Term Project

Deep Reinforcement Learning for Optimal Operation of a Semi-Batch Reactor

1. Introduction & Problem Statement

You will design a Deep Reinforcement Learning (RL) agent to act as an autonomous operator for a chemical reactor. Finding an economically optimal operating policy is challenging, especially when market conditions are not static; raw material costs and product values often fluctuate.

Your agent's mission is to maximize **economic profit**. To achieve this, it must learn to dynamically adapt its operating "recipe" in real-time based on stochastic market prices.

2. Process Dynamics

The plant consists of a single **Isothermal Semi-Batch Reactor**.

Reactor Setup

- Maximum volume: 100 L
- Initialization: Each episode starts with the reactor filled with 20 L of raw material A.
- Feed Concentrations:
  - Raw Material A: 10 mol/L
  - Raw Material B: 10 mol/L

Chemical Reactions

We produce a valuable product D from reactants A and B. However, a competing side reaction produces an undesirable, worthless byproduct U.

1. Main Reaction (Product): $A + B \rightarrow D$
   - Reaction kinetics: $r_D = k_D C_A C_B$
2. Side Reaction (Byproduct): $A + B \rightarrow U$
   - Reaction kinetics: $r_U = k_U C_A C_B^2$

Note: Process dynamics will be stochastic; it has some noises.

3. Economic Environment

Optimal operation depends on market prices, which are stochastic.

Fixed Cost

Raw Material A: $0.5 / mol

Stochastic Prices

The prices for Raw Material B and Product D are not fixed. They update daily (every midnight) following unknown stochastic processes.

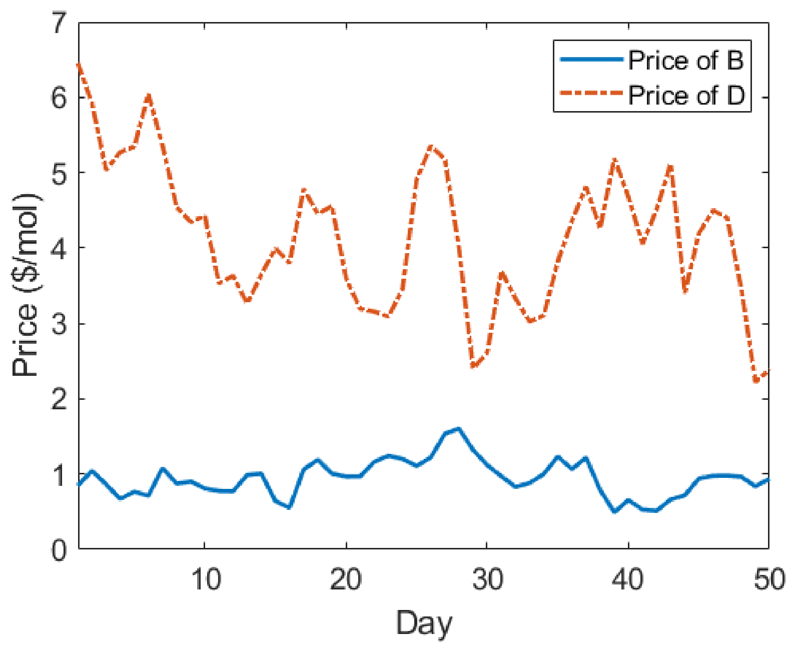

Figure 1. Example of price fluctuation of reactant B and product D.

4. Problem Definition

You are provided with a Python simulator (*Simulator* class in BatchEnv.py) that implements this environment.

- Objective: Maximize the long-term cumulative net profit.
- Timestep: 2.0 hours. The agent makes one decision every 2 hours.
- Observation: 7 continuous values: $[C_A, C_B, C_D, C_U, V, P_B, P_D]$
    - C_A, C_B, C_D, C_U: Current concentrations of species A, B, D and U. (mol/L)
    - V: Current volume of mixture (L)
    - P_B, P_D: Current market prices ($/mol)
    - Additional info about current time is available but not part of the standard observation. You may use them for learning if necessary.

- Action
    - At each timestep, the agent must decide whether to **stop & start next batch** or **continue operating**.
        - Stop & start next batch
            - Ends the current batch.
            - The product is sold at the current market price if the molar purity of D (relative to byproduct U) exceeds 70%.
                - Condition: $C_D/(C_D + C_U) \geq 0.7$
            - If the purity threshold is not met, the mixture is discarded (Revenue = \$0).
            - Cleans the reactor (takes 2 hours / 1 timestep).
            - Starts a new batch filled with 20L of raw material A.
        - Continue operating:
            - Feeds materials into the reactor. You may feed both A and B simultaneously.
            - Feed A: Range 0.0~10.0 L/hr
            - Feed B: Range 0.0~10.0 L/hr

---

5. Project Description

Your task is to develop, analyze, and justify an RL agent that masters this environment.

5-1. Baseline Implementation

Your first step is to implement a simple DQN to establish a baseline agent.

Markov Process formulation

- State: Use the 6 observations defined above directly.
- Action: Discretize feed rates into 3 bins: [0.0, 5.0, 10.0] L/hr. This results in 10 discrete actions:
  - 1 Action for Stop & start next batch
  - 9 Actions for Feeding (3 bins for A × 3 bins for B)
- Reward:
  - The net profit from the operation (revenue from sold D - cost of raw materials fed) is given as a reward when each batch operation ends.

Gymnasium environment based on the given Markov Process formulation **is provided (*BatchEnv* class in BatchEnv.py)**.

Implement a vanilla DQN, train it on the provided BatchEnv, and analyze the results. It is likely that your agent is not well-training and suboptimal; this is a simple baseline. Discuss its suboptimality by inspecting the resulting trajectory using your domain knowledge.

5-2. Improvement & Justification

Improve your agent by identifying and addressing the issues. You may modify the Markov Formulation (e.g., reward shaping, state/action spaces) provided you do not alter the fundamental physics defined in Sections 1-4.

For improvements you make, you should follow this logical flow in your report:

1. **Hypothesis:** What is preventing the agent from learning the optimal policy?
2. **Proposed Solution:** What technique will you apply?
3. **Result:** Demonstrate that your solution worked (or didn't). Compare learning curves or trajectories before and after the change.

You can refer to the following diagnostic checklist. (This is **not** an exhaustive list!)

- Reward: Is the reward too sparse or not informative? Would reward shaping help?
- Action Space: Is the discretization too coarse? Would a different discretization or keeping a continuous action space work better?
- Exploration: Is the agent exploring sufficiently to discover good actions? Does it exploit effectively over training?
- Stability: Is learning stable? Consider techniques to reduce variance.
- Algorithm: Is the chosen RL algorithm efficient and suitable for the problem? Would different algorithms such as actor-critic algorithms work better?

5-3. Final Deliverable

Hand in your report, script files, and your RL agent. Your report must justify your final design choices using the evidence collected during training.

Inspect the final policy. Does the agent behave like a skilled operator? Does it react logically to market prices? Demonstrate its behavior through sampled trajectories.