# Risk-Thompson Sampling

February 16, 2026

## 1 $\rho$-Thompson Sampling

First, generate some helper functions to simplify code later.

[1]:
```
import random
from random import betavariate
import numpy as np
import matplotlib.pyplot as plt

# (Replace `true_p` with your environment's unknown means when integrating.)

def sample_thetas(alpha, beta):
    """Sample theta_k ~ Beta(alpha_k, beta_k) for each arm k."""
    return [betavariate(alpha[k], beta[k]) for k in range(len(alpha))]


def kl_bernoulli(p, q, eps=1e-12):
    """KL divergence between Bernoulli(p) and Bernoulli(q)."""
    p = min(max(p, eps), 1 - eps)
    q = min(max(q, eps), 1 - eps)
    return p * np.log(p / q) + (1 - p) * np.log((1 - p) / (1 - q))


def k_inf_rho(p, rho_fn, rho_max, grid_size=100001, eps=1e-12, min_gap=0.1):
    """Compute K_inf^rho(p, rho_max) via grid search over q in [0,1]."""
    qs = np.linspace(0.0, 1.0, grid_size)
    rho_vals = np.array([rho_fn(q) for q in qs])
    # strict constraint rho(q) > rho_max (use eps for stability)
    mask = rho_vals >= (rho_max - eps)
    mask &= np.abs(qs - p) >= min_gap
    if not np.any(mask):
        return np.inf
    qs_feasible = qs[mask]
    kl_vals = np.array([kl_bernoulli(p, q) for q in qs_feasible])
    return float(np.min(kl_vals))
def select_arm(scores):
    """Select arm with highest score."""
    return scores.index(max(scores))
```

```
def rho_expectation(p):
    return p

def rho_cvar(p, alpha=0.05):
    return min(p / (1 - alpha), 1.0)

def rho_mean_variance(p, gamma=2.0):
    # rho = gamma * mean + negative variance
    return gamma * p - p * (1 - p)

def rho_entropic(p, theta=0.5):
    return -(1.0 / theta) * np.log((1 - p) + p * np.exp(-theta))


def rho_1(p):
    return 0.5 * rho_cvar(p, alpha=0.05) + 0.5 * rho_entropic(p, theta=0.5)

def rho_2(p):
    return rho_mean_variance(p, gamma=2.0) + np.exp(2 * p)

RHO_FUNCS = [
    ('Expectation', rho_expectation),
    ('CVaR_0.05', lambda p: rho_cvar(p, alpha=0.05)),
    ('Mean-Variance (Gamma = 2)', lambda p: rho_mean_variance(p, gamma=2.0)),
    ('Entropic Risk (Theta = 0.5)', lambda p: rho_entropic(p, theta=0.5)),
    ('rho_1', rho_1),
    ('rho_2', rho_2),
]
```

Initialize:

- number of arms: $K$
- number of rounds: $T$
- For each arm $k = 1, \dots, K$ set $\alpha_k = 1$, $\beta_k = 1$ (prior Beta$(1, 1)$).

The implementation cell below sets imports and parameters.

[2]:
```
# Configuration
K = 10            # number of arms
# fixed instance for all experiments
rng = np.random.default_rng(0)
true_p = 0.25 + 0.75 * rng.random(K)

def play_env(arm):
    """Simulate pulling an arm and observe reward."""
    return 1 if random.random() < true_p[arm] else 0
```

```
T = 10000           # number of rounds
NUM_EXPERIMENTS = 100  # number of independent runs
alpha = [1]*K
beta = [1]*K


# rewards and counts track empirical outcomes per arm
rewards = [0]*K  # total reward observed per arm
counts = [0]*K   # number of times each arm was played
```

For an initial exploration phase, pull each arm once:

- Observe reward $r_t \in \{0, 1\}$
- If $r_t = 1$ then $\alpha_{A_t} \leftarrow \alpha_{A_t} + 1$, else $\beta_{A_t} \leftarrow \beta_{A_t} + 1$.

