Universidad del Valle de Guatemala - UVG
Facultad de Ingeniería - Computación
Curso: CC3104 - Aprendizaje por Refuerzo **Sección:** 10
Laboratorio 4: Métodos de Monte Carlo

Autores:

- Diego Alexander Hernández Silvestre - **21270**
- Linda Inés Jiménez Vides - **21169**
- Mario Antonio Guerra Morales - **21008**

Task 1

1. ¿Cómo afecta la elección de la estrategia de exploración (exploring starts vs soft policy) a la precisión de la evaluación de políticas en los métodos de Monte Carlo? Considere la posibilidad de comparar el desempeño de las políticas evaluadas con y sin explorar los inicios o con diferentes niveles de exploración en políticas blandas.

- La elección de estrategia de exploración sí tiene un impacto directo en la evaluación de políticas al usar los métodos de Monte Carlo. Si se utiliza Exploring Starts, se garantiza una mayor diversidad en las trayectorias debido a que se puede iniciar desde distintas combinaciones de estados y de acciones. Mientras que, con Soft Policies se añade un equilibrio entre la exploración y la explotación. Cuando se mantiene una probabilidad de selección de imágenes aleatorias, se añade la posibilidad de seguir explorando nuevas y aleatorias trayectorias, aún priorizando acciones de mayor valor.

2. En el contexto del aprendizaje de Monte Carlo fuera de la póliza, ¿cómo afecta la razón de muestreo de importancia a la convergencia de la evaluación de políticas? Explore cómo la razón de muestreo de importancia afecta la estabilidad y la convergencia.

- La razón de muestreo de importancia ajusta las estimaciones de retorno para que la política objetivo se refleje utilizando datos de manera diferente. Esta afecta al permitirse aprender sobre una política objetivo sin hacer un seguimiento directo, pero también es capaz de hacer inestables las razones si tanto el comportamiento como el objetivo difieren demasiado.

3. ¿Cómo puede el uso de una soft policy influir en la eficacia del aprendizaje de políticas óptimas en comparación con las políticas deterministas en los métodos de Monte Carlo? Compare el desempeño y los resultados de aprendizaje de las políticas derivadas de estrategias épsilon-greedy con las derivadas de políticas deterministas.

- Una soft policy puede influir con una mayor exploración, esto para ayudar al descubrimiento de acciones que no son efectivas a corto plazo pero sí a un largo plazo. Mientras que, una política determinista elegirá la misma acción siempre, lo que puede llegar a darse el riesgo de no hacer una correcta exploración.

4. ¿Cuáles son los posibles beneficios y desventajas de utilizar métodos de Monte Carlo off-policy en comparación con los on-policy en términos de eficiencia de la muestra, costo computacional y velocidad de aprendizaje?

- Los métodos de Monte Carlo off-policy ofrecen una mejor eficiencia al existir la posibilidad de usar datos previos o simulados, pero ello contempla un mayor costo computacional. Sin embargo, esto a su vez permite una gran flexibilidad y con ello una velocidad de aprendizaje considerable, si es que las políticas no difieren mucho con el objetivo.

Task 2

En este ejercicio, simulará un sistema de gestión de inventarios para una pequeña tienda minorista. La tienda tiene como objetivo maximizar las ganancias manteniendo niveles óptimos de existencias de diferentes productos. Utilizará métodos de Monte Carlo para la evaluación de pólizas, exploring starts, soft policies y aprendizaje off-policy para estimar el valor de diferentes estrategias de gestión de inventarios. Su objetivo es implementar una solución en Python y responder preguntas específicas en función de los resultados.

Definición del entorno

- Utilice el ambiente dado más adelante para simular el entorno de la tienda. Considere que:
  - El estado representa los niveles de existencias actuales de los productos.
  - Las acciones representan decisiones sobre cuánto reponer de cada producto

