

(1) 演算法公式 (LaTeX)

```
\textbf{Thompson Sampling Algorithm (Gaussian Rewards)}

\begin{align*}
& \& \text{Initialize: } \hat{\mu}_a = 0, \& n_a = 0 \quad \forall a \in \{1, \dots, k\} \\
& \& \text{For each time step } t = 1 \& \text{ to } T: \\
& \& \quad \theta_a \sim \text{mathcal{N}}(\hat{\mu}_a, 1 / (n_a + 1)) \quad \forall a \\
& \& \quad a_t = \arg\max_a \theta_a \\
& \& \quad \text{Receive reward } r_t \sim \text{mathcal{N}}(\mu_{a_t}, 1) \\
& \& \quad n_{a_t} \leftarrow n_{a_t} + 1 \\
& \& \quad \hat{\mu}_{a_t} \leftarrow \hat{\mu}_{a_t} + \frac{1}{n_{a_t}}(r_t - \hat{\mu}_{a_t})
\end{align*}
```

(2) ChatGPT 提示語

請實作 Gaussian reward 下的 Thompson Sampling 方法，根據每個 arm 的後驗分布進行抽樣選擇，並更新估計值，觀察其自動平衡探索與利用的能力。

(3) Python 程式碼

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

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

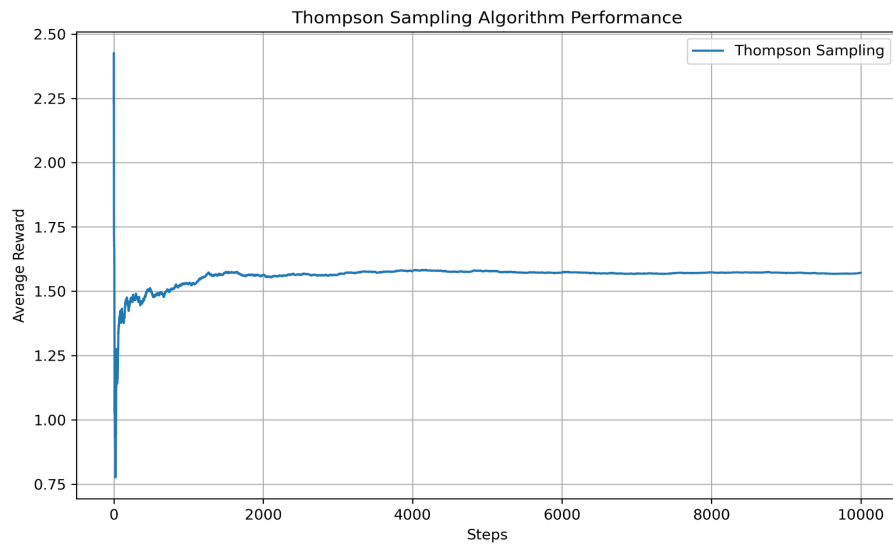
mu_hat = np.zeros(k)
N = np.zeros(k)
rewards = np.zeros(steps)

for t in range(steps):
    samples = np.random.normal(mu_hat, 1 / (np.sqrt(N + 1)))
    a = np.argmax(samples)
    reward = np.random.normal(true_rewards[a], 1)
    N[a] += 1
    mu_hat[a] += (reward - mu_hat[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="Thompson Sampling")
plt.xlabel("Steps")
plt.ylabel("Average Reward")
plt.title("Thompson Sampling Algorithm Performance")
plt.legend()
plt.grid(True)
plt.savefig("thompson_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：

- 屬於貝葉斯方法，根據後驗分布進行決策
- 可自動調節探索強度，無需手動設定 epsilon 或 tau
- 實務上表現優異，尤其適用於非穩定或有不確定性的場景