[3]:
```
# Initial exploration: pull each arm once
for arm in range(K):
    r = play_env(arm)
    counts[arm] += 1
    if r == 1:
        alpha[arm] += 1
    else:
        beta[arm] += 1
    rewards[arm] += r
```

In each turn:

- For each $k = 1, \ldots, K$ draw $\theta_k \sim \text{Beta}(\alpha_k, \beta_k)$.
- Compute $\text{score}_k = \rho(\theta_k)$ (here $\rho(x) = x$, so $\text{score}_k = \theta_k$).
- Choose $A_t \leftarrow \arg\max_k \text{score}_k$.
- Pull arm $A_t$ and observe reward $r_t \in \{0, 1\}$.
- If $r_t = 1$ then $\alpha_{A_t} \leftarrow \alpha_{A_t} + 1$, else $\beta_{A_t} \leftarrow \beta_{A_t} + 1$.

[4]:
```
# Multiple experiment loop for each rho
results = {}
for rho_name, rho in RHO_FUNCS:
    # precompute rho_max for this rho
    rho_vals = [rho(p) for p in true_p]
    rho_max = max(rho_vals)

    all_cumulative_regrets = []  # list of lists, each length T
    for exp in range(NUM_EXPERIMENTS):
        # reset per-run stats
        alpha = [1]*K
        beta = [1]*K
        rewards = [0]*K
        counts = [0]*K

        cumulative_regret = []
```

```
        cum = 0.0

        for t in range(1, T + 1):
            theta = sample_thetas(alpha, beta)
            scores = [rho(th) for th in theta]
            A = select_arm(scores)
            r = play_env(A)
            counts[A] += 1
            rewards[A] += r
            if r == 1:
                alpha[A] += 1
            else:
                beta[A] += 1
            # expected regret: rho_max - rho(nu_A)
            inst_regret = rho_max - rho(true_p[A])
            cum += inst_regret
            cumulative_regret.append(cum)

        all_cumulative_regrets.append(cumulative_regret)

    results[rho_name] = np.array(all_cumulative_regrets)
```

Summarise the results in the plotted graph below.

Define the mixed risk functionals:

- $\rho_1 = 0.5\,\mathrm{CVaR}_{0.05} + 0.5\,\rho_{\mathrm{ER}_{0.5}}$
- $\rho_2(\nu_p) = \rho_{\mathrm{MV}_2}(\nu_p) + \exp(2p)$

[5]:
```
# Plot regret for each rho
n = len(RHO_FUNCS)
cols = 3
rows = (n + cols - 1) // cols
fig, axes = plt.subplots(rows, cols, figsize=(6*cols, 4*rows))
axes = np.array(axes).reshape(-1)
x = np.arange(1, T + 1, dtype=float)

for idx, (rho_name, rho) in enumerate(RHO_FUNCS):
    ax = axes[idx]
    all_cumulative_regrets_np = results[rho_name]
    mean_regret = all_cumulative_regrets_np.mean(axis=0)
    p16 = np.percentile(all_cumulative_regrets_np, 16, axis=0)
    p84 = np.percentile(all_cumulative_regrets_np, 84, axis=0)

    # Theoretical coefficient per rho
    rho_vals = [rho(p) for p in true_p]
    rho_max = max(rho_vals)
    gaps = [rho_max - rho_vals[k] for k in range(K)]
```

```
    k_inf = [k_inf_rho(true_p[k], rho, rho_max) for k in range(K)]
    theory_coef = 0.0
    for k in range(K):
        if rho_vals[k] < rho_max and k_inf[k] > 0 and np.isfinite(k_inf[k]):
            theory_coef += gaps[k] / k_inf[k]
    theory_curve = theory_coef * np.log(x)

    ax.plot(x, mean_regret, color='green', label='Empirical Cumulative Regret')
    ax.fill_between(x, p16, p84, alpha=0.3, color='green', label='1-Sigma Band')
    ax.plot(x, theory_curve, '--', color='red', label='Theoretical Lower Bound')

    ax.set_title(rho_name)
    ax.set_xlim(0, T)
    ax.set_xticks(np.linspace(0, T, 11))
    ax.set_xlabel('Round')
    ax.set_ylabel('Regret')
    ax.grid(True)

# Hide any unused axes
for j in range(n, len(axes)):
    axes[j].set_visible(False)

# Use a single legend
handles, labels = axes[0].get_legend_handles_labels()
fig.legend(handles, labels, loc='upper center', ncol=3)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```