In [1]:
```
import numpy as np
import random

class InventoryEnvironment:
    def __init__(self):
        self.products = ['product_A', 'product_B']
        self.max_stock = 10
        self.demand = {'product_A': [0, 1, 2], 'product_B': [0, 1, 2]}
        self.restock_cost = {'product_A': 5, 'product_B': 7}
        self.sell_price = {'product_A': 10, 'product_B': 15}
        self.state = None

    def reset(self):
        self.state = {product: random.randint(0, self.max_stock) for product in self.products}
        return self.state

    def step(self, action):
        reward = 0
        for product in self.products:
            stock = self.state[product]
            restock = action[product]
            self.state[product] = min(self.max_stock, stock + restock)
            demand = random.choice(self.demand[product])
            sales = min(demand, self.state[product])
            self.state[product] -= sales
            reward += sales * self.sell_price[product] - restock * self.restock_cost[product]
        return self.state.copy(), reward
```

Generación de episodios

- Cada episodio representa una serie de días en los que la tienda sigue una política de inventario específica.
- Debe recopilar datos para varios episodios y registrar las recompensas (ganancias) de cada día

In [2]:
```
def generate_episode(env, policy, num_days=10):
    episode = []
    state = env.reset()

    for _ in range(num_days):
        # Elegir acción según la política
        action = policy(state)
        # Ejecutar acción en el entorno
        next_state, reward = env.step(action)
        # Guardar transición: estado, acción, recompensa
        episode.append((state.copy(), action.copy(), reward))
        # Actualizar estado actual
        state = next_state

    return episode
```

In [3]:
```
def random_policy(state):
    return {
        'product_A': random.randint(0, 3),
        'product_B': random.randint(0, 3)
    }
```

In [4]:
```
def generate_episodes(env, policy, num_episodes=100, num_days=10):
    episodes = []
    for _ in range(num_episodes):
        episode = generate_episode(env, policy, num_days)
        episodes.append(episode)
    return episodes
```

In [5]:
```
env = InventoryEnvironment()
episodes = generate_episodes(env, random_policy, num_episodes=3, num_days=5)

for i, ep in enumerate(episodes):
    print(f"\n--- Episodio {i+1} ---")
    for day, (state, action, reward) in enumerate(ep):
        print(f"Día {day+1}: Estado={state}, Acción={action}, Recompensa={reward}")
```

```
--- Episodio 1 ---
Día 1: Estado={'product_A': 10, 'product_B': 10}, Acción={'product_A': 3, 'product_B': 3}, Recompensa=-36
Día 2: Estado={'product_A': 10, 'product_B': 10}, Acción={'product_A': 0, 'product_B': 1}, Recompensa=43
Día 3: Estado={'product_A': 8, 'product_B': 8}, Acción={'product_A': 0, 'product_B': 1}, Recompensa=23
Día 4: Estado={'product_A': 8, 'product_B': 7}, Acción={'product_A': 3, 'product_B': 2}, Recompensa=-9
Día 5: Estado={'product_A': 8, 'product_B': 9}, Acción={'product_A': 0, 'product_B': 1}, Recompensa=33

--- Episodio 2 ---
Día 1: Estado={'product_A': 1, 'product_B': 3}, Acción={'product_A': 1, 'product_B': 3}, Recompensa=-6
Día 2: Estado={'product_A': 1, 'product_B': 3}, Acción={'product_A': 0, 'product_B': 3}, Recompensa=4
Día 3: Estado={'product_A': 0, 'product_B': 5}, Acción={'product_A': 0, 'product_B': 0}, Recompensa=0
Día 4: Estado={'product_A': 0, 'product_B': 5}, Acción={'product_A': 2, 'product_B': 0}, Recompensa=30
Día 5: Estado={'product_A': 1, 'product_B': 3}, Acción={'product_A': 3, 'product_B': 2}, Recompensa=11

--- Episodio 3 ---
Día 1: Estado={'product_A': 7, 'product_B': 9}, Acción={'product_A': 3, 'product_B': 2}, Recompensa=-14
Día 2: Estado={'product_A': 7, 'product_B': 9}, Acción={'product_A': 0, 'product_B': 1}, Recompensa=-7
Día 3: Estado={'product_A': 7, 'product_B': 10}, Acción={'product_A': 2, 'product_B': 1}, Recompensa=-2
Día 4: Estado={'product_A': 9, 'product_B': 9}, Acción={'product_A': 2, 'product_B': 1}, Recompensa=23
Día 5: Estado={'product_A': 9, 'product_B': 8}, Acción={'product_A': 1, 'product_B': 1}, Recompensa=18
```

Exploring Starts

- Implemente explorar inicios para garantizar un conjunto diverso de estados y acciones iniciales

