**Reinforcement Learning** Aksel Kretsinger-Walters, adk2164
**Problem Set 1** adk2164@columbia.edu
ORCS 4529: Fall 2025 **Due:** 10/02/2025 11:59 PM ET

---

# Problem 1

### MDP Formulation

I've decided to formulate the problem as a finite horizon undiscounted MDP problem. There are 5 rounds, and 4 decisions to be made (the last digit is forced). The state consists of a mask (ie boolean vector) indicating which positions are still available (we will define $r$ to be the number of rounds of the game, and denote this mask as $\mathbf{m} \in \mathbb{R}^{r\times 1}$) and the currently observed digit $d$. The round number $t$ is implicit in the state as $t = \sum_p m_p$.

Note that I have **not** included the partially filled number in the state. It's easier to bake this into the reward structure instead of carrying it around in the state.

**Indexing convention:** I will index the positions of the five-digit number from left to right, starting at 1 (ie the most significant digit is position 1, and the least significant digit is position 5).

On that note, I'll define the reward structure, and some of my reasons for the setup. The reward function is simply the immediate value of placing digit $d$ in position $p$, which is $d \cdot 10^{r-p}$. This feels natural, intuitive, and doesn't require additional bookkeeping nor recalculation at the completion of the game.

**States.** At the point of each decision in round $t \in [r]$, the state is

$$s_t = (\mathbf{m}, d)$$

Like I touched upon in the intro, the state consists of a mask $\mathbf{m} \in \{0,1\}^{r\times 1}$ indicating which positions are filled ($m_p = 1$ if position $p$ is filled, else 0), and the currently observed digit $d \in \{0,\dots,9\}$.

**Actions.** From state $(\mathbf{m}, d)$, choose any empty position:

$$\mathcal{A}(\mathbf{m}) = \{\, p \in \{1,\dots,r\} :\ m_p = 0 \,\}.$$

**Transition model.** Let's define the helper function $\text{fill}(\mathbf{m}, p)$ that returns the mask with position $p$ set to 1:

$$\text{fill}(\mathbf{m}, p) = (m_1, \dots, m_{p-1}, 1, m_{p+1}, \dots, m_r).$$

After taking action $a = p$ (placing digit $d$ at position $p$), the next state is

$$\mathbf{m}' = \text{fill}(\mathbf{m}, p), \qquad d' \sim \text{Unif}\{0,\dots,9\}$$

For each $d' \in \{0, \ldots, 9\}$,

$$\Pr\big(\big(\text{fill}(\mathbf{m}, p), d'\big) \,\big|\, (\mathbf{m}, d), a = p\big) = \tfrac{1}{10},$$

and all other next states have probability 0.

In plain English, the next state is the same as the current state except that position $p$ is now filled, and the next digit is drawn uniformly from $\{0, \ldots, 9\}$.

**Rewards.** Receive the value of placing digit $d$ in position $p$:

$$r\big((\mathbf{m}, d), a = p\big) = d\, 10^{r-p}.$$

The nice thing about this reward structure is that there's no need for additional calculation at the end of the game.

**Objective.** Maximize expected total reward over the five decisions:

$$\max_{\pi} \; \mathbb{E}_{\pi}\Big[\sum_{t=0}^{4} r(s_t, a_t)\Big].$$

At first, it might seem weird that we're not carrying the current digit vector in the state. Like, shouldn't you care about the current state of the digits vector? In truth it's easier to just view this problem as a sequence of rewards multiplied by the digit's positional value. The only real state information we need to carry from round to round is the mask of filled positions and the current digit. This also drastically reduces the size of the state space, and allows our agent to converge on the optimal policy exponentially faster (especially as the number of rounds increases).

# Problem 2

### Finite-horizon DP formulation and solution

We have a finite horizon undiscounted MDP with 3 rounds. The state encodes the customer's excitement for the two products $s = (x_1, x_2) \in \{E, U\}^2$ at the start of each round and the action space $a \in \{1, 2\}$ represents the seller's recommendation.

One could consider adding the round number $t$ to the state, but it is not necessary since the horizon is fixed and known.

