Contents

1 Chapter 0 — Exercises & Labs (Application Mode)

We keep every experiment executable. These warm-ups extend the Chapter 0 toy environment and force you to compare learning curves against the analytical expectations stated in the draft.

1.1 Lab 0.1 — Tabular Boost Search (Toy World)

Goal: reproduce the $\geq 90\%$ of oracle guarantee using the public `scripts/ch00/toy_problem_solution.py`.

```
from scripts.ch00.toy_problem_solution import (
    TabularQLearning,
    discretize_action_space,
    run_learning_experiment,
)

actions = discretize_action_space(n_bins=5, a_min=-1.0, a_max=1.0)
agent = TabularQLearning(
    actions,
    epsilon_init=0.9,
    epsilon_decay=0.995,
    epsilon_min=0.05,
    learning_rate=0.15,
)

results = run_learning_experiment(
    agent,
    n_train=800,
    eval_interval=40,
    n_eval=120,
```

```
    seed=314,
)
print(f"Final mean reward: {results['final_mean']:.2f} (target >= 0.90 * oracle)")
print(f"Per-user reward: {results['final_per_user']}")
```

Output (representative):

```
Final mean reward: 16.13 (target >= 0.90 * oracle)
Per-user reward: {'price_hunter': 14.62, 'premium': 22.65, 'bulk_buyer': 11.02}
```

What to analyze 1. Compare the printed percentages against the oracle baseline emitted by `scripts/ch00/toy_problem_solution.py`. 2. Highlight which segments remain under-optimized and tie that back to action-grid resolution (Section 0.3). 3. Export the figure `toy_problem_learning_curves.png` produced by the script and annotate regime changes (exploration vs exploitation) in your lab notes.

1.2 Exercise 0.2 — Stress-Testing Reward Weights

This exercise validates the sensitivity discussion in Section 0.3.2. Modify the toy reward to over-weight engagement and measure how Q-learning reacts:

```
from scripts.ch00.toy_problem_solution import (
    USER_TYPES,
    compute_reward,
    rank_products,
    simulate_user_interaction,
)
import numpy as np

alpha, beta, delta = 0.6, 0.3, 0.3  # delta intentionally oversized
rng = np.random.default_rng(7)
user = USER_TYPES["price_hunter"]
ranking = rank_products(0.8, 0.1)
interaction = simulate_user_interaction(user, ranking, seed=7)
reward = compute_reward(interaction, alpha=alpha, beta=beta, gamma=0.0, delta=delta)
print(f"Reward with delta={delta:.1f}: {reward:.2f}")
```

Output:

```
Reward with delta=0.3: 0.30
```

Discussion prompts - Explain why the oversized δ inflates reward despite lower GMV, linking directly to the `delta/alpha` ≤ 0.10 guideline in Chapter 1. - Propose how you would encode the same guardrail once you migrate to the full simulator (`zoosim/dynamics/reward.py` assertions already enforce it).

1.3 Exercise 0.1 — Reward Sensitivity Analysis

Goal: Compare learned policies under different reward configurations.

Three configurations to test: - **(a) Pure GMV:** $(\alpha, \beta, \delta) = (1.0, 0.0, 0.0)$ - **(b) Profit-focused:** $(\alpha, \beta, \delta) = (0.4, 0.5, 0.1)$ - **(c) Engagement-heavy:** $(\alpha, \beta, \delta) = (0.5, 0.2, 0.3)$

```
from scripts.ch00.lab_solutions import exercise_0_1_reward_sensitivity

results = exercise_0_1_reward_sensitivity(seed=42)
```

What to analyze: 1. Does the learned policy change across configurations? For which user types? 2. Which configuration shows the highest risk of "clickbait" behavior (high engagement, questionable quality)? 3. Connect your findings to the guardrail discussion in Chapter 1.

Time estimate: 20 minutes

1.4 Exercise 0.2 — Action Geometry and the Cold Start Problem

Learning objective: Understand how exploration strategy effectiveness depends on policy quality.

This is the pedagogical highlight of Chapter 0. You'll form a hypothesis, test it, discover it's wrong, diagnose why, and validate when the original intuition *does* hold.

1.4.1 Setup

We compare two ε-greedy exploration strategies on the toy world: - **Uniform exploration:** When exploring, sample ANY action uniformly from the 25-action grid - **Local exploration:** When exploring, sample only NEIGHBORS (± 1 grid cell) of the current best action

1.4.2 Part A — Form Hypothesis (5 min)

Before running experiments, predict: *Which strategy will converge faster?*

Write down your reasoning. The intuitive answer is "local exploration should be more efficient because it exploits structure near good solutions."