In [6]:
```
def generate_episode_exploring_starts(env, policy, num_days=10):
    # Estado inicial aleatorio
    state = {
        product: random.randint(0, env.max_stock)
        for product in env.products
    }
    env.state = state.copy()

    episode = []

    for day in range(num_days):
        # Acción inicial aleatoria solo en el primer paso
        if day == 0:
            action = {
                product: random.randint(0, 3)
                for product in env.products
            }
        else:
            action = policy(state)

        next_state, reward = env.step(action)
        episode.append((state.copy(), action.copy(), reward))
        state = next_state

    return episode
```

In [7]:
```
def generate_episodes_exploring_starts(env, policy, num_episodes=5, num_days=7):
    episodes = []
    for _ in range(num_episodes):
        episode = generate_episode_exploring_starts(env, policy, num_days)
        episodes.append(episode)
    return episodes
```

In [8]:
```
episodes_es = generate_episodes_exploring_starts(env, random_policy, num_episodes=3, num_days=5)

for i, ep in enumerate(episodes_es):
    print(f"\n--- Episodio (Exploring Starts) {i+1} ---")
    for day, (state, action, reward) in enumerate(ep):
        print(f"Día {day+1}: Estado={state}, Acción={action}, Recompensa={reward}")
```

```
--- Episodio (Exploring Starts) 1 ---
Día 1: Estado={'product_A': 7, 'product_B': 8}, Acción={'product_A': 0, 'product_B': 3}, Recompensa=-21
Día 2: Estado={'product_A': 7, 'product_B': 10}, Acción={'product_A': 3, 'product_B': 3}, Recompensa=-26
Día 3: Estado={'product_A': 9, 'product_B': 10}, Acción={'product_A': 0, 'product_B': 3}, Recompensa=-6
Día 4: Estado={'product_A': 9, 'product_B': 9}, Acción={'product_A': 0, 'product_B': 2}, Recompensa=21
Día 5: Estado={'product_A': 7, 'product_B': 9}, Acción={'product_A': 1, 'product_B': 3}, Recompensa=-16

--- Episodio (Exploring Starts) 2 ---
Día 1: Estado={'product_A': 2, 'product_B': 6}, Acción={'product_A': 0, 'product_B': 0}, Recompensa=35
Día 2: Estado={'product_A': 0, 'product_B': 5}, Acción={'product_A': 2, 'product_B': 2}, Recompensa=-24
Día 3: Estado={'product_A': 2, 'product_B': 7}, Acción={'product_A': 0, 'product_B': 2}, Recompensa=-4
Día 4: Estado={'product_A': 1, 'product_B': 9}, Acción={'product_A': 0, 'product_B': 0}, Recompensa=10
Día 5: Estado={'product_A': 0, 'product_B': 9}, Acción={'product_A': 3, 'product_B': 3}, Recompensa=-36

--- Episodio (Exploring Starts) 3 ---
Día 1: Estado={'product_A': 8, 'product_B': 10}, Acción={'product_A': 1, 'product_B': 1}, Recompensa=-12
Día 2: Estado={'product_A': 9, 'product_B': 10}, Acción={'product_A': 1, 'product_B': 3}, Recompensa=-26
Día 3: Estado={'product_A': 10, 'product_B': 10}, Acción={'product_A': 1, 'product_B': 3}, Recompensa=-11
Día 4: Estado={'product_A': 10, 'product_B': 9}, Acción={'product_A': 2, 'product_B': 3}, Recompensa=9
Día 5: Estado={'product_A': 9, 'product_B': 8}, Acción={'product_A': 1, 'product_B': 1}, Recompensa=23
```

Soft Policies

- Utilice una soft policy (como epsilon-greedy) para garantizar un equilibrio entre la exploración y la

explotación.

In [ ]:
```

```

Aprendizaje off-policy

- Implemente el aprendizaje off-policy para evaluar una política objetivo utilizando datos generados

por una política de comportamiento diferente.

In [ ]:
```

```

1. ¿Cuál es el valor estimado de mantener diferentes niveles de existencias para cada producto?
2. ¿Cómo afecta el valor epsilon en la política blanda al rendimiento?
3. ¿Cuál es el impacto de utilizar el aprendizaje fuera de la política en comparación con el aprendizaje dentro de la política?