**Reward.**

$$r(s,a) = \begin{cases} 1, & a = 1 \text{ and } x_1 = E, \\ 2, & a = 2 \text{ and } x_2 = E, \\ 0, & \text{otherwise.} \end{cases}$$

**Transition model.** At the end of the round, only the recommended product's excitement may flip:

$$\Pr(x_1' = \neg x_1 \mid a = 1) = 0.1, \qquad \Pr(x_2' = \neg x_2 \mid a = 2) = 0.5,$$

and the other component stays unchanged.

**Bellman Equation.** Let $V_k(s)$ be the optimal expected revenue with $k$ rounds remaining (so $V_0 \equiv 0$). Then for $k \geq 1$,

$$V_k(s) = \max_{a \in \{1,2\}} \big[ r(s,a) + \mathbf{E}\big[\, V_{k-1}(s') \mid s, a \,\big] \big].$$

*Intuition / Strategy* Product #2 offers a higher reward per purchase and over the long run, it will be available at the same frequency as product #1 (both products have symmetric flip probabilities). In an infinite horizon setting, the optimal policy is to always recommend product #2. However, in a finite horizon setting, it can be higher EV to recommend product #1 when product #2 is unavailable (state $EU$), and there are few turns left.

**Backward induction.** The state space is $\{EE, EU, UE, UU\}$. Let's calculate $V_k$ and an optimal action $\pi_k^*(s)$.

$k = 1$ (one round left): At this point, we're just trying to maximize immediate reward. So, if the second product is available, we recommend it. If not, recommend the first product. If neither is available, the policy is indifferent.

| $s$ | $EE$ | $EU$ | $UE$ | $UU$ |
|---|---|---|---|---|
| $V_1(s)$ | 2 | 1 | 2 | 0 |
| $\pi_1^*(s)$ | 2 | 1 | 2 | $\varnothing$ |

$\underline{k = 2:}$ We now have to consider the final round as well. Our strategy will be similar to $k = 1$, but we have a preference to recommend product #2 because it has a higher reward.

| $s$ | $EE$ | $EU$ | $UE$ | $UU$ |
|---|---|---|---|---|
| $V_2(s)$ | 3.5 | 1.9 | 3.0 | 1.0 |
| $\pi_2^*(s)$ | 2 | 1 | 2 | 2 |

$\underline{k = 3:}$ Our strategy is the same as $k = 2$

| $s$ | $EE$ | $EU$ | $UE$ | $UU$ |
|---|---|---|---|---|
| $V_3(s)$ | 4.7 | 2.81 | 4.0 | 2.0 |
| $\pi_3^*(s)$ | 2 | 1 | 2 | 2 |

**Optimal 3 round policy.** From any state with $k \in \{2, 3\}$ rounds left, the same mapping applies:

$$EE \to 2, \quad EU \to 1, \quad UE \to 2, \quad UU \to 2,$$

with $k = 1$ agreeing except that $UU$ is a tie. Starting from the given initial state $EE$ (both eager), the optimal expected total revenue is

$$W_3(EE) = \boxed{4.7}.$$

# Problem 3

## Part (a)

Lets first restate the relationship between average reward and discounted value from our lecture slides:

$$\rho^\pi(s) \;=\; \lim_{\gamma\to 1}(1-\gamma)\, V_\gamma^\pi(s)$$

Bias is defined as

$$h^\pi(s) \;=\; \lim_{T\to\infty} \mathbf{E}\Big[\sum_{t=1}^{T}\big(r_t - \rho^\pi(s_t)\big)\;\Big|\; s_1 = s\Big],$$

and the slides show

$$h^\pi(s) - h^\pi(s') \;=\; \lim_{\gamma\to 1}\Big(V_\gamma^\pi(s) - V_\gamma^\pi(s')\Big).$$

There are two states $s_1, s_2$; in each, two actions $a_1, a_2$ with deterministic transitions and rewards:

$$s_1 \xrightarrow[a_1]{\$4} s_1, \qquad s_1 \xrightarrow[a_2]{\$3} s_2,$$
$$s_2 \xrightarrow[a_1]{\$7} s_2, \qquad s_2 \xrightarrow[a_2]{\$5} s_1.$$

I will denote the four deterministic stationary policies by $\pi^{ij}$ where $i$ is the action in $s_1$ and $j$ is the action in $s_2$. So for example, $\pi^{12}$ will loop in $s_1$ and transition from $s_2$ to $s_1$.