1.4.3 Part B — Cold Start Experiment (10 min)

Run both strategies from random initialization (Q=0 everywhere):

```
from scripts.ch00.lab_solutions import exercise_0_2_action_geometry

results = exercise_0_2_action_geometry(
    n_episodes_cold=500,
    n_episodes_warmup=200,
    n_episodes_refine=300,
    n_runs=5,
    seed=42,
)
```

Questions: 1. Which strategy wins? By how much? 2. Does this match your hypothesis?

1.4.4 Part C — Diagnosis (10 min)

The code reports action coverage. Examine how many of the 25 actions each strategy explored.

Questions: 1. Why does local exploration explore fewer actions? 2. What does "cold start problem" mean in this context? 3. Why is the local agent doing "local refinement of garbage"?

1.4.5 Part D — Warm Start Experiment (10 min)

The code also runs a warm start experiment: first train with uniform for 200 episodes, then compare strategies for 300 more episodes.

Questions: 1. How does the gap between strategies change after warm start? 2. Why is local exploration now competitive? 3. When would local exploration actually *win*?

1.4.6 Part E — Synthesis (15 min)

Connect your findings to real RL algorithms:

1. **SAC** uses entropy regularization that naturally decays. How does this relate to the cold start problem?
2. **ε-greedy** schedules typically decay ε from $0.9 \to 0.05$. Why?
3. **Optimistic initialization** (starting with high Q-values) is a common trick. How does it help with cold start?

Deliverable: Write a 1-paragraph guideline for choosing exploration strategies based on policy maturity.

Time estimate: 50 minutes total

1.5 Exercise 0.3 — Regret Analysis

Goal: Track cumulative regret and verify sublinear scaling.

```
from scripts.ch00.lab_solutions import exercise_0_3_regret_analysis

results = exercise_0_3_regret_analysis(n_train=2000, seed=42)
```

What to analyze: 1. Is regret sublinear? (Does average regret per episode decrease?) 2. Fit the curve to $\text{Regret}(T) \approx C \cdot T^{\alpha}$. What is α? 3. Compare to theory: constant ε-greedy gives $O(T^{2/3})$, decaying ε gives $O(\sqrt{T \log T})$

Time estimate: 20 minutes

1.6 Exercise 0.4 — Constrained Q-Learning with CM2 Floor

Goal: Add profitability constraint $\mathbb{E}[\text{CM2} \mid x, a] \geq \tau$ and study the GMV-CM2 tradeoff.

```
from scripts.ch00.lab_solutions import exercise_0_4_constrained_qlearning

results = exercise_0_4_constrained_qlearning(seed=42)
```

What to analyze: 1. Do you get a clean Pareto frontier? Why or why not? 2. What causes the high violation rates? 3. Propose an alternative approach (hint: Lagrangian relaxation, chance constraints)

Connection to Chapter 3: This motivates the CMDP formalism in Section 3.6.

Time estimate: 25 minutes

1.7 Exercise 0.5 — Bandit-Bellman Bridge (Conceptual)

Goal: Show that contextual bandit Q-learning is the $\gamma = 0$ case of MDP Q-learning.

```
from scripts.ch00.lab_solutions import exercise_0_5_bandit_bellman_bridge

results = exercise_0_5_bandit_bellman_bridge()
```

Theoretical derivation:

1. Write the Bellman optimality equation for Q-values
2. Set $\gamma = 0$ and simplify
3. Show that the resulting update rule matches the bandit Q-update

What to verify: 1. Do the numerical tests pass? 2. What happens to the "bootstrap target" $r + \gamma \max_{a'} Q(s', a')$ when $\gamma = 0$?

Connection to Chapter 11: Multi-episode search requires $\gamma > 0$ because today's ranking affects tomorrow's return probability.

Time estimate: 15 minutes

1.8 Summary: Exercise Time Budget

Exercise	Time	Key Concept
Lab 0.1	30 min	Q-learning on toy world
Ex 0.2 (stress)	10 min	Reward weight sensitivity
Ex 0.1	20 min	Policy sensitivity to rewards
Ex 0.2 (geometry)	**50 min**	**Cold start problem**
Ex 0.3	20 min	Regret analysis
Ex 0.4	25 min	Constrained RL
Ex 0.5	15 min	Bandit-MDP connection

Total: ~170 minutes (adjust based on depth of analysis)

1.9 Running All Solutions

```
# Run all exercises
python scripts/ch00/lab_solutions.py --all

# Run specific exercise
python scripts/ch00/lab_solutions.py --exercise lab0.1
python scripts/ch00/lab_solutions.py --exercise 0.2  # Action Geometry

# Interactive menu
python scripts/ch00/lab_solutions.py
```