Aim

Evaluate the state-value function

$v\pi(s)$

v

π

(s) for a given policy

π

π in a warehouse grid MDP, where states are robot positions, actions are four-directional moves, rewards include +2 for picking items, +5 for reaching the goal, and -2 for obstacles, using iterative policy evaluation.

Algorithm

Iterative Policy Evaluation (Dynamic Programming):

1. Initialize
2. $v(s)=0$
3. $v(s)=0$ for all states
4. s
5. s.
6. For each iteration until convergence (
7. $\max s|v(s)-v'(s)|<\theta$
8. $\max$
9. s
10. $|v(s)-v$
11. $'$
12. $(s)|<\theta$):
    - For each state
    - s
    - s:
    - $v(s)\leftarrow\sum a\pi(a|s)\sum s',rp(s',r|s,a)[r+\gamma v(s')]$
    - $v(s)\leftarrow$

- a
- $\sum$
- $\pi(a|s)$
- s
- $'$
- $,r$
- $\sum$
- $p(s$
- $'$
- $,r|s,a)[r+\gamma v(s$
- $'$
    - $)]$

13. Return
14. vπ
15. v
16. π
17. .

For deterministic policies/environments, simplify to

v(s)←r(s,π(s))+γv(s′)

$v(s)\leftarrow r(s,\pi(s))+\gamma v(s$

$'$

$)$.

Environment Setup

A 5x5 grid is used: positions (0,0) to (4,4). States are (row,col) tuples. Obstacles at (2,2),(3,1). Item at (1,3) (+2 reward). Goal at (4,4) (+5). Moves: 0=up,1=right,2=down,3=left. Deterministic transitions with -1 step cost. Policy: prefer right (1), then down (2).

Code

python

```python
import numpy as np

# Grid world parameters
grid_size = 5
obstacles = [(2,2), (3,1)]
item_pos = (1,3)
goal_pos = (4,4)
gamma = 0.9  # discount factor
theta = 1e-6  # convergence threshold
policy = {  # deterministic policy: action index
0=up,1=right,2=down,3=left
    (i,j): 1 if j < 3 else 2 for i in range(grid_size) for j in
range(grid_size)
}
policy.update({obstacles[0]: 0, obstacles[1]: 0})  # stay at
obstacles

actions = [(-1,0), (0,1), (1,0), (0,-1)]  # up, right, down, left

def is_valid(state):
    i, j = state
    return 0 <= i < grid_size and 0 <= j < grid_size and state
not in obstacles

def step(state, action_idx):
    di, dj = actions[action_idx]
    next_state = (state[0] + di, state[1] + dj)
    if not is_valid(next_state):
        return state, -2  # hit obstacle
    reward = -1  # step cost
    if next_state == item_pos:
        reward += 2
    if next_state == goal_pos:
        reward += 5
    return next_state, reward

# States list
states = [(i,j) for i in range(grid_size) for j in
range(grid_size) if (i,j) not in obstacles]

# Iterative Policy Evaluation
V = {s: 0 for s in states}
iteration = 0
while True:
    delta = 0
    for s in states:
        a = policy[s]
        s_next, r = step(s, a)
        v_new = r + gamma * V[s_next]
        delta = max(delta, abs(V[s] - v_new))
        V[s] = v_new
    iteration += 1
    if delta < theta:
        break

# Display value function (top 10 states by value)
sorted_states = sorted(V.items(), key=lambda x: x[1],
reverse=True)[:10]
print("Top 10 state values under policy:")
for state, value in sorted_states:
    print(f"State {state}: {value:.3f}")
print(f"Converged in {iteration} iterations.")
```