Lets start in state $s_1$ and calculate the discounted values for each policy.

$V_\gamma^{\pi^{11}}(s_1) = 4 + 4\gamma + 4\gamma^2 + ... = \frac{4}{1-\gamma}$

$V_\gamma^{\pi^{12}}(s_1) = 4 + 4\gamma + 4\gamma^2 + ... = \frac{4}{1-\gamma}$

$V_\gamma^{\pi^{21}}(s_1) = 3 + 7\gamma + 7\gamma^2 + ... = 3 + \gamma\frac{7}{1-\gamma}$

$V_\gamma^{\pi^{22}}(s_1) = 3 + 5\gamma + 3\gamma^2 + 5\gamma^3 + ... = \frac{3+5\gamma}{1-\gamma^2}$

Lets move to state $s_2$

$V_\gamma^{\pi^{11}}(s_2) = 7 + 7\gamma + 7\gamma^2 + ... = \frac{7}{1-\gamma}$

$V_\gamma^{\pi^{12}}(s_2) = 5 + 4\gamma + 4\gamma^2 + ... = 5 + \gamma\frac{4}{1-\gamma}$

$V_\gamma^{\pi^{21}}(s_2) = 7 + 7\gamma + 7\gamma^2 + ... = \frac{7}{1-\gamma}$

$V_\gamma^{\pi^{22}}(s_2) = 5 + 3\gamma + 5\gamma^2 + 3\gamma^3 + ... = \frac{5+3\gamma}{1-\gamma^2}$

Moving on to average rewards, we can see that $\pi^{21}$ is the only policy that yields an average reward of 7 in both states. $\pi^{11}$ yields 4 in $s_1$ and 7 in $s_2$. $\pi^{12}$ and $\pi^{22}$ both yield 4 in both states.

## Part (b)

**(1) -1 discount optimal (average-reward optimal):** At $s_1$, the only policy that maximizes average reward is $\pi^{21}$. At $s_2$, both $\pi^{11}$ and $\pi^{21}$ maximize average reward. However, to be optimal at both states simultaneously, we must choose $\pi^{21}$. So the only -1 discount optimal policy is $\pi^{21}$.

**(2) 0 discount optimal (bias optimal):** Apologies in advance for the brevity, but we can quickly see that at $s_2$, both $\pi^{11}$ and $\pi^{21}$ maximize the gain, regardless of discount applied. We'll conclude that $\pi^{21}$ is the optimal 0-discount policy for the same reason as above.

The situation is more interesting at $s_1$. Depending on the discount factor, it might be better to choose $\pi^{11}$ or $\pi^{21}$. Lets setup the inequality:

$$\begin{aligned} V_\gamma^{\pi^{21}}(s_1) - V_\gamma^{\pi^{11}}(s_1) &\geq 0 \\ 3 + \gamma\frac{7}{1-\gamma} - \frac{4}{1-\gamma} &\geq 0 \\ \frac{3(1-\gamma) + 7\gamma - 4}{1-\gamma} &\geq 0 \\ 3 - 3\gamma + 7\gamma - 4 &\geq 0 \\ 4\gamma - 1 &\geq 0 \\ \gamma &\geq 0.25 \end{aligned}$$

So for $\gamma \geq 0.25$, $\pi^{21}$ is optimal at $s_1$. For $\gamma < 0.25$, $\pi^{11}$ is optimal at $s_1$.

**(3) 1 and $\infty$ discount optimal:** The same logic applies as in (2). $\pi^{21}$ is optimal at $s_2$ for all $\gamma$. At $s_1$, $\pi^{21}$ is optimal for $\gamma \geq 0.25$. So the only 1-discount optimal policy is $\pi^{21}$.

