In [ ]:

```
import numpy as np
import gymnasium as gym  # ✅ use Gymnasium

# -------------------------------
# Create Taxi environment
# -------------------------------
env = gym.make("Taxi-v3")

print("Number of states:", env.observation_space.n)  # 500
print("Number of actions:", env.action_space.n)      # 6

# Parameters
gamma = 0.9
theta = 1e-6

# -------------------------------
# Value Iteration
# -------------------------------
def value_iteration(env, gamma=0.9, theta=1e-6):
    value_table = np.zeros(env.observation_space.n)

    while True:
        delta = 0
        for state in range(env.observation_space.n):
            action_values = np.zeros(env.action_space.n)
            for action in range(env.action_space.n):
                for prob, next_state, reward, done in env.unwrapped.P[state][a
                    action_values[action] += prob * (reward + gamma * value_ta
            best_action_value = np.max(action_values)
            delta = max(delta, abs(value_table[state] - best_action_value))
            value_table[state] = best_action_value
        if delta < theta:
            break

    # Derive policy
    policy = np.zeros(env.observation_space.n, dtype=int)
    for state in range(env.observation_space.n):
        action_values = np.zeros(env.action_space.n)
        for action in range(env.action_space.n):
            for prob, next_state, reward, done in env.unwrapped.P[state][actio
                action_values[action] += prob * (reward + gamma * value_table[
        policy[state] = np.argmax(action_values)

    return policy, value_table

# -------------------------------
# Policy Evaluation
# -------------------------------
def policy_evaluation(policy, env, gamma=0.9, theta=1e-6):
    value_table = np.zeros(env.observation_space.n)
    while True:
        delta = 0
        for state in range(env.observation_space.n):
```

```
            v = 0
            action = policy[state]
            for prob, next_state, reward, done in env.unwrapped.P[state][actic
                v += prob * (reward + gamma * value_table[next_state])
            delta = max(delta, abs(value_table[state] - v))
            value_table[state] = v
        if delta < theta:
            break
    return value_table

# ------------------------------
# Policy Improvement
# ------------------------------
def policy_improvement(value_table, policy, env, gamma=0.9):
    policy_stable = True
    for state in range(env.observation_space.n):
        old_action = policy[state]
        action_values = np.zeros(env.action_space.n)
        for action in range(env.action_space.n):
            for prob, next_state, reward, done in env.unwrapped.P[state][actic
                action_values[action] += prob * (reward + gamma * value_table[
        policy[state] = np.argmax(action_values)
        if old_action != policy[state]:
            policy_stable = False
    return policy, policy_stable

# ------------------------------
# Policy Iteration
# ------------------------------
def policy_iteration(env, gamma=0.9, theta=1e-6):
    policy = np.random.choice(env.action_space.n, size=env.observation_space.r
    value_table = np.zeros(env.observation_space.n)

    while True:
        value_table = policy_evaluation(policy, env, gamma, theta)
        policy, policy_stable = policy_improvement(value_table, policy, env, g
        if policy_stable:
            return policy, value_table

# ------------------------------
# Run Both Algorithms
# ------------------------------
print("\n Running Policy Iteration...")
pi_policy, pi_value = policy_iteration(env, gamma, theta)
print("Policy Iteration: Optimal Value Function shape =", pi_value.shape)
print("Policy Iteration: Optimal Policy shape =", pi_policy.shape)

print("\n Running Value Iteration...")
vi_policy, vi_value = value_iteration(env, gamma, theta)
print("Value Iteration: Optimal Value Function shape =", vi_value.shape)
print("Value Iteration: Optimal Policy shape =", vi_policy.shape)

# ------------------------------
```

```python
# Quick check: same optimal policy?
# -------------------------------
print("\n Do both methods give same optimal policy? ->", np.array_equal(pi_pol

# -------------------------------
# Demonstration: play one episode using optimal policy
# -------------------------------
state, _ = env.reset()
done = False
total_reward = 0
steps = 0

print("\n Demo run with Optimal Policy:")
while not done and steps < 20:
    action = pi_policy[state]
    state, reward, terminated, truncated, _ = env.step(action)
    done = terminated or truncated
    total_reward += reward
    steps += 1
    env.render()

print("Episode finished in", steps, "steps with reward:", total_reward)
```

```
Number of states: 500
Number of actions: 6

 Running Policy Iteration...
Policy Iteration: Optimal Value Function shape = (500,)
Policy Iteration: Optimal Policy shape = (500,)

 Running Value Iteration...
Value Iteration: Optimal Value Function shape = (500,)
Value Iteration: Optimal Policy shape = (500,)

 Do both methods give same optimal policy? -> True

 Demo run with Optimal Policy:
Episode finished in 12 steps with reward: 9
```

```
/usr/local/lib/python3.12/dist-packages/gymnasium/envs/toy_text/taxi.py:443: Us
erWarning: WARN: You are calling render method without specifying any render mo
de. You can specify the render_mode at initialization, e.g. gym.make("Taxi-v3",
render_mode="rgb_array")
  gym.logger.warn(
```