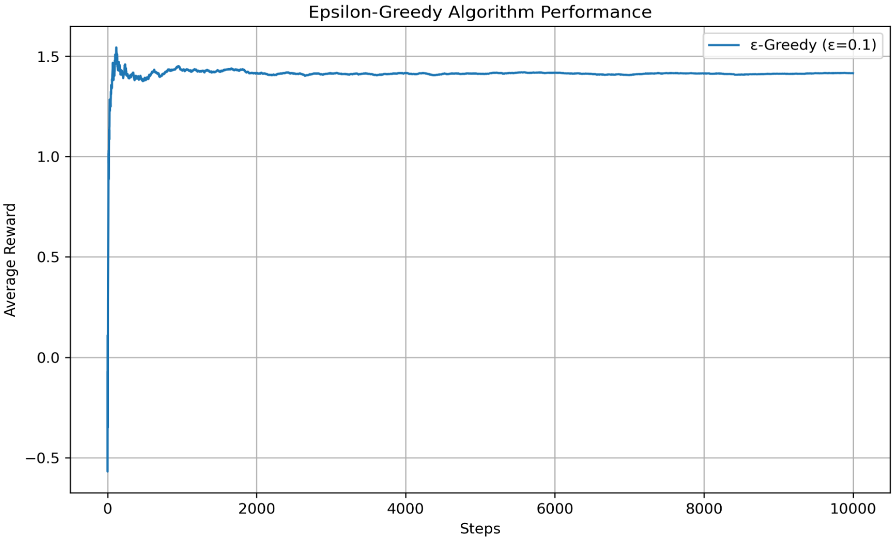
(2) ChatGPT 提示語

請實作 epsilon-greedy
多臂拉霸演算法，需包含探索與利用的邏輯，並用遞增平均更新動作價值，繪製報酬曲線。

(3) Python 程式碼

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

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

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

for t in range(steps):
    if np.random.rand() < epsilon:
        a = np.random.randint(k)
    else:
        a = np.argmax(Q)
    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"ε-Greedy (ε={epsilon})")
plt.xlabel("Steps")
plt.ylabel("Average Reward")
plt.title("Epsilon-Greedy Algorithm Performance")
plt.legend()
plt.grid(True)
plt.savefig("epsilon_greedy_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：

- 結構簡單，易於實作
- 探索與利用比依賴 epsilon 參數
- 適合靜態環境但對 epsilon 選擇較敏感