## Part (c)

We have shown that when $\gamma \geq 0.25$, $\pi^{21}$ is n-discount optimal for all $n \geq -1$. When $\gamma < 0.25$, $\pi^{11}$ is optimal at $s_1$ but not at $s_2$. So, our blackwell optimal policy is $\pi^{21}$, and the smallest discount factor that suffices is $\gamma^*(s_1) = 0.25$.

$$\boxed{\text{Blackwell optimal policy } = \pi^{21}, \quad \gamma^*(s_1) = 0.25,\ \gamma^*(s_2) = 0}.$$

# HW1_nb

October 2, 2025

[1]:
```
%reload_ext autoreload
%autoreload 2
# %autoreload 1
# %aimport from kret_studies import *
# %aimport from kret_studies.notebook_imports import *
# %load_ext fireducks.pandas # linux only for now
```

[2]:
```
from kret_studies import *
from kret_studies.notebook import *
from kret_studies.complex import *

logger = get_notebook_logger()
```

```
Loaded environment variables from
/Users/Akseldkw/Desktop/Columbia/ORCS4529/.env.
/Users/Akseldkw/coding/kretsinger/data/nb_log.log
```

[12]:
```
gamma = 0.5
actions = (0.0, 0.5, 1.0)
states = (1, 2)


def reward(s: t.Literal[1, 2], action: float) -> float:
    if s == 1:
        return -action

    return (action / 12) - 1


def transition(s: t.Literal[1, 2], action: float) -> t.Tuple[float, float]:
    """
    Return the probability of staying in the same state or flipping to the next␣
 ↪state
    """
    if s == 1:
        p_1 = action**2 / 2
        return p_1, 1 - p_1
```

```
    p_2 = action**2 / 4
    return 1 - p_2, p_2
```

### 0.1 Quick Heuristic Visualization

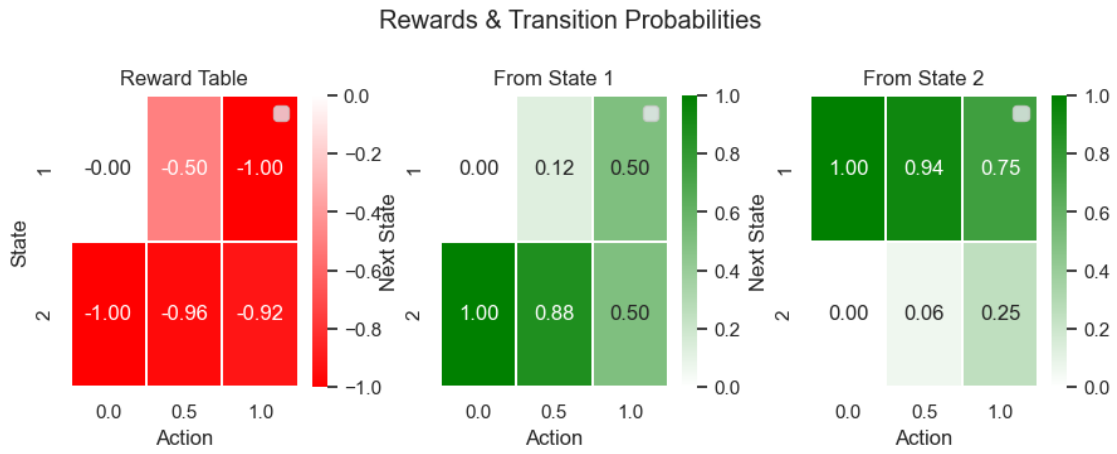
[19]:
```
rewards = pd.DataFrame(
    index=states,
    columns=actions,
    data=[[reward(s, a) for a in actions] for s in states],
).round(3)
transitions = {
    s: pd.DataFrame(
        index=states,
        columns=actions,
        data=[[transition(s, a)[i] for a in actions] for i in␣
 ↪range(len(states))],
    )
    for s in states
}
rewards.index.name = "State"
rewards.columns.name = "Action"
# rewards.title = "Reward Table"

for s in transitions:
    transitions[s].index.name = "Next State"
    transitions[s].columns.name = "Action"
    # transitions[s].title = f"Transition Probabilities (from state {s})"
```

[20]:
```
fig, ax = uks_mpl.subplots(3, 1, 3, 3)
fig.subplots_adjust(wspace=0.3)
fig.suptitle("Rewards & Transition Probabilities", y=1.15)

uks_mpl.heatmap_df(rewards, ax=ax[0])
ax[0].set_title("Reward Table")
uks_mpl.heatmap_df(transitions[1], ax=ax[1])
ax[1].set_title("From State 1")
uks_mpl.heatmap_df(transitions[2], ax=ax[2])
ax[2].set_title("From State 2")
fig
```

[20]:




## 0.2 Define Greedy Update Function

[21]:
```python
def bellman_update(v: np.ndarray, s: t.Literal[1, 2]) -> float:
    """
    Perform a Bellman update for state s given value function v.
    """
    action_values = [
        reward(s, a) + gamma * sum(transition(s, a)[i] * v[i] for i in range(2))
        for a in actions
    ]
    best_action = actions[np.argmax(action_values)]
    assert best_action in actions
    return max(action_values)


def policy_update(v: np.ndarray, s: t.Literal[1, 2]) -> float:
    """
    Perform a policy update for state s given value function v.
    """
    action_values = [
        reward(s, a) + gamma * sum(transition(s, a)[i] * v[i] for i in range(2))
        for a in actions
    ]
    best_action = actions[np.argmax(action_values)]
    assert best_action in actions
    # pi[s - 1] = best_action
    return best_action
```

### 0.3 Iterative Policy Evaluation

[22]:
```python
def value_iteration(
    v_init: np.ndarray | None, max_iterations: int | None = None, print_first_k:
 ↪ int = 4
) -> np.ndarray:
    """
    Perform value iteration until convergence.
    """
    v = np.zeros(2) if v_init is None else v_init.copy()
    k = 0
    max_iter = 1000 if max_iterations is None else max_iterations
    while True:
        new_v = np.array([bellman_update(v, s) for s in states])
        if k < print_first_k:
            print(f"Iter {k+1}: V = {new_v}")
        if np.max(np.abs(new_v - v)) < 1e-6:
            break
        v = new_v
        k += 1
        if max_iter != -1 and k >= max_iter:
            print(f"Stopped after reaching max_iter={max_iter}")
            break
    print(f"Converged after {k+1} iterations.")
    return v


def policy_iteration(
    v_init: np.ndarray | None, max_iterations: int | None = None, print_first_k:
 ↪ int = 4
) -> t.Tuple[np.ndarray, np.ndarray]:
    """
    Perform policy iteration until convergence.
    """
    v = np.zeros(2) if v_init is None else v_init.copy()
    pi = np.array([0.0, 0.0])
    k = 0
    max_iter = 1000 if max_iterations is None else max_iterations
    while True:
        new_v = np.array([bellman_update(v, s) for s in states])
        new_pi = np.array([policy_update(new_v, s) for s in states])
        if k < print_first_k:
            print(f"Iter {k+1}: V = {new_v}, pi = {new_pi}")
        if np.max(np.abs(new_v - v)) < 1e-6 and np.all(new_pi == pi):
            break
        v = new_v
        pi = new_pi
```

```
        k += 1
        if max_iter != -1 and k >= max_iter:
            print(f"Stopped after reaching max_iter={max_iter}")
            break
    print(f"Converged after {k+1} iterations.")
    return v, pi
```

### 0.4 Results

[23]:
```
values = value_iteration(None)
print("Optimal values:", values)
```

```
Iter 1: V = [ 0.         -0.91666667]
Iter 2: V = [-0.45833333 -0.98697917]
Iter 3: V = [-0.49348958 -1.20402018]
Iter 4: V = [-0.60201009 -1.22728221]
Converged after 21 iterations.
Optimal values: [-0.65248136 -1.30496348]
```

[24]:
```
policies = policy_iteration(None)
print("Optimal policies:", policies)
```

```
Iter 1: V = [ 0.         -0.91666667], pi = [0.  0.5]
Iter 2: V = [-0.45833333 -0.98697917], pi = [0.  0.5]
Iter 3: V = [-0.49348958 -1.20402018], pi = [0.  0.5]
Iter 4: V = [-0.60201009 -1.22728221], pi = [0.  0.5]
Converged after 21 iterations.
Optimal policies: (array([-0.65248136, -1.30496348]), array([0. , 0.5]))
```

[ ]:

[ ]: